[90] N. Vorobjev. New derivability algorithm in the constructive propositional calculus. (In Russian). In *Proceedings of the Steklov institute of Mathematics (Trudy), v.52*, pages 193–225, 1958.

[91] P. Wadler. Is there a use for linear logic? To Appear ACM/IFIP PEPM, 1991.

[92] P. Wadler. Linear types can change the world! IFIP TC 2 Conf. on Prog. Concepts and Methods, 1991.

[93] P. Wadler. There's no substitute for linear logic. Draft, 1991.

[94] M. Wajsberg. Untersuchungen uber den Aussagenkalkul von A. Heyting. *Wiadomosci Matematyczne 46*, pages 45–101, 1938.

[95] D. Wakeling and C. Runciman. Linearity and laziness. In *5th Conf. on Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science 523*, New York, 1991. Springer-Verlag.

[96] S. Wray and J. Fairbairn. Non-strict languages - programming and implementation. *Computer Journal*, 32(2):142–151, 1989.

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

[78] H. Ono. Structural rules and a logical hierarchy. In: Mathematical Logic, ed by P. Petkov.

[79] R. Pliuskevicius. On a version of the constructive predicate calculus without structural rules. *Soviet Math. Doklady*, 6:416–419, 1965.

[80] E.L. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[81] V. Pratt. Event spaces and their linear logic. In *Second International Conference on Algebraic Methodology and Software Technology*, 1992.

[82] D. Prawitz. *Natural Deduction.* Almquist and Wiksell, Stockholm, 1965.

[83] V. Saraswat and P. Lincoln. Higher-order, linear, concurrent constraint programming. In *Proc. 20-th ACM Symp. on Principles of Programming Languages*, page Submitted, January 1993.

[84] A. Scedrov. A brief guide to linear logic. *Bulletin of the European Assoc. for Theoretical Computer Science*, 41:154–165, June 1990.

[85] R.A.G. Seely. Linear logic, *-autonomous categories, and cofree algebras. In: Contemporary Math. 92, Amer. Math. Soc., 1989.

[86] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

[87] A. Troelstra. *Lectures on Linear Logic.* Center for the Study of Language and Information, 1991.

[88] A. Urquhart. The undecidability of entailment and relevant implication. *Journal of Symbolic Logic*, 49:1059–1073, 1984.

[89] A. Urquhart. The complexity of decision procedures in relevance logic. Technical Report 217/89, Department of Computer Science, University of Toronto, 1989.

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

[68] S. Martini and A. Masini. An exponential-free interpretation of classical logic into linear logic. Draft, 1992.

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

[70] E. W. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1981.

[71] 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.

[72] D. Miller. Abstractions in logic programming. In P. Odifreddi, editor, *Logic and Computer Science*, pages 329–359. APIC Studies in Data Processing, Vol. 31, Academic Press, 1990.

[73] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 1990. Special Issue on the 2nd IEEE Symposium on Logic in Computer Science, 1987. To appear.

[74] R. Milner. The Standard ML core language. *Polymorphism*, 2(2), 1985. 28 pages. An earlier version appeared in Proc. 1984 ACM Symp. on Lisp and Functional Programming.

[75] M. Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Annals of Mathematics*, 74:3:437–455, 1961.

[76] G. Mints. Gentzen-type systems and resolution rules. Part I. Propositional logic. In P. Martin-Lof and G. Mints, editors, *COLOG-88*, pages 198–231. *Lecture Notes in Computer Science vol. 417, Springer*, 1990.

[77] G. Mints. Some Information on Linear Logic. Draft, 1991.

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

[56] J. Lambek. How to program an infinite abacus. *Canadian Math. Bulletin*, 4:295–302, 1961.

[57] P. Lincoln. Linear logic. *Sigact*, 23(2):29–37, Spring 1992.

[58] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[59] 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.

[60] 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.

[61] P. Lincoln and A. Scedrov. First Order Multiplicative-Additive Linear Logic is NEXPTIME-Complete. Draft, 1992.

[62] P. Lincoln, A. Scedrov, and N. Shankar. Linearizing intuitionistic implication. In *Proc. 6th IEEE Symp. on Logic in Computer Science*, 1991.

[63] P. Lincoln and T. Winkler. Constant-Only Multiplicative Linear Logic is NP-Complete. Draft, 1992.

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

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

[66] N. Martí-Oliet and J. Meseguer. An algebraic axiomatization of linear logic models. Technical Report SRI-CSL-89-11, SRI International, 1989.

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

[44] J. Hudelmaier. *Bounds for Cut Elimination in Intuitionistic Propositional Logic.* PhD thesis, Universitat Tubingen, 1989.

[45] R.J.M. Hughes. *The Design and Implementation of Programming Languages.* PhD thesis, PRG-40, Oxford, 1984.

[46] S.L.P. Jones. *The Implementation of Functional Programming Languages.* Prentice Hall, 1987.

[47] M. Kanovich. The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, Institute for Language, Logic, and Information, June 1991.

[48] M. Kanovich. The multiplicative fragment of linear logic is NP-complete. Email Message, 1991.

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

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

[51] S.C. Kleene. *Introduction to Metamathematics.* North-Holland, 1952.

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

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

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

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

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

[35] J.-Y. Girard and Y. Lafont. Linear logic and lazy computation. In *TAPSOFT '87, Volume 2*, pages 52–66. Springer LNCS 250, 1987.

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

[37] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: A modular approach to polynomial time computability. In S.R. Buss and P.J. Scott, editors, *Feasible Mathematics, A Mathematical Sciences Institute Workshop, Ithaca, New York, June, 1989*, pages 195–209. Birkhauser, Boston, 1990.

[38] G. Gonthier, M. Abadi, and J-J. Levy. Linear logic without boxes. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[39] 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.

[40] 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.

[41] 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.*

[42] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company, 1979.

[21] J. Chirimar, C. Gunter, and J. Riecke. Linear ml. In *Proc. ACM Symp. on Lisp and Functional Programming*, June 1992.

[22] P-L. Curien. Games and sequentiality. Linear Logic Mailing List, 1992. linear@cs.stanford.edu.

[23] H.B. Curry. *Foundations of Mathematical Logic*. McGraw-Hill, 1963.

[24] 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.

[25] J. Dunn. Relevence logic and entailment. In: Handbook of Philosophical Logic III, ed by D. Gabbay and F. Gunther, 1986.

[26] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. Manuscript, January 1991.

[27] J. Fairbairn and S. Wray. Tim: A simple, lazy abstract machine to execute supercombinators. In *3rd Conf. on Functional Programming and Computer Architecture, Lecture Notes in Computer Science 274*, New York, 1985. Springer-Verlag.

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

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

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

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

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

[9] 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.

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

[11] A. Avron. Some properties of linear logic proved by semantic methods. Technical Report 260/92, Eskenasy Institute of Computer Science, Tel-Aviv University, 1992.

[12] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North Holland, 1984.

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

[14] G. Bellin. Proof Nets for Multiplicative-Additive Linear Logic. Draft, 1991.

[15] J. Van Bentham. *Language in Action.* North-Holland, 1991.

[16] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. Term Assignment for Intuitionistic Linear Logic (Preliminary Report). Draft, 1992.

[17] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 81–94, January 1990.

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

[19] A.K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[20] A.K. Chandra and L.J. Stockmeyer. Alternation. In *Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science*, pages 98–108, 1976.

Bibliography

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 1991. Special Issue on the 1990 Workshop on Math. Found. Prog. Semantics. To appear.

[2] S. Abramsky. Tutorial on linear logic. Lecture Notes from Tutorial at ILPS, 1991.

[3] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[4] V. Abrusci. Sequent calculus for intuitionistic linear propositional logic. In: Mathematical Logic, ed by P. Petkov.

[5] Amiot. Decision Problems for Second Order Linear Logic Without Exponentials. Draft, 1990.

[6] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. Draft, 1991.

[7] 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.

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

$$\textbf{!WL}\quad \frac{\Delta \vdash u : !A \qquad \Sigma \vdash t : B}{\Sigma, \Delta \vdash \texttt{discard}\ u\ \texttt{in}\ t : B}$$

$$\textbf{!DL}\quad \frac{\Delta \vdash u : !A \qquad \Sigma, x : A \vdash t : B}{\Sigma, \Delta \vdash \texttt{read}\ \ \texttt{store}\ x\ \texttt{as}\ u\ \texttt{in}\ t : B}$$

$$\textbf{!CL}\quad \frac{\Delta \vdash u : !A \qquad \Sigma, x : !A, y : !A \vdash t : B}{\Sigma, \Delta \vdash \texttt{copy}\ x@y\ \texttt{as}\ u\ \texttt{in}\ t : B}$$

$$\textbf{!SR}\quad \frac{\Delta_i \vdash t_i : !B_i \qquad x_i : !B_i, !\Sigma \vdash u : A}{\Delta_1 \cdots \Delta_n, !\Sigma \vdash \texttt{store}\ u[t_1/x_1 \cdots t_n/x_n] : !A}$$

$$1\textbf{L}\quad \frac{\Delta \vdash u : 1 \qquad \Sigma \vdash t : A}{\Sigma, \Delta \vdash \texttt{let}\ 1\ \texttt{be}\ u\ \texttt{in}\ t : A}$$

$$1\textbf{R}\quad \frac{}{\vdash 1 : 1}$$

I
$$\frac{}{x : A \vdash x : A}$$

Subst
$$\frac{\Sigma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Sigma, \Gamma \vdash u[t/x] : B}$$

Exch. Left
$$\frac{\Sigma, x : A, y : B, \Delta \vdash u : A}{\Sigma, y : B, x : A, \Delta \vdash u : A}$$

A
$$\frac{\Delta \vdash u : (A \multimap B) \qquad \Sigma \vdash t : A}{\Sigma, \Delta \vdash (u\, t) : B}$$

$\multimap$**R**
$$\frac{\Sigma, x : A \vdash t : B}{\Sigma \vdash \lambda x.t : (A \multimap B)}$$

$\otimes$**L**
$$\frac{\Delta \vdash u : (A \otimes B) \quad \Sigma, x : A, y : B \vdash t : C}{\Sigma, \Delta \vdash \ \texttt{let}\ (x \times y)\ \texttt{be}\ u\ \texttt{in}\ t : C}$$

$\otimes$**R**
$$\frac{\Sigma \vdash u : A \qquad \Gamma \vdash t : B}{\Sigma, \Gamma \vdash (u \times t) : (A \otimes B)}$$

$\oplus$**L**
$$\frac{\Delta \vdash t : (A \oplus B) \quad \Sigma, x : A \vdash u : C \quad \Sigma, y : B \vdash v : C}{\Sigma, \Delta \vdash \ \texttt{case}\ t\ \texttt{of}\ inl(x) \Rightarrow u, inr(y) \Rightarrow v : C}$$

$\oplus$**R1**
$$\frac{\Sigma \vdash t : A}{\Sigma \vdash inl(t) : (A \oplus B)}$$

$\oplus$**R2**
$$\frac{\Sigma \vdash t : B}{\Sigma \vdash inr(t) : (A \oplus B)}$$

&L1
$$\frac{\Delta \vdash u : (A \,\&\, B) \qquad \Sigma, x : A \vdash t : C}{\Sigma, \Delta \vdash \ \texttt{let}\ \langle x, \_ \rangle\ \texttt{be}\ u\ \texttt{in}\ t : C}$$

&L2
$$\frac{\Delta \vdash u : (A \,\&\, B) \qquad \Sigma, y : B \vdash t : C}{\Sigma, \Delta \vdash \ \texttt{let}\ \langle \_, y \rangle\ \texttt{be}\ u\ \texttt{in}\ t : C}$$

&R
$$\frac{\Sigma \vdash t : A \qquad \Sigma \vdash u : B}{\Sigma \vdash \langle t, u \rangle : (A \,\&\, B)}$$

Appendix E

NAT Proof Rules

The informal reading of NAT sequents is the same as the reading of SEQ sequents. That is, an NAT sequent is composed of a type context, a $\vdash$, a linear term, and a linear type. The informal meaning of a sequent is that if one assumes the types of variables given by the type context, then the linear term has the given linear type.

$$\textbf{!WL}\quad \frac{\Sigma \vdash t : A}{\Sigma, z :!B \vdash \text{ let } z \text{ be } \_ \text{ in } t : A}$$

$$\textbf{!DL}\quad \frac{\Sigma, x : A \vdash t : B}{\Sigma, z :!A \vdash \text{ let } z \text{ be } !x \text{ in } t : B}$$

$$\textbf{?DR}\quad \frac{\Sigma \vdash t : A}{\Sigma \vdash ?t :?A}$$

$$\textbf{!CL}\quad \frac{\Sigma, x :!A, y :!A \vdash t : B}{\Sigma, z :!A \vdash \text{ let } z \text{ be } x@y \text{ in } t : B}$$

$$\textbf{?SL}\quad \frac{!\Sigma, x : A \vdash t :?B}{!\Sigma, z :?A \vdash \text{ let } z \text{ be } ?x \text{ in } t :?B}$$

$$\textbf{!SR}\quad \frac{!\Sigma \vdash t : A}{!\Sigma \vdash !t :!A}$$

$$1\textbf{L}\quad \frac{\Sigma \vdash t : A}{\Sigma, z : 1 \vdash \text{ let } z \text{ be } 1 \text{ in } t : A}$$

$$1\textbf{R}\quad \frac{}{\vdash 1 : 1}$$

$$0\textbf{L}\quad \frac{}{\Sigma, z : 0 \vdash \text{ let } z \text{ be } 0 \text{ in } t : A}$$

$$\top\textbf{R}\quad \frac{}{\Sigma \vdash \top : \top}$$

I $$\frac{}{x : A \vdash x : A}$$

Cut $$\frac{\Sigma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Sigma, \Gamma \vdash u[t/x] : B}$$

E Left $$\frac{\Sigma, x : A, y : B, \Delta \vdash u : A}{\Sigma, y : B, x : A, \Delta \vdash u : A}$$

$\otimes$**L** $$\frac{\Sigma, x : A, y : B \vdash t : C}{\Sigma, z : (A \otimes B) \vdash \text{ let } z \text{ be } (x \times y) \text{ in } t : C}$$

$\otimes$**R** $$\frac{\Sigma \vdash t : A \qquad \Gamma \vdash u : B}{\Sigma, \Gamma \vdash (t \times u) : (A \otimes B)}$$

$\multimap$**L** $$\frac{\Sigma \vdash t : A \qquad \Gamma, x : B \vdash u : C}{\Sigma, \Gamma, f : (A \multimap B) \vdash u[(ft)/x] : C}$$

$\multimap$**R** $$\frac{\Sigma, x : A \vdash t : B}{\Sigma \vdash \lambda x.t : (A \multimap B)}$$

$\oplus$**L** $$\frac{\Sigma, x : A \vdash u : C \qquad \Sigma, y : B \vdash v : C}{\Sigma, z : (A \oplus B) \vdash \text{ case } z \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v : C}$$

&R $$\frac{\Sigma \vdash t : A \qquad \Sigma \vdash u : B}{\Sigma \vdash \langle t, u \rangle : (A \,\&\, B)}$$

&L1 $$\frac{\Sigma, x : A \vdash t : C}{\Sigma, z : (A \,\&\, B) \vdash \text{ let } z \text{ be } \langle x, \_ \rangle \text{ in } t : C}$$

$\oplus$**R1** $$\frac{\Sigma \vdash t : A}{\Sigma \vdash inl(t) : (A \oplus B)}$$

&L2 $$\frac{\Sigma, y : B \vdash t : C}{\Sigma, z : (A \,\&\, B) \vdash \text{ let } z \text{ be } \langle \_, x \rangle \text{ in } t : C}$$

$\oplus$**R2** $$\frac{\Sigma \vdash u : B}{\Sigma \vdash inr(u) : (A \oplus B)}$$

Appendix D

SEQ Proof Rules

An SEQ sequent is composed of a type context, a $\vdash$, a linear term, and a linear type. The informal meaning of a sequent is that if one assumes the types of variables given by the type context, then the linear term has the given linear type.

$$\frac{t \twoheadrightarrow \lambda x.v \qquad u \twoheadrightarrow c \qquad v[c/x] \twoheadrightarrow d}{(tu) \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow (c \times d) \qquad u[c/x, d/y] \twoheadrightarrow e}{\text{let } x \times y \text{ be } t \text{ in } u \twoheadrightarrow e} \qquad \frac{t \twoheadrightarrow c \qquad u \twoheadrightarrow d}{(t \times u) \twoheadrightarrow (c \times d)}$$

$$\frac{t \twoheadrightarrow inl(c) \qquad u[c/x] \twoheadrightarrow d}{\text{case } t \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow inr(c) \qquad v[c/y] \twoheadrightarrow d}{\text{case } t \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow \langle v, w\rangle \qquad v \twoheadrightarrow c \qquad u[c/x] \twoheadrightarrow d}{\text{let } \langle x, \_\rangle \text{ be } t \text{ in } u \twoheadrightarrow d} \qquad \frac{t \twoheadrightarrow c}{inl(t) \twoheadrightarrow inl(c)}$$

$$\frac{t \twoheadrightarrow \langle v, w\rangle \qquad w \twoheadrightarrow c \qquad u[c/y] \twoheadrightarrow d}{\text{let } \langle \_, y\rangle \text{ be } t \text{ in } u \twoheadrightarrow d} \qquad \frac{t \twoheadrightarrow c}{inr(t) \twoheadrightarrow inr(c)}$$

$$\frac{}{t \twoheadrightarrow t} \qquad \frac{t \twoheadrightarrow u \qquad u \twoheadrightarrow v}{t \twoheadrightarrow v}$$

$$\frac{u \twoheadrightarrow c}{\text{discard } t \text{ in } u \twoheadrightarrow c} \qquad \frac{t \twoheadrightarrow c \quad u[c/x] \twoheadrightarrow d}{let_{cut}\, x \text{ be } t \text{ in } u \twoheadrightarrow d}\ let_{cut}$$

$$\frac{t \twoheadrightarrow !v \qquad v \twoheadrightarrow c \qquad u[c/x] \twoheadrightarrow d}{\text{read } !x \text{ as } t \text{ in } u \twoheadrightarrow d}$$

$$\frac{u[t/x, t/y] \twoheadrightarrow c}{\text{copy } x@y \text{ as } t \text{ in } u \twoheadrightarrow c}$$

$$\frac{t \twoheadrightarrow 1 \qquad u \twoheadrightarrow c}{\text{let } 1 \text{ be } t \text{ in } u \twoheadrightarrow c} \qquad \frac{}{1 \twoheadrightarrow 1}$$

Appendix C

Linear Calculus

The notation $t \twoheadrightarrow u$ is meant to be read "t evaluates in any number of steps to u". As usual, the following rules are universally quantified over terms t, u, v, and variables x, y, z.

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Sigma \vdash A \qquad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$	**Cut**
E Left	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C}$		
$\otimes$**L**	$\dfrac{\Sigma, A, B \vdash C}{\Sigma, (A \otimes B) \vdash C}$	$\dfrac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}$	$\otimes$**R**
$\multimap$**L**	$\dfrac{\Sigma \vdash A \qquad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \multimap B) \vdash C}$	$\dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}$	$\multimap$**R**
$\oplus$**L**	$\dfrac{\Sigma, A \vdash C \qquad \Sigma, B \vdash C}{\Sigma, (A \oplus B) \vdash C}$	$\dfrac{\Sigma \vdash A \qquad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}$	**&R**
&L1	$\dfrac{\Sigma, A \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R1**
&L2	$\dfrac{\Sigma, B \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R2**
!WL	$\dfrac{\Sigma \vdash A}{\Sigma, !B \vdash A}$		
!DL	$\dfrac{\Sigma, A \vdash B}{\Sigma, !A \vdash B}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash ?A}$	**?DR**
!CL	$\dfrac{\Sigma, !A, !A \vdash B}{\Sigma, !A \vdash B}$		
?SL	$\dfrac{!\Sigma, A \vdash ?B}{!\Sigma, ?A \vdash ?B}$	$\dfrac{!\Sigma \vdash A}{!\Sigma \vdash !A}$	**!SR**
1**L**	$\dfrac{\Sigma \vdash A}{\Sigma, 1 \vdash A}$	$\dfrac{}{\vdash 1}$	1**R**
0**L**	$\dfrac{}{\Sigma, 0 \vdash A}$	$\dfrac{}{\Sigma \vdash \top}$	$\top$**R**

Appendix B

Prop. Intuitionistic Linear Logic

An intuitionistic linear logic sequent is composed of a multiset of linear logic formulas, and a single formula separated by a $\vdash$. This intuitionistic proof system is restricted so that there is no way to derive a sequent where the multiset on the right of the $\vdash$ contains more than one element. Below we give the inference rules for the intuitionistic linear sequent calculus. We assume a set of propositions p_i given.

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2}$	**Cut**
E Left	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Sigma}{\Gamma_1, B, A, \Gamma_2 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma_1, A, B, \Sigma_2}{\Gamma \vdash \Sigma_1, B, A, \Sigma_2}$	**E Right**
$\otimes$ **Left**	$\dfrac{\Gamma, A, B \vdash \Sigma}{\Gamma, (A \otimes B) \vdash \Sigma}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2 \vdash B, \Sigma_2}{\Gamma_1, \Gamma_2 \vdash (A \otimes B), \Sigma_1, \Sigma_2}$	$\otimes$ **Right**
$\parr$ **Left**	$\dfrac{\Gamma_1, A \vdash \Sigma_1 \quad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \parr B) \vdash \Sigma_1, \Sigma_2}$	$\dfrac{\Gamma \vdash A, B, \Sigma}{\Gamma \vdash (A \parr B), \Sigma}$	$\parr$ **Right**
$\oplus$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	$\&$ **Right**
$\&$ **Left1**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right1**
$\&$ **Left2**	$\dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right2**
! W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	**! C**
! D	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma}$	**! S**
? W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{\Gamma \vdash ?A, ?A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	**? C**
? D	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{!\Gamma, A \vdash ?\Sigma}{!\Gamma, ?A \vdash ?\Sigma}$	**? S**
$^\perp$ **Left**	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma, A^\perp \vdash \Sigma}$	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma \vdash A^\perp, \Sigma}$	$^\perp$ **Right**
0 **Left**	$\Gamma, 0 \vdash \Sigma$	$\Gamma \vdash \top, \Sigma$	$\top$ **Right**
1 **Left**	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, 1 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash \perp, \Sigma}$	$\perp$ **Right**
$\perp$ **Left**	$\perp \vdash$	$\vdash 1$	1 **Right**

Appendix A

Linear Logic Sequent Calculus Rules

A linear logic sequent is composed of two sequences of linear logic formulas separated by a $\vdash$. In most proofs, sequents are assumed to be constructed from *multisets* of linear logic formulas, effectively ignoring applications of the exchange rule.

The following notational conventions are used

p_i	Positive propositional literal
$p_i^\perp$	Negative propositional literal
A, B, C	Arbitrary formulas
$\Sigma, \Gamma, \Delta, \Theta$	Arbitrary sequences of formulas
$\otimes$	Tensor, the multiplicative conjunction
1	One, the unit of tensor
$\parr$	Par, the multiplicative disjunction
$\perp$	Bottom, the unit of par
$\&$	With, the additive conjunction
$\top$	Top, the unit of with
$\oplus$	Plus, the additive disjunction
0	Zero, the unit of plus

systems is a promising approach for the study of proof theory and for the study of more mainstream computer science. Indeed, linear logic has been introduced only recently and the body of work connecting theoretical computer science and linear logic is already quite large. This thesis contributes to this exciting application of logic in computer science, showing that linear logic is not about "Truth"; it is about computation.

linear logic and counter machines. This connection shows how to read proofs as descriptions of (successful) computations. This connection also provides a proof that propositional linear logic is undecidable. Other computationally interesting classes are also naturally represented in fragments of linear logic: semi-Thue systems may be embedded in noncommutative linear logic, classical quantified boolean formulas may be encoded in MALL, showing that MALL is PSPACE-complete, and 3-Partition can be encoded in constant-only multiplicative linear logic, showing that constant-only MLL is NP-complete.

Finally, exploring an additional computational interpretation of linear logic based on the Curry-Howard isomorphism, this thesis continues a line of research begun early in the history of linear logic. This thesis proves the subject-reduction (type-soundness) and the most general type theorems for ML^{--}, a language related to those studied by others [53, 1, 91, 93, 21, 65, 16]. This thesis pushes proof theory a little further further into compilation, providing a sound theoretical basis for some compiler optimizations which are currently performed in an ad hoc manner.

For future directions, the complexity of a few interesting decision problems for linear logic is still unknown. Without additives, multiplicative exponential linear logic (MELL) can encode Petri-net reachability problems, which are known to be EXPSPACE-hard, but decidable [64, 71, 70, 69, 52]. Without positive polarity occurrences of !, or negative polarity occurrences of ?, MELL is decidable, by reduction to a Petri-net reachability problem. However, in general the decidability of the decision problem for MELL is open. Another decision problem of interest involves the quantifiers. Work is currently progressing on computational interpretations of first order MALL and full first order linear logic [61, 83]. Although some preliminary work has been done on second order MALL [5], the decidability of pure second-order MALL is open.

Future work in the functional language direction involves the translation of typed lambda calculus into the linear lambda calculus. It also involves further study of the computational efficiency of implementations of the linear lambda calculus. Work on both end of this problem is required in order to make practical use of the results of this thesis.

Establishing close connections between intuitive computational models and logical

Chapter 8

Conclusion

This thesis investigates computational aspects of linear logic. The main results of this work support the proposition: "Linear logic is a *computational* logic behind logics."

This thesis augments the proof theoretic framework of linear logic by providing theorems such as permutability, impermutability, and cut-standardization with non-logical theories. On this expanded proof theoretic base, many complexity results are proved using the Girard correspondence between proofs and computations. Among these results are the undecidability of propositional linear logic, the PSPACE-completeness of MALL, and the NP-completeness of the constant-only multiplicative fragment of linear logic. Another application of proof theory to computation is explored for a functional language ML^{--} and its (compiled) implementation. The proposed linear type system for ML^{--} yields compile-time type information about resource manipulation, which can be used to control aspects of program execution such as storage allocation, garbage collection, and array update in place. Most general type and subject reduction theorems are proved, and a compiled implementation based on the Three Instruction Machine is described.

In more detail, we demonstrate the power of the Girard correspondence between linear logic proofs and computation, establishing the complexity of the decision problem in several fragments of linear logic. There had been little previous work on decision problems in linear logic, and full propositional linear logic had been suspected to be decidable. Using the Girard correspondence, this thesis connects full propositional

argument stack, the label contains a pointer to the current continuation frame, and the space the current frame takes up cannot be reclaimed. However, nonlinear values in frames are overwritten once computed (in the lazy style of the Tim), and whole frames can be shared, instead of being copied. The penalty for this is that some traditional garbage collection mechanism must be used on frames.

All other objects are handled with explicit sharing instructions. The objects handled in this way include arrays, structures, and cons cells. These are separated into two classes: linear and nonlinear. Linear objects are not copyable, and are never referenced by two pointers at the same time. In our implementation all arrays (even linear ones) have elements which are reusable (of ! type), although the arrays can be of arbitrary dimension. A nonlinear object is essentially handled in the a traditional way, with sharing of pointers to the same object. That is, nonlinear objects may be referenced by any number of pointers simultaneously.

7.7 Summary of Chapter

We have presented a linear calculus and three type inference systems: SEQ, NAT, and NAT2. We have shown that SEQ and NAT equivalent, and that NAT2 is closely related. We have demonstrated the existence of most general types and the subject reduction theorem. The linear calculus and very closely related type systems have appeared elsewhere, perhaps most well known in [53, 1].

Also, we have implemented a two-space abstract machine based on the three instruction machine which may be used to exploit the information available in linear types to generate more efficient code. For example, one may perform update in place on arrays in linear space. Although the study of opportunities for update in place in functional languages has a long history, the linear calculus and its type systems present a logical foundation for this kind of "resource-conscious" compiler optimization.

garbage collection by copying, a rather costly implementation technique [40]. Chirimar, Gunter, and Riecke have described an implementation which also focuses on the issue of garbage collection [21]. In their implementation, objects may be shared, so dynamic garbage collection is potentially required on all objects. However, the linear types of terms may be used to identify potential times at which objects may become garbage. Their implementation does not include the additives, but is extended with a recursion operator and polymorphism. Abramsky has described the implementation of a linear SECD machine further studied by Mackie [1, 65], and went on to generalize the linear calculus to one based on classical linear logic and described an implementation based on the chemical abstract machine [17]. Wadler [93] has also described several implementation issues regarding the linear calculus. He points out the importance of `!(!A)` being isomorphic to `!A` (which is true in our operational model), and suggests several extensions, including, for example, arrays, let! with read-only access, the removal of syntax for weakening and contraction, etc. Wadler also discusses the separation of types into linear and nonlinear, giving the types different syntax, very similar to our two memory spaces. We have considered only the extension of the linear calculus to include recursion and arrays, essentially as mentioned by Wadler [92].

Our LTim implementation does not count the references of integers, continuation frames, nor code. In linear logic terms, it is assumed that code, continuations, and and base integer values are of ! type. That is, they are reusable. However, arrays, structures, and cons cells are not treated in this manner. Since integer values are assumed one word long, it is more efficient to copy them, rather than sharing. Code is always assumed to be nonlinear, and is shared. Code is traditionally assumed to be static and reusable, and it is difficult to imagine an implementation taking much advantage of code-space freed up when some code is executed for the last time. Continuations are assumed to be nonlinear, and are shared. This (mis)management of the storage for continuation frames could be a serious deficiency of this implementation. Continuation frames contain a sequence of pairs of pointers into code space and data. A continuation frame is created upon entry into every combinator, and must be preserved whenever a combinator suspends computation while control is transferred to some other combinator. That is, whenever a combinator pushes a label on the

with four special combinators: DELAY, FORCE, COPY, and DISCARD, and modifies some of the internal data structures of Tim to also support eager evaluation and explicit storage management. The LTim implementation was pursued for two reasons. First, it provides further evidence that the linear calculus may be executed efficiently. Second, it embodies a natural dual space memory model well suited to the execution of linear calculus terms.

The key point of departure of our implementation from the previous implementations of the linear calculus is the memory model. The LTim implements two spaces, one linear, and one nonlinear. The idea is that objects in the linear space are purely linear, and thus have a reference count of exactly one at all times. Objects in the nonlinear space represent "stored" or reusable entities. Little or no static information is available about reference counts of objects in this space. The execution model we have in mind is that a ! or `store` instruction (corresponding to the !SR rule of linear logic) ensures that objects reside in nonlinear space. Once an object is stored, it may be discarded or copied, a `discard` operation removes a pointer to a stored object, and a `copy` operation simply copies a pointer to an object in nonlinear space, thus implementing sharing, or call-by-need. However, in this nonlinear space, objects can be referenced any number of times (including 0), requiring some form of dynamic garbage collection. In the linear space, objects are never shared; there is always exactly one reference to all objects. Thus garbage collection is not needed, since objects in that space become garbage the first time they are used, and linear objects may always be updated in place. In other words, in our execution model dynamic garbage collection is never applied to objects in linear space, but may occasionally be applied to objects in nonlinear space. Update in place is always applicable to linear objects, but is never applied to nonlinear objects.

Other implementations of the linear calculus have effectively assumed a single memory space. A potential disadvantage of the single memory space is that it obfuscates the distinction between shared and unshared objects. Lafont built an implementation of the linear calculus with the fantastic property that dynamic garbage collection is never used: all terms effectively have exactly one reference to them, and thus become garbage the first (only) time they are referenced. However, Lafont avoids

The above judgement is provable in SEQ, NAT, and NAT2, but after one step of reduction, the judgement becomes $x :!A \vdash 1$ which is not provable in any type system discussed in this chapter. Chirimar, Gunter, and Riecke have also noticed this failure of subject reduction for open terms [21].

On the other hand, we do have this more general form of subject reduction of the reflexive transitive closure of $\rightarrow_0$. That is, we may reduce using $\rightarrow_0$ anywhere in a term and still preserve the types.

With a slight modification of the systems we are working with, an intermediate form of these subject reduction theorems is possible. If Γ and Σ are multisets, then we write $\Sigma \subseteq \Gamma$ to mean that Σ may by obtained from Γ by removing elements or adding duplicates. The following theorem holds in a version of these type systems where the restriction that every variable occur exactly once in binding and once in use is relaxed to the restriction that every variable occur as many times in binding as it occurs in use, and all occurrences of a variable have the same type.

Theorem 7.5.92 (Generalized Subject Reduction) *If there is a proof of* $!\Gamma \vdash t :A$ *in* NAT *or* SEQ, *and* $t \twoheadrightarrow s$, *then there is a proof of* $!\Sigma \vdash s :A$ *in* NAT *and* SEQ, *where* $!\Sigma \subseteq !\Gamma$.

Also, a weaker form of subject reduction theorem holds for NAT2. The restrictions on reduction order are sufficient to guarantee that whenever $(\lambda x.t)u$ is reduced, under the conditions given in the theorem below, $t[u/x]$ has the same type.

Theorem 7.5.93 *If there is a proof of* $\vdash t :A$ *in* NAT2, *and* $t \twoheadrightarrow s$, *then there is a proof of* $\vdash s' :A$ *in* NAT2, *for some term* s' *related to* s.

7.6 Implementation of LC

We now give an overview of a compiled implementation of the linear calculus based on insights provided by these studies of type systems. This implementation is based on a modified version (LTim) of the Three Instruction Machine (Tim).

The Tim is an extremely simple abstract machine designed to facilitate lazy reduction of super combinator expressions [27, 96, 45, 46]. The LTim extendeds the Tim

for SEQ as a corollary. The main reason that the proof is simpler for NAT can be seen by comparing the $\multimap$**L** rule of SEQ with the **A** (application) rule of NAT. When an application $(\lambda x.t)u$ is β-reduced, the structure of the NAT rules guarantee that this was typed by the **A** rule and t was typed subject to some hypothesis about the type of variable x. The same type may be given to $t[u/x]$ using a substitution instance of this proof. In SEQ, we would need a series of detailed lemmas giving us some information about the possible structure of the typing proof for $(\lambda x.t)u$. In particular, since the sequent rule $\multimap$**L** allows arbitrary substitution, the structure of the typing proof is not determined by the form of an application.

The following lemma is the key step in the inductive proof of Theorem **??**. It covers the case of the very general **!SR** rule, essentially stating that for one-step linear reduction, terms of ! type do not interact with any other terms. Note that the reduction relation $\rightarrow_0^{\mathcal{C}}$ does not include any of the ! reductions.

Lemma 7.5.89 *If* $t = r[s_1/x_1, \cdots, s_n/x_n]$, *and* $t \rightarrow_0^{\mathcal{C}} u$ *and for* $1 \leq i \leq n : \Delta_i \vdash s_i : !B_i$
then $(u = r'[s_1/x_1, \cdots, s_n/x_n]$ *and* $r \rightarrow_0^{\mathcal{C}} r')$ *or*
$(\exists j : u = r[s_1/x_1, \cdots, s'_j/x_j, \cdots, s_n/x_n]$ *and* $s_j \rightarrow_0^{\mathcal{C}} s'_j)$.

Theorem 7.5.90 (NAT Subject Reduction) *If there is a proof of* $\vdash t : A$ *in* NAT, *and* $t \twoheadrightarrow s$, *then there is a proof of* $\vdash s : A$ *in* NAT.

Proof. Induction on the derivation $t \twoheadrightarrow s$. ∎

Corollary 7.5.91 *If there is a proof of* $\vdash t : A$ *in* SEQ, *and* $t \twoheadrightarrow s$, *then there is a proof of* $\vdash s : A$ *in* SEQ.

The stronger property that if there is any typing proof of $\Sigma \vdash t : A$ for term t with free variables, and $t \twoheadrightarrow s$, then there is a typing proof of $\Sigma \vdash s : A$ does not hold. As we have seen, control over evaluation order is of critical importance in maintaining linear type soundness, as the following example demonstrates.

$$x : !A \vdash \texttt{discard}\ x\ \texttt{in}\ 1$$

```
nat(Context, discard(U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(Sigma, T, B),
        append(Delta, Sigma, Context).

nat(Context, copy(var(X), var(Y), U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(S1, T, B),
        remove(type(var(X), ofcourse(A)), S1, S2),
        remove(type(var(Y), ofcourse(A)), S2, S3),
        append(Delta, S3, Context).

nat(Context, read(store(var(X)), U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma2),
        append(Delta, Sigma2, Context).

nat(Sigma, store(T), ofcourse(A)) :-
        nat(Sigma, T, A),
        bangify(Sigma).

nat(Context, let(U, 1, T), A) :-
        nat(Delta, U, 1),
        nat(Sigma, T, A),
        append(Delta, Sigma, Context).

bangify([]).
bangify([ type(var(X), ofcourse(A)) | Rest]) :-
        bangify(Rest).
```

Figure 7.3: Prolog Implementation of the ! fragment of NAT2

```
nat([ type(var(X), A) ], var(X), A).
nat(Context, apply(U, T), B) :-
        nat(Delta, U, la(A, B)),
        nat(Sigma, T, A),
        append(Delta, Sigma, Context).

nat(Sigma, lambda(var(X), T), la(A, B)) :-
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma),
        notmember(type(var(X),Any), Sigma).

nat(Context, let(U, times(var(X), var(Y)), T), C) :-
        nat(Delta, U, tensor(A, B)),
        nat(Sigma1, T, C),
        remove(type(var(X), A), Sigma1, Sigma2),
        remove(type(var(Y), B), Sigma2, Sigma3),
        append(Delta, Sigma3, Context).

nat(Context, times(U, T), tensor(A, B)) :-
        nat(Sigma, U, A),
        nat(Gamma, T, B),
        append(Sigma, Gamma, Context).

nat(Context, let(U, var(X), T), B) :-
        nat(Delta, U, A),
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma2),
        append(Delta, Sigma2, Context).
```

Figure 7.2: Prolog Implementation of the $\otimes, \multimap$ fragment of NAT2

It is well known that one may view a (traditional, well-typed) functional program as a proof notation [36] for intuitionistic logic. In this light, the type of a program is the conclusion of the proof which it represents. In our present context, we view a linear program as proof notation for intuitionistic linear logic. Thus one could view this procedure as an intuitionistic linear logic proof checker, which also computes the most general conclusion which may be drawn from the given proof.

Noting that the type rules for NAT2 may be written as horn clauses, we have implemented NAT2 in Prolog. The side conditions on the rules may easily be encoded in Prolog. For example the **!SR** rule requires a procedure which ensures that element of the type context has ! type (`bangify`). The type system implemented in this manner has the wonderful property that given a linear term, no search is required to discover it's proof, if one exists, and the entire type checking may be performed in low order polynomial time.

In the Prolog implementation of NAT2 most of which is shown in Figures **??** and **??**, there is exactly one clause for each inference rule of NAT2. There are also predicates defining `append`, `remove`, `notmember`, and `bangify`, although only `bangify` is given in the figure. We give the code implementing NAT2 for the $\otimes$, $\multimap$, ! fragment of the logic. For this implementation to be sound, the Prolog implementation would have to provide a sound unification procedure (with occurs-check).

7.5 Type Soundness

In this section we prove a technical property commonly called "subject reduction" for both NAT and SEQ. This property is that if linear term t has type A, and t reduces to t', then t' also has type A. The term t' may also have other types; rewriting a term may allow us to deduce more typing properties. However, if the typing rules allows us to derive some property of a term, this property remains as we reduce, or evaluate, the term. Without the subject reduction property, it might be possible for a typed term to become untypable during execution. In this case, we would consider the type system "unsound" as a method for determining the absence of type errors.

We prove subject reduction by considering NAT first and then deriving the result

The common feature of these judgements is that they both allow us to apply ! to the expression, in the first case because all of the types of free variables begin with ! and in the second case because the type of the expression begins with !. In NAT, the two typings of the expression with `store`are derived using two different substitutions in the **!SR**, rule. In SEQ, the two typings are derived using **Cut** to substitute into a `store` expression in two different ways.

Accounting for the possibility of several different substitutions in the **!SR** rules (some of which are provably unnecessary), all of the NAT rules are straightforward syntax-directed rules that may be translated into Prolog Horn clauses without complication. This gives us an algorithm that finds a finite set of most general types for each linearly typable term or (since the search is bounded) terminates with failure on untypable terms.

Theorem 7.4.87 (MGT) *Every* NAT *typable term has a finite set of most general types. There is a unification-based algorithm that, given any term, either computes a set of most general types or halts with* failure *if the term is not typable.*

7.4.2 Most General Types in NAT2

In the simplified NAT2 system, **!SR** is replaced by a simple syntax-directed rule with no possibility of substitution. Consequently, the most general type of any typable linear term may be computed by a unification-based algorithm or simple Prolog program. The following most general typing theorem is due independently to Mackie [65].

Theorem 7.4.88 (MGT) *There is a unification-based algorithm that computes a most general linear type for any* NAT2 *typable term t and terminates in* failure *on any untypable term.*

Proof Sketch. If t is NAT2 typable, then it has a type judgement in NAT2 ending in a sequent $\vdash t : A$ for some type A. This type judgement is unique up to the instantiation of types at the axioms. The most general type will be found by unification, where fresh type variables are initially used at each application of identity. ∎

specifically, a linear formula, A, is *more general* than another, B, if there exists a substitution σ mapping linear propositions to linear formulas such that $(A)\sigma = B$. A set, S, of formulas is more general than another, T, if every element of T is a substitution instance of some element of S. Given a term t, the typing algorithm either returns a finite set of formulas more general than all types of t, or terminates with *failure* if t has no linear type. The number of formulas in the set of most general types is bounded by an exponential function of the number of uses of `store` in the term. Without `store`, every typable term has a single most general type.

Up to **!SR**, the rules of NAT that are used in a typing derivation, and the order of application, are totally determined by the syntactic structure of the linear term. For example, if the term is a variable then the only possible proof in NAT is one use of identity. If the term is $\lambda x.t$, then the only possible rule is $\multimap$ **R**. The only freedom, except for **!SR**, is in the choice of linear types for variables and the division of a type multiset among hypotheses (in the rules with multiple hypotheses). However, type judgements contain exactly the set of free variables of a term in the type context. This property determines the division of a multiset among hypotheses of the rule. Thus, for NAT without **!SR**, we may compute the most general typing by a simple Prolog program, obtained by translating the typing rules into Horn clauses in a straightforward manner.

An example that shows the complications associated with terms of the form `store` t is

$$\lambda a.\lambda b.\ \texttt{store}\ ((\ \texttt{read}\ \ \texttt{store}\ c\ \texttt{as}\ a\ \texttt{in}\ c)\, b)$$

which has the two incomparable NAT and SEQ types

$$!(!B \multimap C) \multimap !B \multimap !C$$
$$!(B \multimap !C) \multimap B \multimap !!C$$

The basic idea is very similar to the example in Section 7.2 that involves implicit `store`. If $a : !A$, then the expression (`read store` c `as` a `in` c) has type A. This gives us the two typing judgements

$$a : !(!B \multimap C),\ b : !B \vdash (\texttt{read}\ \ \texttt{store}\ c\ \texttt{as}\ a\ \texttt{in}\ c)\, b\ :\ C$$
$$a : !(B \multimap !C),\ b : B \vdash (\texttt{read}\ \ \texttt{store}\ c\ \texttt{as}\ a\ \texttt{in}\ c)\, b\ :\ !C$$

A term t' is *related* to a term t if t can be obtained from t' by replacing occurrences $let_{cut}x$ `be` t `in` v with $v[t/x]$.

Theorem 7.3.84 (SEQ **equiv** NAT2) *A type sequent* $\Gamma \vdash t : A$ *is provable in* SEQ *if and only* $\Gamma \vdash t' : A$ *is provable in* NAT2 *for some* t' *related to* t.

This theorem may be proven in the same manner as the above, although in some cases the extra syntax of let_{cut} is used in the NAT2 term t'. The reason for let_{cut} is that **Subst** is not a derived rule of NAT2.

We now turn our attention to the main technical differences between the three sets of typing rules. If we imagine searching for a cut-free proof of a typing derivation, beginning with a prospective conclusion and progressing toward appropriate instances of the axioms, our search will be driven by the form of the term in NAT2 and the form of the type in SEQ. To state this precisely, we begin by reviewing the routine definitions of type subformula and term subformula.

A type A is a *type subformula* of a type B if A syntacticly occurs in B. Similarly, a term t is a *term subformula* of a term s if t syntacticly occurs in s.

Lemma 7.3.85 (NAT2 **Term Subformula Property**) *In any proof of* $\vdash u : B$ *in* NAT2, *every term* t *that appears anywhere in the proof is a subformula of* u.

Note that the system NAT fails to have this property because of the form of the **!SR** rule.

Lemma 7.3.86 (SEQ **Type Subformula Property**) *For any cut-free proof of* $\vdash t : B$ *in* SEQ, *any type* A *that appears anywhere in the proof is a subformula of* B.

Note that the cut rule violates the subformula property, and so Lemma **??** does not hold of SEQ proofs with cut.

7.4 Most General Linear Type

7.4.1 Most General Types in NAT and SEQ

In this section, we show that every linearly typable term has a finite set of most general types. This set may be used to decide the set of types of the term. More

7.3.5 Equivalence of SEQ and NAT

In the last two subsections, we presented two systems intended for the automatic inference of linear type information for linear terms. In the following Theorem **??**, we prove that the two systems are equivalent, that is, that any linear type judgement achievable in one system is also achievable in the other, up to reductions of let_{cut}, which marks occurrences of the unrestricted cut rule in NAT.

We should emphasize that although NAT and SEQ are equivalent (up to let_{cut} reduction) in the sense that any typing judgement provable in SEQ is provable in NAT and vice-versa, they are not equivalent with respect to operations on proofs. In particular, cut-elimination is of hyperexponential complexity in SEQ, and subst-elimination is of polynomial complexity in NAT. This is explained by the fact that all the left rules of NAT have an "implicit cut" incorporated into them. As a result, cut-elimination in SEQ does not correspond to subst-elimination in NAT, nor is there a direct connection between subst-elimination in NAT and reduction in the linear calculus. Instead, in NAT one focuses on the elimination of introduction/elimination pairs as the model of computation.

Theorem 7.3.83 (SEQ **equiv** NAT) *A type sequent* $\Gamma \vdash t : A$ *is provable in* SEQ *if and only* $\Gamma \vdash t : A$ *is provable in* NAT.

Proof. One can show that each rule in SEQ is derivable in NAT, and vice versa, using local transformations. All the right rules, identity, are the same in both systems, and **Cut** and **Subst** take the same form. For most left rules, one may simulate the NAT version of the rule in SEQ with one application of the rule of similar name and one application of **Cut**. One may simulate the SEQ version of most left rules in NAT by using the rule of the same name and identity. The multi-hypothesis **!SR** rule of NAT is derivable in SEQ with the use of **!SR** and multiple instances of **Cut**.

One may transform instances of the **Cut** rule in SEQ as applications of **Subst**, which is a derivable rule in NAT. Upon removal of **Subst**, one may see that **Cut** in SEQ corresponds to introduction-elimination pairs of rules in NAT. ■

There is a somewhat looser correspondence between NAT2 and SEQ, than that just claimed between NAT and SEQ.

By induction, we can produce proofs of $\Sigma \vdash u : B$ and $\Delta, x : B \vdash v : A$ of degree less than d. By a single application of Lemma **??** to the resulting proof constructed from the modified hypotheses, we obtain a proof of $\Gamma \vdash t : A$ of degree less than d. ■

Theorem 7.3.82 (NAT **Subst-Elimination**) *If a sequent is provable in* NAT*, then it is provable in* NAT *without using the* **Subst** *rule.*

Proof. By induction on the degree of the assumed proof. We may apply Lemma **??** at each inductive step, and at the base case the degree of the proof is zero, so therefore by definition of proof degree there are no substs, and we have our desired subst-free proof. ■

0L subst formula on the left

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma \vdash t : C & \dfrac{\Delta \vdash v : 0}{x : C, \Sigma, \Delta \vdash \text{ let } v \text{ be } 0 \text{ in } u : A}\,\mathbf{0L}\end{array}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } v \text{ be } 0 \text{ in } u)[t/x] : A}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots \\ \Delta \vdash v : 0\end{array}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } v \text{ be } 0 \text{ in } u)[t/x] : A}\,\mathbf{0L}$$

This exhausts all the cases. ■

Thus, we have a procedure which given a proof which ends in **Subst** of degree d, and which has no applications of **Subst** in the proof of either hypothesis of degree greater than or equal to d, produces a proof of degree less than d.

Lemma 7.3.81 (Lower-Degree-Substs) *If a sequent is provable in* NAT *with a proof of degree $d > 0$, then it is provable in* NAT *with a proof of degree less than d.*

Proof. By induction on the height of the derivation tree of the conclusion. We show that given any proof of degree d of $\Gamma \vdash t : A$ in NAT, we may find a (possibly much larger) proof of $\Gamma \vdash t : A$ in NAT of degree less than d.

We examine the proof of $\Gamma \vdash t : A$. Since the degree of this proof is greater than zero, there must be some **Subst** in the proof. If the last rule is not **Subst**, then by induction we may form proofs of its hypotheses of degree less than d. Applying the same rule to the resulting reduced degree hypotheses produces the desired proof of degree less than d.

In the case that the last rule is **Subst**, we have the following situation for some Σ and Δ which together (in multiset union) make up Γ:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Sigma \vdash u : B & \Delta, x : B \vdash v : A\end{array}}{\Gamma \vdash v[u/x] : A}\,\mathbf{Subst} \quad \text{where } \Sigma \cup \Delta = \Gamma \text{ and } v[u/x] = t$$

1L, formula descends from the right

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Delta \vdash w : 1 \qquad x : C, \Sigma \vdash u : D}\,\mathbf{1L}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{let } w \text{ be } 1 \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{let } w \text{ be } 1 \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Sigma \vdash u : D}\,\mathbf{Subst}}{\Delta \vdash w : 1 \qquad \Gamma, \Sigma \vdash u[t/x] : D}\,\mathbf{1L}}{\Gamma, \Sigma, \Delta \vdash \text{let } w \text{ be } 1 \text{ in } (u[t/x]) : D}$$

1L, formula descends from the left

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{x : C, \Delta \vdash w : 1 \qquad \Sigma \vdash u : D}\,\mathbf{1L}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{let } w \text{ be } 1 \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{let } w \text{ be } 1 \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Delta \vdash w : 1}\,\mathbf{Subst} \qquad \vdots}{\Gamma, \Sigma \vdash w[t/x] : 1 \qquad \Sigma \vdash u : D}\,\mathbf{1L}$$
$$\Gamma, \Sigma, \Delta \vdash \text{let } w[t/x] \text{ be } 1 \text{ in } u : D$$

$\top$R

$$\dfrac{\vdots}{\dfrac{\Gamma \vdash t : C \qquad \dfrac{}{x : C, \Sigma \vdash \top : \top}\,\top\mathbf{R}}{\Gamma, \Sigma \vdash \top[t/x] : \top}\,\mathbf{Subst}}$$

$$\Downarrow$$

$$\dfrac{}{\Gamma, \Sigma \vdash \top : \top}\,\top\mathbf{R}$$

!DL, formula descends from the right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Delta \vdash w : !B \qquad x : C, \Sigma, y : B \vdash u : D}{x : C, \Sigma, \Delta \vdash \text{ read store} y \text{ as } w \text{ in } u : D}\,\textbf{!DL}}{\Gamma, \Sigma, \Delta \vdash (\text{ read store} y \text{ as } w \text{ in } u)[t/x] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\Delta \vdash w : !B \qquad \dfrac{\Gamma \vdash t : C \qquad x : C, \Sigma, y : B \vdash u : D}{\Gamma, \Sigma, y : B \vdash u[t/x] : D}\,\textbf{Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w \text{ in } (u[t/x]) : D}\,\textbf{!DL}$$

!DL, formula descends from the left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{x : C, \Delta \vdash w : !B \qquad \Sigma, y : B \vdash u : D}{x : C, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w \text{ in } u : D}\,\textbf{!DL}}{\Gamma, \Sigma, \Delta \vdash (\text{ read store} y \text{ as } w \text{ in } u)[t/x] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad x : C, \Delta \vdash w : !B}{\Gamma, \Delta \vdash w[t/x] : !B}\,\textbf{Subst} \qquad \Sigma, y : B \vdash u : D}{\Gamma, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w[t/x] \text{ in } u : D}\,\textbf{!DL}$$

!CL, formula descends from the right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Delta \vdash w : !B \qquad s : C, \Sigma, x : !B, y : !B \vdash u : D}{s : C, \Sigma, \Delta \vdash \text{ copy } x @ y \text{ as } w \text{ in } u : D}\,!\mathbf{CL}}{\Gamma, \Sigma, \Delta \vdash (\text{ copy } x @ y \text{ as } w \text{ in } u)[t/s] : D}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\Delta \vdash w : !B \qquad \dfrac{\Gamma \vdash t : C \qquad s : C, \Sigma, x : !B, y : !B \vdash u : D}{\Gamma, \Sigma, x : !B, y : !B \vdash u[t/s] : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ copy } x @ y \text{ as } w \text{ in } (u[t/s] : D}\,!\mathbf{CL}$$

!CL, formula descends from the left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{s : C, \Delta \vdash w : !B \qquad \Sigma, x : !B, y : !B \vdash u : D}{s : C, \Sigma, \Delta \vdash \text{ copy } x @ y \text{ as } w \text{ in } u : D}\,!\mathbf{CL}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } x @ y \text{ as } w \text{ in } u)[t/s] : D}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad s : C, \Delta \vdash w : !B}{\Gamma, \Delta \vdash w[t/s] : !B}\,\mathbf{Subst} \qquad \Sigma, x : !B, y : !B \vdash u : D}{\Gamma, \Sigma, \Delta \vdash \text{ copy } x @ y \text{ as } (w[t/s]) \text{ in } u : D}\,!\mathbf{CL}$$

!WL, formula descends from the right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Delta \vdash w : !B \qquad x : C, \Sigma \vdash u : D}{x : C, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}\,\textbf{!WL}}{\Gamma, \Sigma, \Delta \vdash (\text{ discard } w \text{ in } u)[t/x] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad x : C, \Sigma \vdash u : D}{\Gamma, \Sigma \vdash u[t/x] : D}\,\textbf{Subst} \qquad \Delta \vdash w : !B}{\Gamma, \Sigma, \Delta \vdash \text{ discard } w \text{ in } (u[t/x]) : D}\,\textbf{!WL}$$

!WL, formula descends from the left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{x : C, \Delta \vdash w : !B \qquad \Sigma \vdash u : D}{x : C, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}\,\textbf{!WL}}{\Gamma, \Sigma, \Delta \vdash (\text{ discard } w \text{ in } u)[t/x] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad x : C, \Delta \vdash w : !B}{\Gamma, \Delta \vdash w[t/x] : !B}\,\textbf{Subst} \qquad \Sigma \vdash u : D}{\Gamma, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}\,\textbf{!WL}$$

&L2, formula descends from the right

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Delta \vdash w : (A \oplus B) \qquad x : C, \Sigma, y : B \vdash u : D}\,\&\mathbf{L2}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Sigma, y : B \vdash u : D}\,\mathbf{Subst}}{\Delta \vdash w : (A \oplus B) \qquad \Gamma, \Sigma, y : B \vdash u[t/x] : D}\,\&\mathbf{L2}}{\Gamma, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } (u[t/x]) : D}$$

&L2, formula descends from the left

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{x : C, \Delta \vdash w : (A \oplus B) \qquad \Sigma, y : B \vdash u : D}\,\&\mathbf{L2}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Delta \vdash w : (A \oplus B)}\,\mathbf{Subst} \qquad \vdots}{\Gamma, \Delta \vdash w[t/x] : (A \oplus B) \qquad \Sigma, y : B \vdash u : D}\,\&\mathbf{L2}$$
$$\Gamma, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w[t/x] \text{ in } u : D$$

&L1, formula descends from the right

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Delta \vdash w : (A \oplus B) \qquad x : C, \Sigma, y : A \vdash u : D}\,\&\mathbf{L1}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{ let } \langle y, \_ \rangle \text{ be } w \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle y, \_ \rangle \text{ be } w \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Sigma, y : A \vdash u : D}\,\mathbf{Subst}}{\Delta \vdash w : (A \oplus B) \qquad \Gamma, \Sigma, y : A \vdash u[t/x] : D}\,\&\mathbf{L1}$$

$$\Gamma, \Sigma, \Delta \vdash \text{ let } \langle y, \_ \rangle \text{ be } w \text{ in } (u[t/x]) : D$$

&L1, formula descends from the left

$$\dfrac{\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{x : C, \Delta \vdash w : (A \oplus B) \qquad \Sigma, y : A \vdash u : D}\,\&\mathbf{L1}}{\Gamma \vdash t : C \qquad x : C, \Sigma, \Delta \vdash \text{ let } \langle y, \_ \rangle \text{ be } w \text{ in } u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle y, \_ \rangle \text{ be } w \text{ in } u)[t/x] : D}$$

$$\Downarrow$$

$$\dfrac{\vdots \qquad \vdots \qquad \vdots}{\dfrac{\Gamma \vdash t : C \qquad x : C, \Delta \vdash w : (A \oplus B)}{\Gamma, \Delta \vdash w[t/x] : (A \oplus B)}\,\mathbf{Subst} \qquad \Sigma, y : A \vdash u : D}\,\&\mathbf{L1}$$

$$\Gamma, \Sigma, \Delta \vdash \text{ let } \langle y, \_ \rangle \text{ be } w[t/x] \text{ in } u : D$$

$\oplus$L,formula descends from the left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots \qquad \vdots \qquad \vdots}{\dfrac{x : C, \Delta \vdash w : (A \oplus B) \quad \Sigma, y : A \vdash u : D \quad \Sigma, j : B \vdash v : D}{x : C, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\oplus\mathbf{L}}}{\Gamma, \Sigma, \Delta \vdash (\text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v)[t/x] : D}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \quad x : C, \Delta \vdash w : (A \oplus B)}{\Gamma, \Delta \vdash w[t/x] : (A \oplus B)}\mathbf{Subst} \qquad \begin{array}{c}\vdots \\ \Sigma, y : A \vdash u : D\end{array} \qquad \mathcal{A}}{\Sigma, \Gamma, \Delta \vdash \text{ case } w[t/x] \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\oplus\mathbf{L}$$

Where for space reasons, the $\mathcal{A}$ stands for the proof:

$$\begin{array}{c}\vdots \\ \Sigma, j : B \vdash v : D\end{array}$$

&R

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots \qquad \vdots}{\dfrac{x : C, \Sigma \vdash u : A \quad x : C, \Sigma \vdash v : B}{x : C, \Sigma \vdash \langle u, v \rangle : (A \& B)}\&\mathbf{R}}}{\Gamma, \Sigma \vdash \langle u, v \rangle[t/x] : (A \oplus B)}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \quad x : C, \Sigma \vdash u : A}{\Gamma, \Sigma \vdash u[t/x] : A}\mathbf{Subst} \qquad \dfrac{\Gamma \vdash t : C \quad x : C, \Sigma \vdash v : B}{\Gamma, \Sigma \vdash v[t/x] : B}\mathbf{Subst}}{\Gamma, \Sigma \vdash \langle u[t/x], v[t/x] \rangle : (A \& B)}\&\mathbf{R}$$

$\oplus$R2

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots}{x : C, \Sigma \vdash u : B}}{\dfrac{}{x : C, \Sigma \vdash inr(u) : (A \oplus B)}\oplus\mathbf{R}}\,\mathbf{Subst}$$

$$\Gamma, \Sigma \vdash inr(u)[t/x] : (A \oplus B)$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots}{x : C, \Sigma \vdash u : B}}{\Gamma, \Sigma \vdash u[t/x] : B}\,\mathbf{Subst}}{\Gamma, \Sigma \vdash inr(u[t/x]) : (A \oplus B)}\oplus\mathbf{R}$$

$\oplus$L, formula descends from the right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots \qquad \vdots \qquad \vdots}{x : C, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\,\oplus\mathbf{L}}{\Gamma, \Sigma, \Delta \vdash (\text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v)[t/x] : D}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Delta \vdash w : (A \oplus B)} \qquad \dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \quad x : C, \Sigma, y : A \vdash u : D}\,\mathbf{Subst}}{\Gamma, \Sigma, y : A \vdash u[t/x] : D} \qquad \mathcal{A}}{\Gamma, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u[t/x], inr(j) \Rightarrow v[t/x] : D}\,\oplus\mathbf{L}$$

Where for space reasons $\mathcal{A}$ stands for the proof:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\vdots}{x : C, \Sigma, j : B \vdash v : D}}{\Gamma, \Sigma, j : B \vdash v[t/x] : D}\,\mathbf{Subst}$$

A, formula descends from left

$$\dfrac{\Gamma \vdash v : C \qquad \dfrac{\vdots \qquad \vdots}{x : C, \Delta \vdash t : (A \multimap B) \qquad \Sigma \vdash u : A}\,\mathbf{A}}{\Gamma, \Sigma, \Delta \vdash (tu)[v/x] : B}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash v : C \qquad x : C, \Delta \vdash t : (A \multimap B)}{\Gamma, \Sigma \vdash t[v/x] : (A \multimap B)}\,\mathbf{Subst} \qquad \vdots}{\Gamma, \Sigma, \Delta \vdash ((t[v/x])u) : B}\,\mathbf{A}$$

$\oplus$**R1**

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots}{x : C, \Sigma \vdash u : A}}{\Gamma, \Sigma \vdash inl(u)[t/x] : (A \oplus B)}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad x : C, \Sigma \vdash u : A}{\Gamma, \Sigma \vdash u[t/x] : A}\,\mathbf{Subst}}{\Gamma, \Sigma \vdash inl(u[t/x]) : (A \oplus B)}\,\oplus\mathbf{R}$$

$\multimap$R

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Sigma, s : C, x : A \vdash u : B}{\Sigma, s : C \vdash \lambda x.u : (A \multimap B)}\multimap\mathbf{R}}{\Sigma, \Gamma \vdash (\lambda x.u)[t/s] : (A \multimap B)}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad \Sigma, s : C, x : A \vdash u : B}{\Sigma, \Gamma, x : A \vdash u[t/s] : B}\mathbf{Subst}}{\Sigma, \Gamma \vdash \lambda x.(u[t/s]) : (A \multimap B)}\multimap\mathbf{R}$$

The formula $\lambda x.(u[t/s])$ is identical to the formula $(\lambda x.u)[t/s]$ since x may not occur in t.

A, formula descends from right

$$\dfrac{\Gamma \vdash v : C \qquad \dfrac{\Delta \vdash t : (A \multimap B) \qquad x : C, \Sigma \vdash u : A}{x : C, \Sigma, \Delta \vdash (tu) : B}\mathbf{A}}{\Sigma, \Gamma, \Delta \vdash (tu)[v/x] : B}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\Delta \vdash t : (A \multimap B) \qquad \dfrac{\Gamma \vdash v : C \qquad x : C, \Sigma \vdash u : A}{\Gamma, \Sigma \vdash u[v/x] : A}\mathbf{Subst}}{\Sigma, \Gamma, \Delta \vdash (t(u[v/x])) : B}\mathbf{A}$$

$\otimes$L, formula descends from the right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Delta \vdash r : (A \otimes B) \qquad \Sigma, x : A, y : B, s : C \vdash u : D}{\Sigma, \Delta, s : C \vdash \text{ let } (x \times y) \text{ be } r \text{ in } u : D}\otimes\mathbf{L}}{\Sigma, \Delta, \Gamma \vdash (\text{ let } (x \times y) \text{ be } r \text{ in } u)[t/s] : D}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\Delta \vdash r : (A \otimes B) \qquad \dfrac{\Gamma \vdash t : C \qquad \Sigma, x : A, y : B, s : C \vdash u : D}{\Sigma, x : A, y : B, \Gamma \vdash u[t/s] :}\mathbf{Subst}}{\Sigma, \Delta, \Gamma \vdash \text{ let } (x \times y) \text{ be } r \text{ in } (u[t/s]) : D}\otimes\mathbf{L}$$

$\otimes$L, formula descends from the left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\Delta, s : C \vdash r : (A \otimes B) \qquad \Sigma, x : A, y : B \vdash u : D}{\Sigma, \Delta, s : C \vdash \text{ let } (x \times y) \text{ be } r \text{ in } u : D}\otimes\mathbf{L}}{\Sigma, \Delta, \Gamma \vdash (\text{ let } (x \times y) \text{ be } r \text{ in } u)[t/s] : D}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad \Delta, s : C \vdash r : (A \otimes B)}{\Delta, \Gamma \vdash r[t/s] : (A \otimes B)}\mathbf{Subst} \qquad \Sigma, x : A, y : B \vdash u : D}{\Sigma, \Delta, \Gamma \vdash \text{ let } (x \times y) \text{ be } (r[t/s]) \text{ in } u : D}\otimes\mathbf{L}$$

$\otimes$R, formula descends from right

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots \qquad \vdots}{x : C, \Sigma \vdash u : A \qquad \Delta \vdash v : B}\otimes\mathbf{R}}{\Gamma, \Sigma, \Delta \vdash (u \times v)[t/x] : (A \otimes B)}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\Gamma \vdash t : C \qquad x : C, \Sigma \vdash u : A}{\Gamma, \Sigma \vdash u[t/x] : A}\mathbf{Subst} \qquad \vdots}{\Gamma, \Sigma, \Delta \vdash (u[t/x] \times v) : (A \otimes B)}\otimes\mathbf{R}$$

The formula $(u[t/x] \times v)$ is identical to the formula $(u \times v)[t/x]$ since x may not occur in v.

$\otimes$R, formula descends from left

$$\dfrac{\Gamma \vdash t : C \qquad \dfrac{\vdots \qquad \vdots}{\Sigma \vdash u : A \qquad x : C, \Delta \vdash v : B}\otimes\mathbf{R}}{\Sigma, \Gamma, \Delta \vdash (u \times v)[t/x] : (A \otimes B)}\mathbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\vdots \qquad \dfrac{\vdots \qquad \vdots}{\Gamma \vdash t : C \qquad x : C, \Delta \vdash v : B}\mathbf{Subst}}{\Sigma, \Gamma, \Delta \vdash (u \times v[t/x]) : (A \otimes B)}\otimes\mathbf{R}$$

The formula $(u \times v[t/x])$ is identical to the formula $(u \times v)[t/x]$ since x may not occur in u.

Subst, formula descends from right

$$\dfrac{\dfrac{\vdots}{\Delta \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{x : C, \Sigma \vdash u : A} \quad \dfrac{\vdots}{y : A, \Gamma \vdash v : D}}{x : C, \Sigma, \Gamma \vdash v[u/y] : D}\,\textbf{Subst}}{\Delta, \Sigma, \Gamma \vdash (v[u/y])[t/x] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta \vdash t : C} \quad \dfrac{\vdots}{x : C, \Sigma \vdash u : A}}{\Delta, \Sigma \vdash u[t/x] : A}\,\textbf{Subst} \qquad \dfrac{\vdots}{y : A, \Gamma \vdash v : D}}{\Delta, \Sigma, \Gamma \vdash v[(u[t/x])/y] : D}\,\textbf{Subst}$$

The terms $(v[u/y])[t/x] :$ and $v[(u[t/x])/y] :$ are the same since x doesn't occur in v.

Subst, formula descends from left

$$\dfrac{\dfrac{\vdots}{\Delta \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{\Sigma \vdash u : A} \quad \dfrac{\vdots}{x : A, y : C, \Gamma \vdash v : D}}{\Sigma, y : C, \Gamma \vdash v[u/x] : D}\,\textbf{Subst}}{\Sigma, \Delta, \Gamma \vdash (v[u/x])[t/y] : D}\,\textbf{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Sigma \vdash u : A} \qquad \dfrac{\dfrac{\vdots}{\Delta \vdash t : C} \quad \dfrac{\vdots}{x : A, y : C, \Gamma \vdash v : D}}{x : A, \Delta, \Gamma \vdash v[t/y] : D}\,\textbf{Subst}}{\Sigma, \Delta, \Gamma \vdash (v[t/y])[u/x] : D}\,\textbf{Subst}$$

The terms $(v[u/x])[t/y] :$ and $(v[t/y])[u/x] :$ are the same since x isn't in t, and y isn't in u.

Lemma 7.3.80 (Reduce-One-Subst) *Given a proof of the sequent* $\Gamma \vdash A$ *in* NAT *which ends in an application of* **Subst** *of degree* d*, and where the degree of the proofs of both hypothesis is less than* d*, we may construct a proof of* $\Gamma \vdash A$ *in* NAT *of degree less than* d*.*

Proof. By induction on the size of the proof of $\Gamma \vdash A$.

Given a derivation which ends in a **Subst**, we perform case analysis on the rule which is applied immediately above the **Subst** on the right hand branch.

In each case we will provide a reduction, which may eliminate the subst entirely, or replace it with one or two smaller substs. Since this is a proof by induction on the size of a derivation, one may view this proof as a procedure which pushes **Subst**s of large degree up a derivation. Informally, this procedure pushes **Subst**s up through a derivation until the critical point is reached where the right branch of the proof above the subst is simply identity or some constant rule.

We now give the reductions for every proof rule on the right branch.

I

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : A\end{array} \qquad \dfrac{}{x : A \vdash x : A}\,\mathbf{I}}{\Gamma \vdash x[t/x] : A}\,\mathbf{Subst}$$

$$\Downarrow$$

$$\begin{array}{c}\vdots\\ \Gamma \vdash t : A\end{array}$$

The formula t is identical to the formula $x[t/x]$ by the definition of substitution.

7.3.4 NAT Subst Elimination

The subst elimination theorem states that whatever can be proven in NAT can also be proven without the use of the **Subst** rule.

The following detailed demonstration of the subst elimination theorem consists of a proof normalization procedure which slowly eliminates subst from any NAT proof. The procedure may increase the size of the proof, although of course it will still be a proof of the same sequent.

We will call a formula which appears in a hypothesis of an application of **Subst** but which does not occur in the conclusion a *subst formula*. In the list of NAT rules in appendix E the subst formula in the **Subst** rule is the formula named A.

We also define the *degree* of a **Subst** to be the number of symbols in its subst formulas. For concreteness, we define here what is meant by number of symbols. We will consider each propositional symbol p_i to be a single symbol. We also consider the negation of each propositional symbol $p_i{}^{\perp}$ to be a single symbol. Finally, we count each connective and constant, $\otimes, \multimap, \oplus, \&, ?, !, 1, \perp, 0, \top$, as a single symbol, but do not count parentheses. It is important to note that negation is defined, and therefore is not a connective. We also define the *degree* of a proof to be the maximum degree of any subst in the proof, or zero if there are no substs.

Operationally, the subst elimination procedure defined below first finds one of the "highest" substs of maximal degree in the proof. That is, an application of **Subst** for which all applications of **Subst** in the derivation of both hypotheses are of smaller degree. Then a reduction is applied to that occurrence of **Subst**, which simplifies or eliminates it, although it may replicate some other portions of the original proof. We iterate this procedure to remove all substs of some degree, and then iterate the entire procedure to eliminate all substs. In this way, any NAT proof may be normalized into one without any uses of the **Subst** rule, at the possible expense of an exponential blowup in the size of the resulting proof tree.

Technically, we begin with a lemma which constitutes the heart of the proof of subst-elimination. Although the proof of this lemma is rather lengthy, the reasoning is straightforward, and the remainder of the proof of subst-elimination is quite simple.

NAT2, and for Theorem **??** we need some cut-like rule. These differences lead to subtlely different properties of NAT and NAT2 systems. For example, since **Subst** is syntaxless in NAT, one can show that that one need not consider **Subst** in searching for type derivations. On the other hand, NAT2 is entirely driven by term syntax, leading to a unique principle type theorem.

intuitionistic sequent calculus and natural deduction [82, Appendix A]. The main idea is to interpret sequent proof rules as instructions for constructing natural deduction proofs. The sequent rules acting on the left determine constructions on the top (hypotheses) of a natural deduction proof, while sequent rules acting on the right extend the natural deduction proof from the bottom (conclusion). The **Cut** rule is interpreted by substituting a proof for a hypothesis. In order for this to work, the natural deduction proof system must have the substitution property formalized by the derived **Subst** rule in Appendix E.

A simple example that illustrates the general pattern is the tensor rule acting on the left, $\otimes$**L**. In the conclusion of the sequent rule, there is a new term variable $z : (A \otimes B)$. However, we want natural deduction proofs to be closed under the operation of substituting terms for variables (hypotheses). If we use **Cut** in a sequent proof to replace $z : (A \otimes B)$, we end up with a sequence of proof steps whose hypotheses and conclusion are identical to the antecedent and consequent of the natural deduction $\otimes$**L** rule in Appendix E.

The most unusual rule of the NAT system is the **!SR** rule. This may be understood by considering the **!SR** rule of SEQ and remembering that when we substitute proofs for hypotheses in NAT, we must still have a well-formed natural deduction proof. If we follow SEQ **!SR** with **Cut**, we may use $\Delta \vdash t : !B$ and $x : !B, !\Sigma \vdash u : A$ to prove $\Delta, !\Sigma \vdash \texttt{store}\ u[t/x] : !A$. Generalizing to any number of **Cut**'s, so that natural deduction proofs will be closed under substitution, we obtain the **!SR** rule in Appendix E. This rule may without loss of generality be restricted to the case where all the Δ_i contain some non-! type.

The third and final system we consider is called NAT2. The NAT2 rules are generated from the NAT rules by removing the rules of **Subst** and **!SR** from NAT and replacing them with:

$$\frac{\Sigma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Sigma, \Gamma \vdash \ let_{cut}\, x\ \texttt{be}\ t\ \texttt{in}\ u : B} \ let_{cut}$$

$$\frac{!\Sigma \vdash t : A}{!\Sigma \vdash \ \texttt{store}\ t : !A} \ \textbf{!SR}$$

The reason for the explicit syntax of let_{cut} is that **Subst** is not a derived rule of

at the bottom, and the leaves at the top. Each branch of a deduction is a sequence of applications of the proof rules, some of which, such as $\otimes R$ in SEQ, represent branching points in the deduction tree, some, such as $\multimap R$, which extend the length of a branch, and some, such as identity, which terminate a branch. Any branch not terminated by identity or 1 **R** is called an assumption. The leaves therefore embody the type assumptions and the root the conclusion. Such a structure is said to be a deduction of the conclusion from the assumptions. A *proof* is a typing deduction with no assumptions. That is, all the branches terminate with an application or identity or 1 **R**. A closed linear term t is said to be *linearly typable* if there exists some proof with conclusion $\vdash t : A$ for some linear formula A.

7.3.3 The typing rules

The first system we will study is called SEQ, the rules for which are given in Appendix D. This formulation is due mainly to Abramsky [1]. We have modified the syntax used in the original presentation slightly, but the idea is the same: take the rules for (intuitionistic) propositional linear logic and decorate them with linear terms. In the system SEQ, cut-free derivations produce linear terms in normal form. Some derivations with cut correspond to linear terms in non-normal form, and cut-elimination steps transform the term, essentially performing beta-reduction and other linear reduction steps. Performing cut-elimination on an SEQ proof is analogous to reduction in the linear calculus, although the exact correspondence is somewhat complicated.

The typing rules for the second system, called NAT, are given in Appendix E. The main difference between the two systems is that NAT is more "term-driven", while SEQ is more "type-driven". In Section **??**, we show that NAT is equivalent to SEQ for proving typing judgements. This is not surprising since NAT is based on a Gentzen-style sequent calculus presentation of natural deduction for intuitionistic linear logic, while SEQ is based on a sequent calculus presentation of the same logic. The term "decorations" have been chosen in NAT so that the provable sequents in these systems are the same.

In devising the NAT rules from SEQ, we were guided by the correspondence between

All the `let A be B in C` constructs bind variables in A by pattern-matching A against the result of evaluating B, and then evaluate C. For example, consider the term `let` $x \times y$ `be` $(1 \times ((\lambda x.x)1))$ `in` $(x \times y)$. First the subject term is evaluated (to 1×1), then x and y are bound (both to 1), and finally $(x \times y)$ is evaluated (in the extended context) producing a final result of (1×1).

The term `store` u is a reusable, or delayed version of u. The `copy` operation inserts multiple copies of a term `store` u, while `discard` completely eliminates a `store` u term. These `copy` and `discard` operations may be implemented by pointer manipulations (implementing sharing) or by explicit copying. The `read` construct forces evaluation of a `store` d term. The interaction between `read` and `store` is the critical point where the linear calculus determines reduction order. In other terminology, `store` is a wrapper or box which is only opened when the term must be `read`.

The reduction rules for linear lambda calculus are given in Appendix C. A linear term t *reduces* to a linear term s if $t \twoheadrightarrow s$ can be inferred from the linear calculus evaluation rules. Abramsky has demonstrated determinacy for a more restricted form of reduction relation in [1]. The reduction relation given in Appendix C only allows "nonlinear" reductions (involving the **!WL**, **!DL**, and **!CL** reduction rules) to apply at the "top level" of a term, in the empty context. However, the remaining "linear" reduction steps may be applied anywhere in a term. Thus reduction is neither a congruence with respect to all term formation rules, nor is it deterministic. With Mitschke's δ-reduction theorem [12] this reduction system can be shown to be confluent on untyped terms, even though not all untyped terms have a normal form. For typed terms, the usual cut-elimination procedure provides a proof of weak normalization for this reduction system.

7.3.2 Typing preliminaries

We review some standard definitions. A *type multiset* or *type environment* is a multiset of pairs $x_i : A_i$ of variables x_i and linear logic formulas A_i. A *typing judgement* is a type multiset Γ, a single linear term t, and a single linear logic formula A, separated by a $\vdash$, constructed as follows: $\Gamma \vdash t : A$. A *typing deduction* is a tree, presented with the root

$term$ ::=	x	variable
|	$let_{cut}\ x$ `be` t `in` u	bind x to result of t in u
|	`let` $x \times y$ `be` t `in` u	bind x to car, y to cdr of t in u
|	$(t \times u)$	eager pair (like cons in ML)
|	$(t\,u)$	application
|	$(\lambda x.t)$	abstraction
|	$inr(t)$	determines right branch of case
|	$inl(t)$	determines left branch of case
|	`case` t `of` $inl(x) \Rightarrow u, inr(y) \Rightarrow v$	evaluate t then branch
|	$\langle t, u \rangle$	lazy pair
|	`let` $\langle \_, x \rangle$ `be` t `in` u	bind x to cdr of lazy pair
|	`let` $\langle x, \_ \rangle$ `be` t `in` u	bind x to car of lazy pair
|	`let` 1 `be` t `in` u	evaluate t to 1, then become u
|	`store` u	store or delay u
|	`discard` t `in` u	throw away t
|	`read store` x `as` t `in` u	evaluate t, bind x
|	`copy` $x@y$ `as` t `in` u	binds x and y to t

Figure 7.1: Grammar of the Linear Lambda Calculus

7.3.1 Linear terms and reduction

The linear calculus may be considered a functional programming language with fine-grained control over the use of data objects. To a first approximation, no function may refer to an argument twice without explicitly copying, nor ignore an argument without explicitly discarding it. The reason we say, "to a first approximation" is that the notion of referring to an argument twice is somewhat subtle, especially in the presence of additive type connectives. For example, a variable should appear "once" in both branches of a case statement, which is an additive operator. A more precise understanding of the restrictions on use and discard may be gained from reading the typing rules. In our presentation of the linear calculus, we have not restricted reduction order completely (in contrast to [1], for example.) However, as pointed out in Section 7.2, the order of certain reductions must be determinate.

Using x, y, z for term variables, and t, u, v for terms, the syntax of linear lambda terms are summarized by the grammar given in Figure 7.1.

after a function application.

7.3 The Linear Calculus

We describe the terms of linear lambda calculus in Section 7.3.1 and give three sets of typing rules. The first, SEQ, given in Appendix D, is the standard set of rules given by applying the Curry-Howard isomorphism to Girard's sequent calculus proof system, restricted to intuitionistic linear logic. The second system, NAT, given in Appendix E, is based on a Gentzen-style sequent calculus presentation of natural deduction for intuitionistic linear logic. The third system, NAT2, is closely related to NAT, differing only in the **!SR** and **Subst** rules. The difference between NAT and NAT2 lies in the point of view taken on whether the **!SR** rule is a "left" rule or "right" rule of the sequent calculus: it introduces a type constructor on the right, so it appears to be a "right" rule, while it depends on the form of the context, or left side of a sequent, so it may also be a "left" rule. Traditionally, right rules of sequent calculus and introduction rules of natural deduction systems are analogous, while left rules of sequent calculus correspond to a combination of elimination rules and substitution. Consequently, the translation of left rules into natural deduction should be closed under substitution while the translation of right rules should be direct.

Other researchers have independently formulated similar typing rules, although none we know of incorporate a rule of the form of the **!SR** rule of NAT. Lafont, Girard, Abramsky, and others have studied systems very similar to SEQ [35, 1]. In recent unpublished notes [2, 93] and an MS thesis [65], systems close to NAT2 have been studied. Walder also discusses alternative rules for **!SR** and the implications of syntaxless **Subst** rule in the context of a NAT-like system. We take this parallel development of ideas as evidence that these are natural formulations of type systems based on linear logic. We show that NAT and SEQ give the same set of types to each linear term in Section **??**, while NAT2 provides equivalence only up to a point. Technical theorems showing the "term-driven" nature of NAT and NAT2 and the "type-driven" nature of SEQ are proved in Section **??**.

the outer term. In the first case neither r nor s are used at all. In the second case they are both used, but the result is not.

The first general conclusion that follows from this example is that not all terms have a most general type with respect to substitution. This is evident since we have two types for the example term such that neither is a substitution instance of the other, and it can be checked that no shorter type is derivable for this term. Moreover, the two types are disjoint in this system: we can find terms of each type that do not have the other type.

A second conclusion follows from comparing the informal operational readings of each typing with the reduction rules of lambda calculus. In particular, consider the type $A \multimap (C \multimap !B) \multimap C \multimap A$. We may understand the correctness of this type by saying that we apply the second argument to the third and then discard the resulting discardable value. However, this informal reading assumes a particular reduction order. By the usual reduction rules of lambda calculus, we may obtain a term

$$\lambda q.\, \lambda r.\, \lambda s.\, q$$

in which the second and third arguments do not occur. Since this term does not have the linear type
$A \multimap (C \multimap !B) \multimap C \multimap A$, the subject reduction property fails for this simplified system. This is a serious problem, since we always expect types to be preserved by reduction. If types are not preserved by reduction, then reduction of well-typed terms may lead to terms that are not well-typed. Essentially, this means that static typing does not prevent run-time type errors.

Intuitively, the failure of subject reduction seems to result from the contrast between a careful accounting of resources in linear logic and the inherent ambiguity in reduction order in the λ calculus. This implies that reduction order must be restricted in some way. The most natural approach seems to be to introduce additional constants that indicate where the operations associated with ! types are performed. This restricts the set of types in a way that makes type inference possible and also provides a convenient framework for restricting evaluation order. In particular, using explicit discard, we may say explicitly, inside a term, whether a discard happens before or

function of type $A \multimap B$ must use its argument of type A "exactly once" in producing a result of type B. However, if A is of the form $!C$, then the copy and discard rules associated with ! types allow us to define functions that use their argument zero or more times.

All of the intuitive points we will consider may be illustrated using the the λ term

$$\lambda\, q.\, \lambda\, r.\, \lambda\, s.\, (\lambda x.q)(r\, s)$$

The subterm $(\lambda x.q)$ must have a linear type of the form

$$(\lambda\, x.q) : (!B \multimap A),$$

since only arguments of ! type need not appear in the body of a function. Consequently, the application $(r\, s)$ must have type $!B$. There are two possible types of r and s. One is that r is a non-discardable function that produces discardable output. That is, $r : (C \multimap !B)$, $s : C$. The other possibility is more subtle and requires more detailed understanding of the type system. If r and s are both discardable resources, such as $r : !(!C \multimap B)$ and $s : !C$, then by the usual application rule we have $rs : B$. However, whenever we have an expression of type B such that all variables appearing in the term have a type beginning with !, the **!SR** rule allows use to conclude that the term has type $!B$. Using our concepts of linear and nonlinear memory, the **!SR** rule may be explained by saying that if we define a value of type B by referring only to values in non-linear memory, the value we define may reside in non-linear memory, and have type $!B$.

From the discussion above, we can see that there are two types for the example λ-term:

$$A \multimap !(!C \multimap B) \multimap !C \multimap A$$

$$A \multimap (C \multimap !B) \multimap C \multimap A$$

Associated with these types are two different orders of evaluation. The first is the type of the function that reduces the outermost application first; the application $(r\, s)$ is thrown away before it is ever evaluated. The second type is the type of the function that first reduces the inner term $(r\, s)$, to obtain a discardable value, and then reduces

syntax of linear lambda calculus and the typing rules. SEQ and NAT are proved equivalent in Section 7.3, where we also prove complementary term subformula and type subformula properties for the two proof systems. A type inference algorithm and proof of most general typing are given in Section ??, with the subject reduction property proved in Section ??. The remaining sections of this chapter discuss our execution model and TIM-based implementation.

7.2 Why explicit storage operations?

In the pure lambda calculus (typed or untyped), there are no explicit `store`, `read`, `copy`, or `discard` primitives. The usual implementations of languages based on lambda calculus perform these operations as needed, according to one of several possible strategies. In other words, these operations are implicit in the language but explicit in the implementation. Since `store`, `read`, `copy`, and `discard` are explicit in the proof system of linear logic, we might attempt to insert these operations into lambda terms as part of inferring linear logic types. This was part of the program we started to follow in collaboration with Scedrov in 1989, before discovering that this seemed to require algorithms for deciding provability properties of propositional linear logic; this led to the study of decision problems reported in [60]. In the remainder of this motivational section, we sketch two particular problems that arise, namely, the lack of a natural form of principal type and the failure of subject reduction theorem. The first, along with the undecidability results of [60], suggests that the process of inferring types will be algorithmically tractable only if additional operations or typing constraints are added to lambda calculus. The failure of subject reduction reinforces this conclusion by showing that additional operations are needed in the language to determine the order of function application and discarding of data.

In this section, we consider a type system derived from the NAT rules, in Appendix E, by modifying each rule whose name begins with ! so that the term in the consequent is the same as the term in the antecedent. This has the effect of assigning $\multimap, !$ types to pure lambda terms in a way that allows ! operations to be done implicitly at any point in the evaluation of terms. The main properties of this system are that a

system very close to NAT with additional syntax embedded in the **!SR** rule. Their system exhibbits a unique most general type property, and may have additional desirable properties [16]. Walder also discusses alternative rules for **!SR**. Since these systems are not equivalent to SEQ and NAT, it seems an important research problem to evaluate the trade-offs between the systems.

In the final part of this chapter, we explore implementations of the linear lambda calculus. One problem with Lafont's method of eliminating garbage collection is that it requires a tremendous amount of duplication. Essentially, in comparison with a standard reference counting scheme, garbage collection is eliminated by making every datum have reference count one. This is achieved by copying the datum whenever we would otherwise increment the reference count. A consequence is that there is a significant increase in the amount of storage space required. We believe that in practice, it is useful to consider the trade-offs between copying and garbage collection. In particular, if a datum is large, then copying it even a small number of times may be prohibitive, and may outweigh the benefit of suspending garbage collection. In order to explore such trade-off's in a general setting, we have developed an implementation with two forms of memory, called "linear" and "non-linear" memory. Within this framework, we eliminate garbage collection in linear memory but retain traditional garbage collection techniques in non-linear memory. Similarly, we may perform array update in place on arrays in linear memory. Our implementation is based on an extension of the "three instruction machine" (TIM) [27] with additional operations of DELAY, FORCE, COPY, and DISCARD to provide explicit control over evaluation order and storage management, and arrays that are updated in linear memory. The implementation of our abstract machine is written in Common Lisp, with garbage collection in non-linear memory handled by the Lisp garbage collector. This work is similar to that of Wakeling and Runciman [95], who study linear modifications to the G-Machine [46], and suggest studying the spineless G-Machine, which is closely related to the TIM.

In the following section, we describe the problems that arise in using linear logic formulas as types for pure lambda terms. This motivates the use of linear lambda calculus with explicit copy and discard primitives. In Section 7.3, we present the

on to generalize this system into one incorporating quantifiers and full linear logic, a move which enabled him to interpret linear types in terms of concurrent computations. Recently Chirimar, Gunter, and Riecke [21] have implemented a version of the linear calculus. In this chapter we restrict our attention to intuitionistic linear logic.

One important property of type systems is *subject reduction,* which states that if a term t has type A, then any term produced by any number of reduction (evaluation) steps still has type A. This is crucial if we wish to use types to statically determine execution properties of terms. While it may be possible to prove subject reduction for a type system based on sequent calculus rules, there is a significant technical obstacle. If we wish to reason about the effect of reduction, we need to understand the connection between the syntactic form of a term and the set of possible types. However, with sequent calculus rules such as SEQ, a single term may have typing proofs of many different forms. (This is because uses of **Cut**, which are essential for typing terms not in normal form, are not reflected in the syntax of terms.) To avoid this problem, we formulate an equivalent set of natural deduction style typing rules, called NAT. This system has the property that for each form of linear term, there is exactly one typing rule that may be used to give a type. Using the natural deduction typing rules, subject reduction may be proved by traditional means. In addition, with syntax-directed typing rules, it is possible to formulate a unification-based algorithm that determines the most general types of any linear lambda term.

An interesting property of NAT is that one essential rule, **!SR,** based on the modal operator ! of linear logic, involves substitution into terms. Since a term may be written as the result of substitution in many different ways, this rule gives us a system in which a term may have several different principal linear types. Of course, since NAT is equivalent to SEQ, this is not an idiosyncrasy of our presentation, but a property shared by Abramsky's system SEQ that seems inherent to linear logic. If we simplify the **!SR** typing rule of NAT, we obtain an *inequivalent* system, which we call NAT2. If we restrict reduction to closed terms, then subject reduction holds for this system. However, the provable typing judgements are not closed under substitution. Essentially this system has been studied by [65], who proves the existence of unique most-general types. Benton, Bierman, de Paiva, and Hyland have recently studied a

Chapter 7

Linear ML^{--}

In this chapter a declarative language is presented which is based on linear logic through a Curry-Howard style correspondence. Very similar languages have been studied in the past, but in this chapter we present a type-soundness (subject-reduction) theorem, and most general type theorem. Further, we present a novel implementation technique based on a two-space memory model: in the "linear" space there is no need for garbage collection, and destructive update in place may occur, while in the "nonlinear" space, some form of dynamic memory management is needed, and update in place is not always applicable. Finally, an implementation of linear ML^{--} is presented which is based on the Three Instruction Machine (TIM).

7.1 Introduction

Historically, intuitionistic logic has been the basis for type systems, via the Curry-Howard isomorphism, or "formulas-as-types" principle [43]. Through this isomorphism, intuitionistic proofs of propositions may be viewed as functional programs, and logical propositions may viewed as types. A similar use of linear logic has been initiated by Girard and Lafont, and Abramsky [35, 1]. In [35], a linear calculus was developed which effectively determines reduction order, while explicitly marking the points where contraction and weakening are used. Abramsky further defined a type inference system, here called SEQ, which is discussed in Section 7.3.3. Abramsky went

6.8 Summary of Chapter

In this chapter the PSPACE-completeness of MALL is exploited in order to achieve a logical embedding of propositional intuitionistic implication into IMALL. Essentially, the linear restriction of IMALL is satisfied by translation through IIL*, a logic without contraction. Weakening can be encoded in IMALL with additive connectives and constants, but contraction would pose a problem if not for this intermediate logic. Hudelmaier has recently made use of a very similar logic to demonstrate new bounds on cut-elimination procedures for intuitionistic logic, Martini and Masini have recently investigated the translation of classical logic into linear logic using similar techniques, and there may be more interest in logics related to IIL* [44, 68].

Assuming that there was a proof of $\Gamma \vdash F^+[A]$, one simply cuts this against the proof of $(A \supset x), F^+[A] \vdash F^+[x]$ guaranteed by Lemma 6.6.76, and thus obtains a proof of $\Gamma, (A \supset x) \vdash F^+[x]$.

Assuming that there was a proof of $\Gamma \vdash F^-[A]$, one simply cuts this against the proof of $(x \supset A), F^-[A] \vdash F^-[x]$ guaranteed by Lemma 6.6.77, and thus obtains a proof of $\Gamma, (x \supset A) \vdash F^-[x]$. ■

For completeness, we state:

Lemma 6.6.79 *For any intuitionistic sequent ρ , $\Xi(\rho)$ is computable in polynomial time and its size is linear in ρ.*

6.7 Discussion

This embedding of the implicational fragment of propositional intuitionistic logic in the IMALL fragment of linear logic provides an alternative proof for the PSPACE-hardness of IMALL. More importantly, it provides insight into the use and elimination of the structural rules from IIL through the embedding of IIL into IIL*. The system IIL* is an interesting optimization of intuitionistic logic that could be useful in theorem proving and logic programming applications [72].

A number of related questions remain open. An extension of our techniques to all intuitionistic propositional connectives should be investigated. On the other hand, it would be interesting to know whether there is an embedding of intuitionistic implication in IMALL that preserves the structure of all cut-free proofs. It would also be interesting to investigate the connections between IIL*, IMALL, and Hudelmaier's systems [44]. It is worth examining what transformations such as depth reduction mean at the level of proof terms given by the Curry-Howard isomorphism (discussed in Chapter 7, and whether there are some useful optimizations in the evaluation of proof terms arising from such a study.

Lemma 6.6.75 *For any* IIL *formula A, if a sequent involving a proposition p_i is provable in* IIL*, then that sequent with p_i replaced with A is also provable in* IIL*.*

The main lemma regarding the soundness of depth reduction relies on the following two lemmas, which are easily shown by simultaneous induction on the structure of F:

Lemma 6.6.76 *For all* IIL *formulas A and B, all sequence Γ, and positive contexts $F^+[\,]$, the sequent $\Gamma, A \supset B, F^+[A] \vdash F^+[B]$ is provable in* IIL*.*

Lemma 6.6.77 *For all* IIL *formulas A and B, all sequence Γ, and negative contexts $F^-[\,]$, the sequent $\Gamma, B \supset A, F^-[A] \vdash F^-[B]$ is provable in* IIL*.*

The soundness of depth reduction follows:

Lemma 6.6.78 *A sequent $\Gamma \vdash A$ is provable in* IIL *if and only if $\Xi(\Gamma \vdash A)$ is provable in* IIL*.*

Proof. The argument is by induction on the steps of transformation Ξ applied to $\Gamma \vdash A$. Each of the individual transformations may be written in one of four forms:

$$\begin{array}{rcl} \Gamma, F^-[A] \vdash B & \Rightarrow & \Gamma, (A \supset x), F^-[x] \vdash B \\ \Gamma, F^+[A] \vdash B & \Rightarrow & \Gamma, (x \supset A), F^+[x] \vdash B \\ \Gamma \vdash F^+[A] & \Rightarrow & \Gamma, (A \supset x) \vdash F^+[x] \\ \Gamma \vdash F^-[A] & \Rightarrow & \Gamma, (x \supset A) \vdash F^-[x] \end{array}$$

In the *if* direction, assuming we have a proof of the transformed sequent, we simply apply lemma 6.6.75, and we have a proof of the desired sequent with the unpleasant addition of the formula $(A \supset A)$ in the context. Since $\vdash (A \supset A)$ is provable we may cut against this to achieve the desired proof.

In the *only if* direction, there are four cases, although they are all very similar.

Assuming that there was a proof of $\Gamma, F^-[A] \vdash B$, one simply cuts this against the proof of $(A \supset x), F^-[x] \vdash F^-[A]$ guaranteed by Lemma 6.6.77, and thus obtains a proof of $\Gamma, (A \supset x), F^-[x] \vdash B$.

Assuming that there was a proof of $\Gamma, F^+[A] \vdash B$, one simply cuts this against the proof of $(x \supset A), F^+[x] \vdash F^+[A]$ guaranteed by Lemma 6.6.76, and thus obtains a proof of $\Gamma, (x \supset A), F^+[x] \vdash B$.

$$
\begin{array}{rcl}
\Gamma, (A \supset B) \supset (C \supset D) \vdash Z & \Rightarrow & x \supset (C \supset D), \Gamma, (A \supset B) \supset x \vdash Z \\
\Gamma, p_i \supset ((A \supset B) \supset C) \vdash Z & \Rightarrow & (A \supset B) \supset x, \Gamma, p_i \supset (x \supset C) \vdash Z \\
\Gamma, p_i \supset (A \supset (B \supset C)) \vdash Z & \Rightarrow & x \supset (B \supset C), \Gamma, p_i \supset (A \supset x) \vdash Z \\
\Gamma, ((A \supset B) \supset C) \supset p_i \vdash Z & \Rightarrow & x \supset (A \supset B), \Gamma, (x \supset C) \supset p_i \vdash Z \\
\Gamma, (A \supset (B \supset C)) \supset p_i \vdash Z & \Rightarrow & (B \supset C) \supset x, \Gamma, (A \supset x) \supset p_i \vdash Z \\
\Gamma \vdash (A \supset B) \supset (C \supset D) & \Rightarrow & (C \supset D) \supset x, \Gamma \vdash (A \supset B) \supset x \\
\Gamma \vdash p_i \supset (A \supset (B \supset C)) & \Rightarrow & (B \supset C) \supset x, \Gamma \vdash p_i \supset (A \supset x) \\
\Gamma \vdash p_i \supset ((A \supset B) \supset C)) & \Rightarrow & x \supset (A \supset B), \Gamma \vdash p_i \supset (x \supset C) \\
\Gamma \vdash (A \supset (B \supset C)) \supset p_i & \Rightarrow & x \supset (B \supset C), \Gamma \vdash (A \supset x) \supset p_i \\
\Gamma \vdash ((A \supset B) \supset C)) \supset p_i & \Rightarrow & (A \supset B) \supset x, \Gamma \vdash (x \supset C) \supset p_i
\end{array}
$$

Figure 6.19: Definition of Ξ

6.6.1 Depth Reduction in IIL

An IIL formula of depth one is either an atom p or has the form $(p_i \supset p_j)$. A formula of depth two is one of the form $(p_i \supset (p_j \supset p_k))$, or the form $((p_i \supset p_j) \supset p_k)$. Given a sequent $\Gamma \vdash D$, we define $\Xi(\Gamma \vdash D)$ to be the result of repeatedly applying any of the the set of transformations given in Figure 6.19 until none of them apply.

These transformations each reduce the depth of implications, at the expense of building a new implication (which is also shallower than the original). Thus this sequence of reductions always terminates. Notice that the only kinds of formulas left after the Ξ transformation are of the form: $p_i, p_i \supset p_j, p_i \supset (p_j \supset p_k)$, or $(p_i \supset p_j) \supset p_k$, where p_i, p_j, and p_k are atomic propositions. Although all the formulas appearing are very small, there may be many more of them. This technique goes back to [94], see also [76].

We define a *positive contextual formula*, written $F^+[C]$, to be a formula with a specific occurrence of a subformula C identified, which has positive polarity in the formula F. Similarly, a *negative contextual formula*, written $F^-[C]$, is one where the specific occurrence of a subformula C has negative polarity in the formula F. Note that the occurrence specified is unique. That is, even if the formula C occurs multiple times as a subformula of F, the occurrence indicated by $F^+[C]$ or $F^-[C]$ is unique.

Proposition 6.2.56 readily yields:

reasoning applies six times, leaving one with the proof displayed below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{[\Gamma_1]^-,[(B\supset C)]^-,k\vdash[(A\supset B)]^+}}{[\Gamma_1]^-,[(B\supset C)]^-\vdash k\multimap[(A\supset B)]^+}\multimap\mathbf{R}\quad\dfrac{\vdots}{[\Gamma_2]^-,k,b\vdash[D]^+}}{[\Gamma_1]^-\vdash([(B\supset C)]^-\multimap(k\multimap[(A\supset B)]^+))}\multimap\mathbf{R}\quad[\Gamma_2]^-,k\otimes b\vdash[D]^+}{[\Gamma]^-,([(B\supset C)]^-\multimap(k\multimap[(A\supset B)]^+))\multimap(k\otimes b)\vdash[D]^+}\otimes\mathbf{L}\quad\dfrac{\vdots}{[\Gamma]^-,k,[C]^-\vdash[D]^+}\multimap\mathbf{L}}{[\Gamma]^-,k\otimes[C]^-\vdash[D]^+}\otimes\mathbf{L}}{[\Gamma]^-,(([(B\supset C)]^-\multimap(k\multimap[(A\supset B)]^+))\multimap(k\otimes b))\oplus(k\otimes[C]^-)\vdash[D]^+}\oplus\mathbf{L}$$

Where $[\mathbf{\Gamma_1}]^-\bigcup[\mathbf{\Gamma_2}]^-=[\Gamma]^-$. From this proof, by induction, we can generate a IIL* proof of $[\Gamma_1]^-,[(B\supset C)]^-,k\vdash[(A\supset B)]^+$ By Proposition 6.3.65 we can generate a IIL* proof of $[\Gamma]^-,[(B\supset C)]^-,k\vdash[(A\supset B)]^+$. Then, using **Left** $\supset$ **2** with this proof and the translation (by induction) of the rightmost branch from the above proof figure, we can construct an IIL* proof of $\Gamma\vdash C$. The middle unfinished branch of the above figure is irrelevant to the translation, but happens to always be provable by Lemma 6.4.69. ■

6.6 Efficiency of Transformation

For any IIL sequent ρ we have provided an equiprovable IMALL sequent $\theta(\rho)$. This encoding into IMALL could be exponential in the size of ρ, but if ρ is of depth two or less, then $\theta(\rho)$ is linear in the size of ρ. Below we give a depth-reduction procedure that takes polynomial time and that produces a sequent $\Xi(\rho)$ of depth at most two, which is only linearly larger than ρ. The transformation $\theta(\Xi(\rho))$ therefore provides an argument for the PSPACE-hardness of the decision problem for IMALL. The argument for membership of this problem in PSPACE is immediate and appears in [59].

The transformation from IIL* to IMALL is efficient in another stronger manner. It preserves the structure of IIL* proofs. The IMALL translation of an IIL* proof is linear in the size of the given IIL* proof. Note that our transformation from IIL to IIL* does not necessarily preserve the structure of cut-free proofs in IIL due to the permutations that are needed to make copying redundant. Neither of our transformations preserves the structure of proofs with cut.

completed by one application of $\top$R. Whatever its form, one may mimick this entire proof in IIL* by an application of identity.

This completes the analysis in the case that the last proof rule applied is right tensor. In the case that the last rule applied is left implication, there are two possible forms this formula can take in any θ-translation: $k \multimap (((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-))$ and $k \multimap ((([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [C]^-))$.

The first possibility would imply that the assumed IMALL proof has the form given in Figure 6.17. The IMALL proof must take this almost form, because if any part of $[\Gamma]^-$ were to be included in the left premise, identity would not apply, and in fact there could be no proof of that branch, as stated in Proposition 6.5.72. Also, because there is no k at top level in the right premise of the $\multimap$L rule, Proposition 6.5.73 implies that reducing any formulas in $[\Gamma]^-$ could not lead to a proof. This reasoning applies four times, leaving one with the proof displayed below:

$$\dfrac{\dfrac{}{k \vdash k}\mathbf{I} \qquad \dfrac{\dfrac{\dfrac{\vdots \qquad \vdots}{\dfrac{\dfrac{[\mathbf{\Gamma_1}]^-, k \vdash [p_i]^+}{[\mathbf{\Gamma_1}]^- \vdash k \multimap [p_i]^+}\multimap\mathbf{R} \qquad \dfrac{[\mathbf{\Gamma_2}]^-, b \vdash [C]^+}{[\mathbf{\Gamma_2}]^-, k \otimes b \vdash [C]^+}\otimes\mathbf{L}}{[\Gamma]^-, (k \multimap [p_i]^+) \multimap (k \otimes b) \vdash [C]^+}\multimap\mathbf{L} \qquad \dfrac{\vdots}{\dfrac{[\Gamma]^-, k, [A]^- \vdash [C]^+}{[\Gamma]^-, k \otimes [A]^- \vdash [C]^+}\otimes\mathbf{L}}}{[\Gamma]^-, ((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-) \vdash [C]^+}\oplus\mathbf{L}}{[\Gamma]^-, k, k \multimap (((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-)) \vdash [C]^+}\multimap\mathbf{L}$$

Where $[\mathbf{\Gamma_1}]^- \bigcup [\mathbf{\Gamma_2}]^- = [\Gamma]^-$. From this proof, by induction, we can generate a IIL* proof of $\Gamma_1 \vdash p_i$. By Proposition 6.3.65 we can generate a IIL* proof of $\Gamma \vdash p_i$. Then, using **Left** $\supset$ **1** with the proof of $\Gamma \vdash p_i$ and the translation (by induction) of the rightmost branch from the above proof figure, we can construct an IIL* proof of $\Gamma \vdash C$. The middle unfinished branch of the above figure is irrelevant to the translation, but happens to always be provable by Lemma 6.4.69.

The second possibility would imply that the assumed IMALL proof has almost the same form as that in Figure 6.18. The IMALL proof must take this form, because if any part of $[\Gamma]^-$ were to be included in the left premise, identity would not apply, and in fact there could be no proof of that branch, as stated in Proposition 6.5.72. Also, because there is no k at top level in the right premise of the $\multimap$L rule, Proposition 6.5.73 implies that reducing any formulas in $[\Gamma]^-$ could not lead to a proof. This

that the last proof rule applied must be either $-\circ\mathbf{L}$, $\otimes\mathbf{R}$, or identity. However, even identity cannot apply, because k always appears on the left in any θ-translation, and k never appears at top level on the right in such a translation. Thus there are only two cases to consider, left implication, and right tensor.

First, let us consider the case when $\otimes\mathbf{R}$ is the last rule applied in a proof. There are two possible forms this formula can take in any θ-tranlsation: $k \otimes ([A]^- -\circ (k -\circ [B]^+))$, or $k \otimes ((p_i \oplus b) \otimes \top)$.

The first possibility would imply that the assumed IMALL proof has the form given in Figure 6.16. The IMALL proof must take this form, because if any part of $[\Gamma]^-$ were to be included in the left premise, identity would not apply, and in fact there could be no proof of that branch, as stated in Proposition 6.5.72. Also, because there is no k at top level in the right premise of the $\otimes$R rule, Proposition 6.5.73 implies that reducing any formulas in $[\Gamma]^-$ could not lead to a proof. This reasoning applies twice, leaving one with the proof displayed in Figure 6.16. This proof may be mimicked in IIL*as simply the application of **Right** $\supset$, and the hypothesis, which is itself a translation, may be mimicked by induction.

The second possibility would imply that the assumed IMALL proof has the form:

$$\dfrac{\overset{\overset{\vdots}{[\Delta]^- \vdash p_i \oplus b}}{\overset{\overset{\vdots}{[\Sigma]^- \vdash \top}}{[\Gamma]^- \vdash (p_i \oplus b) \otimes \top}\otimes R} \qquad \dfrac{}{k \vdash k}\mathbf{I}}{[\Gamma]^-, k \vdash k \otimes ((p_i \oplus b) \otimes \top)}\otimes R$$

The IMALL proof must take this form, because if any part of $[\Gamma]^-$ were to be included in the left premise, identity would not apply, and in fact there could be no proof of that branch, as stated in Proposition 6.5.72. Also, because there is no k at top level in the right premise of the $\otimes$R rule, Proposition 6.5.73 implies that reducing any formulas in $[\Gamma]^-$ could not lead to a proof. Thus for some Δ and Σ which together make up Γ, one has the above proof. Investigating the left unfinished branch, one sees by Proposition 6.5.73 that $p_i \oplus b$ must be reduced. Furthermore, it can be seen that this $p_i \oplus b$ must be reduced to p_i . Proposition 6.5.72 implies that $[\Delta]^- \equiv p_i$, and thus $[\Gamma]^- \equiv [\Sigma]^-, [p_i]^-$. On the other hand, the right unfinished branch could be

$A \oplus p$, or $A \multimap p$ for some formula A. If they are identically p, then the conclusion contains the formula $A \oplus p$ or $p \oplus A$ for some A, and the result follows. Otherwise, because all subformulas present in the premises of $\oplus$L are also subformulas of the conclusion, the result follows.

If &L is the last rule applied, the induction hypothesis implies that the context of the premise is identically p, or contains a subformula of the form $p \& A$, $A \& p$, $p \oplus A$, $A \oplus p$, or $A \multimap p$ for some formula A. If it is identically p, then the conclusion contains the formula $A \& p$ or $p \& A$ for some A, and the result follows. Otherwise, because all subformulas present in the premise of &L are also subformulas of the conclusion, the result follows. ∎

Proposition 6.5.73 *If formula F is a proper subformula of an encoding $[\,]^-$ or $[\,]^+$, respectively, and is not identically k, then F must be reduced below any other formula in any* IMALL *proof of $[\Gamma]^-, F \vdash [C]^+$ or $[\Gamma]^- \vdash F$, respectively.*

Proof. The proof of this property is almost immediate from property 6.5.72, since our encoding functions $[\,]^-$ and $[\,]^+$ have the requisite properties. ∎

Lemma 6.5.74 *If there is a proof of $\theta(\Gamma \vdash C)$ in* IMALL*, then there is a proof of $\Gamma \vdash C$ in* IIL*.

Proof. In order to prove Lemma 6.5.74, we perform cut-elimination on the given IMALL proof, and then observe that the resulting proof must be of a very special form. In fact, an IIL* proof can be directly read from any such proof. The action of the "locks and keys" encoded by the positive and negative occurrences of k in the IMALL translations forces any cut-free IMALL proof of a sequent to have a very specific form. Proposition 6.5.73 states this formally. It is exactly this sort of control over the shape of a proof which one can encode in linear logic sequents, but which is impossible to encode in intuitionistic and classical logic. The proof of this lemma proceeds by induction on the size of cut-free IMALL proof.

Given a cut-free IMALL proof of a sequent $\theta(\Gamma \vdash C)$, one considers which IMALL proof rule was applied last. Because the proof is cut-free, the last rule cannot be cut. Investigating the forms of IMALL formulas that can appear in a θ-translation, one sees

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma, (B \supset C) \vdash (A \supset B) \qquad \Gamma, C \vdash D}\,\mathbf{L}\supset \mathbf{2}}{\Gamma, ((A \supset B) \supset C) \vdash D}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{[\Gamma]^-, [(B \supset C)]^-, k \vdash [(A \supset B)]^+}}{[\Gamma]^-, [(B \supset C)]^- \vdash k \multimap [(A \supset B)]^+}\multimap\mathbf{R} \qquad \dfrac{\vdots}{k, b \vdash [D]^+}}{[\Gamma]^- \vdash ([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+))}\multimap\mathbf{R} \qquad \dfrac{\vdots}{k \otimes b \vdash [D]^+}\otimes\mathbf{L}}{[\Gamma]^-, ([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b) \vdash [D]^+}\multimap\mathbf{L} \qquad \dfrac{\vdots}{[\Gamma]^-, k, [C]^- \vdash [D]^+}\otimes\mathbf{L}}{[\Gamma]^-, k \otimes [C]^- \vdash [D]^+}\oplus\mathbf{L}}{[\Gamma]^-, (([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [C]^-) \vdash [D]^+}\multimap\mathbf{L}$$

$$\dfrac{k \vdash k\,\mathbf{I} \qquad [\Gamma]^-, (([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [C]^-) \vdash [D]^+}{[\Gamma]^-, k, k \multimap ((([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [C]^-)) \vdash [D]^+}\multimap\mathbf{L}$$

Figure 6.18: Case **Left** $\supset$ **2**.

If identity is the last rule applied, then $\Delta \equiv p$.

$\otimes$R, $\multimap$R, $\oplus$R, and &R do not apply because the right hand side is an atomic propositional literal.

The $\bot$R, $\bot$L, and $\top$R rules do not apply because the right hand side of the sequent is an atomic propositional literal.

The 1L and 1R rules do not apply since Δ does not contain 1 as a subformula.

If $\otimes$L is the last rule applied, Δ cannot be p, so the induction hypothesis implies that the premise contains a subformula of the form $p \& A$, $A \& p$, $p \oplus A$, $A \oplus p$, or $A \multimap p$ for some formula A. Because all subformulas present in the premise of $\otimes$L are also subformulas of the conclusion, the result follows.

If $\multimap$L is the last rule applied, the induction hypothesis implies that the context of the right hand premise is identically p, or contains a subformula of the form $p \& A$, $A \& p$, $p \oplus A$, $A \oplus p$, or $A \multimap p$ for some formula A. If it is identically p, then the conclusion contains the formula $A \multimap p$ for some A, and the result follows. Otherwise, because all subformulas present in the premises of $\multimap$L are also subformulas of the conclusion, the result follows.

If $\oplus$L is the last rule applied, the induction hypothesis implies that the contexts of both premises are identically p, or contain a subformula of the form $p \& A$, $A \& p$, $p \oplus A$,

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma \vdash p_i \qquad \Gamma, A \vdash C}\,\mathbf{L}\supset\mathbf{1}}{\Gamma, (p_i \supset A) \vdash C}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{[\Gamma]^-, k \vdash [p_i]^+}}{[\Gamma]^- \vdash k \multimap [p_i]^+}\multimap\mathbf{R} \quad \dfrac{\dfrac{\vdots}{k, b \vdash [C]^+}}{k \otimes b \vdash [C]^+}\otimes\mathbf{L}}{[\Gamma]^-, (k \multimap [p_i]^+) \multimap (k \otimes b) \vdash [C]^+}\multimap\mathbf{L} \quad \dfrac{\dfrac{\vdots}{[\Gamma]^-, k, [A]^- \vdash [C]^+}}{[\Gamma]^-, k \otimes [A]^- \vdash [C]^+}\otimes\mathbf{L}}{[\Gamma]^-, ((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-) \vdash [C]^+}\oplus\mathbf{L}}{[\Gamma]^-, k, k \multimap (((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-)) \vdash [C]^+}\multimap\mathbf{L}$$

$$\dfrac{}{k \vdash k}\mathbf{I}$$

Figure 6.17: Case **Left** $\supset$ **1**.

implies that $k, b \vdash [D]^+$ is provable, and by induction hypothesis there exists IMALL proofs of the other two branches. ■

We now introduce two propositions that simplify the other direction of Theorem 6.1.55. These propositions are mild alterations of lemmas used to establish the PSPACE-completeness of IMALL [59]. The first proposition is only used to prove the second, and the second proposition formally states that in a cut-free IMALL proof, no lock can be opened before there is a key available at the top level.

Proposition 6.5.72 *For any atomic proposition p, and multiset Δ not containing the constant 1 or the constant 0, if the sequent $\Delta \vdash p$ is provable in* IMALL*, then Δ is identically p, or contains a positive subformula of the form $p\&A$, $A\&p$, $p \oplus A$, $A \oplus p$, or $A \multimap p$ for some formula A.*

Note that the clause about the constant 0 is not actually needed in our formulation of IMALL. However, this property could be of interest outside the scope of this paper, and thus we state it exactly for full intuitionistic two-sided multiplicative additive linear logic.

Proof. The argument is by induction on the size of the cut-free IMALL proof. For each inductive step, one considers case analysis on the rules of IMALL.

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash (A \supset B)}\mathbf{R}\supset \quad \Rightarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ [\Gamma]^-, [A]^-, k \vdash [B]^+\end{array}}{[\Gamma]^-, [A]^- \vdash k \multimap [B]^+}\multimap\mathbf{R}}{[\Gamma]^- \vdash [A]^- \multimap (k \multimap [B]^+)}\multimap\mathbf{R} \qquad \dfrac{}{k \vdash k}\mathbf{I}}{[\Gamma]^-, k \vdash k \otimes ([A]^- \multimap (k \multimap [B]^+))}\otimes\mathbf{R}$$

Figure 6.16: Case **Right** $\supset$.

6.5 Completeness of Translation

In order to prove Theorem 6.1.55, we have to show that the translation is correct and faithful, *i.e.,* there exists a cut-free proof of $\Gamma \vdash C$ in IIL* if and only if there is a cut-free proof of $\theta(\Gamma \vdash C)$ in IMALL. This will be established in two lemmas below.

Lemma 6.5.71 *If there is a cut-free proof of $\Sigma \vdash C$ in* IIL**, then there is a cut-free proof of $\theta(\Sigma \vdash C)$ in* IMALL.

Proof. One proceeds by induction on the depth of proof in IIL*.

In the case that the proof of $\Sigma \vdash C$ is simply one application of identity, C is actually a proposition p_i (identity is only applicable to atomic propositions in IIL*), and therefore Γ must contain p_i as an element. Thus one can use Lemma 6.4.70.

In the case that the proof of $\Gamma \vdash C$ ends in an application of the Right $\supset$ rule of IIL*, then one may simply unlock the conclusion formula and then apply $\multimap R$ to the IMALL translation. Note that by definition, the translation of $[A \supset B]^+$ is $k \otimes ([A]^- \multimap (k \multimap [B]^+))$. This case is given in Figure 6.16, where the required IMALL proof of $[\Gamma]^-, [A]^-, k \vdash [B]^+$ is given by the induction hypothesis.

Suppose that the IIL* proof of $\Sigma \vdash C$ ends in an application of **Left** $\supset$ **1**. Then $\Sigma = \Gamma, (p_i \supset A)$. Consider the proof given in Figure 6.17. Lemma 6.4.69 implies that $k, b \vdash [C]^+$ is provable, and by induction hypothesis there exists IMALL proofs of the other two branches.

In the final case, suppose that the proof of $\Gamma \vdash C$ ends in an application of **Left** $\supset$ **2**. Consider the proof given in Figure 6.18. As in the previous case, Lemma 6.4.69

$$\dfrac{\dfrac{}{k \vdash k}\mathbf{I} \qquad \dfrac{\dfrac{\dfrac{}{b \vdash b}\mathbf{I}}{b \vdash p_i \oplus b}\oplus R \quad \dfrac{}{[\Gamma]^- \vdash \top}\top R}{[\Gamma]^-, b \vdash (p_i \oplus b) \otimes \top}\otimes R}{[\Gamma]^-, k, b \vdash k \otimes ((p_i \oplus b) \otimes \top)}\otimes R$$

Figure 6.13: Case 1 of Lemma 6.4.69.

$$\dfrac{\dfrac{}{k \vdash k}\mathbf{I} \qquad \dfrac{\dfrac{\dfrac{\vdots}{[\Gamma]^-, [A]^-, k, b \vdash [B]^+}}{[\Gamma]^-, [A]^-, b \vdash k \multimap [B]^+}\multimap\mathbf{R}}{[\Gamma]^-, b \vdash [A]^- \multimap (k \multimap [B]^+)}\multimap\mathbf{R}}{[\Gamma]^-, k, b \vdash k \otimes ([A]^- \multimap (k \multimap [B]^+))}\otimes\mathbf{R}$$

Figure 6.14: Case 2 of Lemma 6.4.69.

In the case that $C = (A \supset B)$ is an implication, we know that B is of smaller depth than C, and we can construct the proof as in Figure 6.14. ∎

Lemma 6.4.70 *For any* IIL* *multiset* Γ *and proposition* p_i*, the sequent* $[\Gamma]^-, [p_i]^-, k \vdash [p_i]^+$ *is provable in* IMALL.

Proof. The proof follows from expanding the definition of $[p_i]^+$, as seen in Figure 6.15.

∎

$$\dfrac{\dfrac{}{k \vdash k}\mathbf{I} \qquad \dfrac{\dfrac{\dfrac{}{p_i \vdash p_i}\mathbf{I}}{p_i \vdash p_i \oplus b}\oplus R \quad \dfrac{}{[\Gamma]^- \vdash \top}\top R}{[\Gamma]^-, p_i \vdash (p_i \oplus b) \otimes \top}\otimes R}{[\Gamma]^-, p_i, k \vdash k \otimes ((p_i \oplus b) \otimes \top)}\otimes R$$

Figure 6.15: Proof of Lemma 6.4.70.

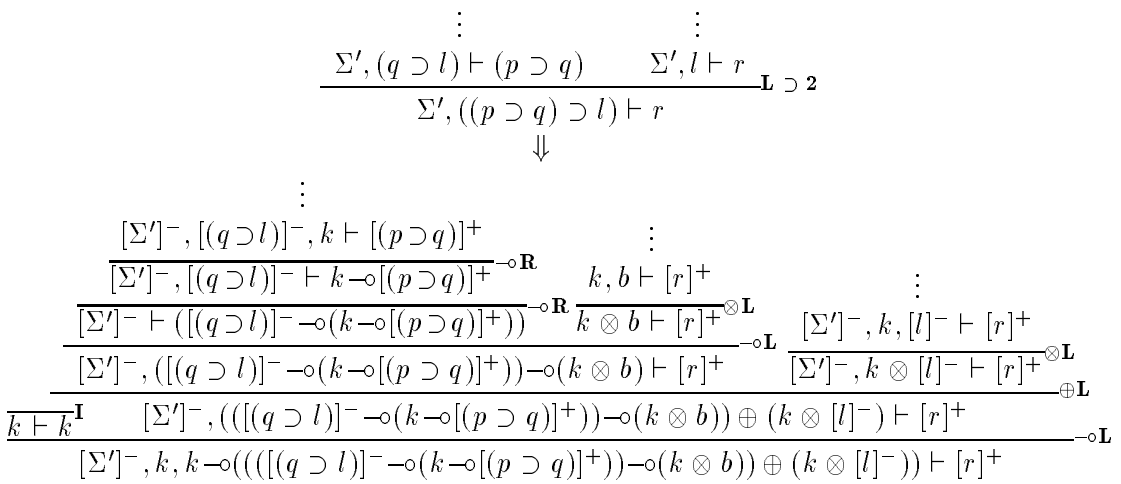

Figure 6.12: IIL* and IMALL proofs of example.

is a choice to be made in the way we split the context Σ' among the branches of the proof. However, because of the form of our translation, we can without loss of generality choose to keep the entire context on the left branch. Lemma 6.4.69 implies that $k, b \vdash [r]^+$, the upper right branch, is provable. Notice how $[r]^+$ has been devised to ensure this. And finally, we see that after two applications of $\mathbf{R} \supset$ we are left with the translation of the right hand branch of the IIL* proof.

In fact, the encoding is such that there are essentially no choices to be made in the proof of the IMALL translation that cannot be made in the proof of an IIL* formula. For example, once a formula is unlocked with the "key" k, no other formula may be unlocked until the unlocked formula is reduced completely, at which point it provides another key k. This method of "locks and keys" was introduced in [59]. In the next section we show that an IIL* formula is provable in IIL* if and only if its translation is provable in IMALL.

Lemma 6.4.69 *For any* IIL* *multiset* Γ *and formula* C*, the sequent* $[\Gamma]^-, k, b \vdash [C]^+$ *is provable in* IMALL.

Proof. The proof is by induction on the right-hand depth of C. If $C = p_i$ is a proposition, we can construct an IMALL proof as in Figure 6.13.

$$
\begin{array}{rcl}
[A \supset B]^+ & \triangleq & k \otimes ([A]^- \multimap (k \multimap [B]^+)) \\
[p_i]^+ & \triangleq & k \otimes ((p_i \oplus b) \otimes \top) \\
 & & \\
[p_i]^- & \triangleq & p_i \\
[p_i \supset A]^- & \triangleq & k \multimap (((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-)) \\
[(A \supset B) \supset C]^- & \triangleq & k \multimap ((([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [C]^-))
\end{array}
$$

Figure 6.11: Definition of translation

definitions of $[\,]^+$ and $[\,]^-$ given in Figure 6.11 can be seen to be well defined by induction on the size of the formulas.

For any IIL* sequent $\Gamma \vdash C$ we define

$$\theta(\Gamma \vdash C) \triangleq [\Gamma]^-, k \vdash [C]^+$$

Here $[\Gamma]^-$ stands for the result of the application of $[\,]^-$ to each element of Γ. Note that the "key" k is present in the context of the encoding of a sequent. We have chosen the notations $[\,]^+$ and $[\,]^-$ to suggest the interpretation of positive and negative polarity of occurrences.

Let us first demonstrate how parts of the example IIL* proof given in Figure 6.3 are translated into IMALL. Consider the sequent $\Sigma', (p \supset q) \supset l \vdash r$, where Σ' abbreviates $l \supset r, (q \supset r) \supset q$. This sequent has the θ-translation $[\Sigma']^-, [(p \supset q) \supset l]^-, k \vdash [r]^+$. By the above definition, $[(p \supset q) \supset l]^- = k \multimap ((([(q \supset l)]^- \multimap (k \multimap [(p \supset q)]^+)) \multimap (k \otimes b)) \oplus (k \otimes [l]^-))$. In the example IIL* proof given in Figure 6.3, the proof of this sequent ends in an application of the $\mathbf{L} \supset \mathbf{2}$ rule.

The intuitive structure of the proof in Figure 6.12 is as follows. The leftmost application of $\mathbf{I}$ and the bottommost application of $\multimap\mathbf{L}$ correspond to "unlocking" the formula of interest. The unlocked formula corresponding to $(p \supset q) \supset l$ has $\oplus$ as its main connective. The proof tree therefore forks, and after a simple application of $\otimes\mathbf{L}$, the rightmost branch can be seen to be the translation of the rightmost branch of the IIL* proof.

The left main branch of the proof progresses by applying the $\multimap\mathbf{L}$ rule. Here there

$$\mathbf{I}\quad \frac{}{p \vdash p} \qquad \frac{\Sigma \vdash A \qquad A, \Gamma \vdash \Delta}{\Sigma, \Gamma \vdash \Delta}\ \mathbf{Cut}$$

$$\otimes\mathbf{L}\quad \frac{\Sigma, A, B \vdash \Delta}{\Sigma, (A \otimes B) \vdash \Delta} \qquad \frac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}\ \otimes\mathbf{R}$$

$$\multimap\mathbf{L}\quad \frac{\Sigma \vdash A \qquad \Gamma, B \vdash \Delta}{\Sigma, \Gamma, (A \multimap B) \vdash \Delta} \qquad \frac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}\ \multimap\mathbf{R}$$

$$\oplus\mathbf{L}\quad \frac{\Sigma, A \vdash \Delta \qquad \Sigma, B \vdash \Delta}{\Sigma, (A \oplus B) \vdash \Delta} \qquad \frac{\Sigma \vdash A \qquad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}\ \mathbf{\&\,R}$$

$$\mathbf{\&\,L1}\quad \frac{\Sigma, A \vdash \Delta}{\Sigma, (A \,\&\, B) \vdash \Delta} \qquad \frac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}\ \oplus\mathbf{R1}$$

$$\mathbf{\&\,L2}\quad \frac{\Sigma, B \vdash \Delta}{\Sigma, (A \,\&\, B) \vdash \Delta} \qquad \frac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}\ \oplus\mathbf{R2}$$

$$\bot\mathbf{L}\quad \frac{}{\bot \vdash} \qquad \frac{\Sigma \vdash}{\Sigma \vdash \bot}\ \bot\mathbf{R}$$

$$1\mathbf{L}\quad \frac{\Sigma \vdash \Delta}{\Sigma, 1 \vdash \Delta} \qquad \frac{}{\vdash 1}\ 1\mathbf{R}$$

$$\frac{}{\Sigma \vdash \top}\ \top\mathbf{R}$$

Figure 6.10: Rules for IMALL

6.4 IIL* to IMALL

An intuitionistic linear logic sequent is composed of two multisets of linear logic formulas separated by a $\vdash$, where there is no more than one formula in the consequent (*i.e.,* right-hand side) multiset. We assume a set of propositional atoms p_i to be given. Figure 6.10 gives the inference rules for the intuitionistic linear sequent calculus, with the slight restriction that the 0 rule is omitted.[2] This omission does not pose problems for cut elimination.

We now define a pair of mutually recursive translation functions that transform any IIL* formula into an IMALL formula. k and b are fresh propositional letters. The

[2]Our arguments also apply to the sequent calculus given on p. 53 of [35] without the 0 rule.

$$
\begin{aligned}
& weight(\Sigma, (D \supset E) \supset F \vdash C) - \ weight(\Sigma, E \supset F, D \vdash E) \\
&= m((D \supset E) \supset F, 1) + m(C, 1) - m(E \supset F, 1) - m(D, 1) - m(E, 1) \\
&= m(D, 3) + 2m(E, 2) + 2 + m(F, 1) + 1 + \\
&\quad m(C, 1) - m(E, 2) - m(F, 1) - 1 - m(D, 1) - m(E, 1) \\
&> 0
\end{aligned}
$$

Figure 6.9: Example calculation of weight

is then $\Sigma, p \supset B, B \vdash C$. By Proposition 6.2.63, the sequent $\Sigma, B \vdash C$ must also have an IIL proof and since it is of smaller weight than $\Sigma, p \supset B \vdash C$, the induction hypothesis can be applied to it yielding an IIL* proof of $\Sigma, B \vdash C$. The required IIL* proof of $\Sigma, p \supset B \vdash C$ can be constructed using the $\mathbf{L} \supset \mathbf{1}$ rule with the premises $\Sigma, B \vdash C$ and $\Sigma \vdash p$.

If the final inference in the given IIL proof is $\mathbf{L} \supset$ applied to a principal formula of the form $(D \supset E) \supset F$, then Γ has the form $\Sigma, (D \supset E) \supset F$, and we have IIL proofs for the two premises $\Sigma, (D \supset E) \supset F \vdash D \supset E$ and $\Sigma, (D \supset E) \supset F, F \vdash C$. Proposition 6.2.63 applied to the second premise yields an IIL proof of $\Sigma, F \vdash C$ to which the induction hypothesis can be applied yielding an IIL* proof of $\Sigma, F \vdash C$. Since in IIL we can prove $D, (E \supset F) \vdash (D \supset E) \supset F$ and $D, (D \supset E) \vdash E$, we can use the cut rule twice with the sequent $\Sigma, (D \supset E) \supset F \vdash D \supset E$ to get an IIL proof of $\Sigma, E \supset F, D \vdash E$. The difference in weight between this last sequent and the original conclusion sequent $\Sigma, (D \supset E) \supset F \vdash C$ is given in Figure 6.9.

So the induction hypothesis yields an IIL* proof of $\Sigma, E \supset F, D \vdash E$ which by $\mathbf{R} \supset$ yields an IIL* proof of $\Sigma, E \supset F \vdash D \supset E$. This last sequent with $\Sigma, F \vdash C$ yield an IIL* proof of $\Sigma, (D \supset E) \supset F \vdash C$ by the $\mathbf{L} \supset \mathbf{2}$ rule of IIL*. ∎

The lack of contraction in IIL* makes this formulation of the sequent rules for implicational intuitionistic propositional logic amenable to encoding into IMALL.

$$\begin{aligned} weight(A_1, \ldots, A_n \vdash C) &= m(A_1, 1) + \ldots + m(A_n, 1) + m(C, 1) \\ m(A \supset B, d) &= m(A, d+1) + d * (m(B, d) + 1) \\ m(p, d) &= d \end{aligned}$$

Figure 6.8: Definition of $weight$

the same size as Θ_2. The backward inference with the subproofs Θ_{12} and Θ_2' is smaller than Π and we can therefore employ the induction hypothesis to eliminate the backward inference from it. The resulting proof is therefore free of backward inferences and has size no larger than Π.

The other possibility is that the principal formula is of the form $q \supset B$ where q occurs in Γ. In this case the inferences permute similarly, and the resulting proof may be seen to be forward by induction, and the fact that q occurs in Γ. ∎

Lemma 6.3.68 *Given a proof of* $\Gamma \vdash C$ *in* IIL*, a proof of* $\Gamma \vdash C$ *can be constructed in* IIL*.

Proof. By Lemma 6.3.67, we can restrict our attention to forward proofs. We proceed by induction, not on the size of the given proof, but on $weight(\sigma)$ for a sequent σ, as defined in Figure 6.8. There are four cases according to the final inference in the given proof.

It is easy to show by induction on the structure of A that if $0 < c < d$, then $0 < m(A, c) < m(A, d)$.

If the given IIL proof of $\Gamma \vdash C$ is an axiom, then the proof is also an IIL* proof.

If the final inference in the given forward IIL proof is $\mathbf{R} \supset$ applied to a conclusion of the form $\Gamma \vdash A \supset B$ to generate the premise $\Gamma, A \vdash B$, then this premise is of smaller weight. We can therefore apply the induction hypothesis to the premise to get an IIL* proof of $\Gamma, A \vdash B$ from which the IIL* proof of $\Gamma \vdash A \supset B$ can be completed by the $\mathbf{R} \supset$ rule of IIL*.

If the final inference in the given forward IIL rule is $\mathbf{L} \supset$ applied to a principal formula of the form $p \supset B$, then Γ has the form $\Sigma, p \supset B$ and p must occur in Σ. Since p occurs in Σ, the sequent $\Sigma \vdash p$ is an IIL* axiom. The nontrivial premise

$$\dfrac{\dfrac{\begin{array}{c}\Theta_{11}\\ \vdots\\ \Gamma, p \supset A \vdash D \supset E\end{array} \qquad \dfrac{\begin{array}{c}\Theta_{12}\\ \vdots\\ \Gamma, p \supset A, F \vdash p\end{array}}{\Gamma, p \supset A \vdash p}\,\mathbf{L}\supset \qquad \dfrac{\begin{array}{c}\Theta_2\\ \vdots\\ \Gamma, p \supset A, A \vdash C\end{array}}{\Gamma, p \supset A \vdash C}\,\mathbf{L}\supset}{\Gamma, p \supset A \vdash C}$$

becomes

$$\dfrac{\begin{array}{c}\Theta_{11}\\ \vdots\\ \Gamma, p \supset A \vdash D \supset E\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\Theta_{12}\\ \vdots\\ \Gamma, p \supset A, F \vdash p\end{array} \qquad \dfrac{\begin{array}{c}\Theta'_2\\ \vdots\\ \Gamma, p \supset A, F, A \vdash C\end{array}}{\Gamma, p \supset A, F \vdash C}\,\mathbf{L}\supset}{\Gamma, p \supset A \vdash C}\,\mathbf{L}\supset}$$

Figure 6.7: Permuting backward inferences

If the final inference in Π is not a backward inference, then we have the result immediately by induction.

If the final step is a backward inference in Π, then we use the induction hypothesis to eliminate the backward inferences in the subproofs of the premises. This transforms the proof Π to the form below, where the only backward inference is the final one.

$$\dfrac{\dfrac{\begin{array}{c}\Theta_1\\ \vdots\\ \Gamma, p \supset A \vdash p\end{array} \qquad \dfrac{\begin{array}{c}\Theta_2\\ \vdots\\ \Gamma, p \supset A, A \vdash C\end{array}}{\Gamma, p \supset A \vdash C}\,\mathbf{L}\supset}{\Gamma, p \supset A \vdash C}$$

The premise $\Gamma, p \supset A \vdash p$ cannot be an axiom since p does not occur in Γ. The final inference in the proof Θ_1 of $\Gamma, p \supset A \vdash p$ must therefore be an $\mathbf{L}\supset$ inference whose principal formula is either of the form $(D \supset E) \supset F$ or of the form $q \supset B$ where q occurs in Γ. In either case, these inferences can be permuted below the final inference in Π, as in Figure 6.7.

In Figure 6.7, the proof Θ'_2 is obtained from Θ_2 by Proposition 6.2.62 but has

$$\frac{}{\Gamma, p_i \vdash p_i}\mathbf{I}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash (A \supset B)}\mathbf{R}\supset$$

$$\frac{\Gamma \vdash p_i \qquad \Gamma, B \vdash C}{\Gamma, (p_i \supset B) \vdash C}\mathbf{L}\supset \mathbf{1}$$

$$\frac{\Gamma, (B \supset C) \vdash (A \supset B) \qquad \Gamma, C \vdash D}{\Gamma, ((A \supset B) \supset C) \vdash D}\mathbf{L}\supset \mathbf{2}$$

Figure 6.6: Rules for IIL*

Lemma 6.3.66 *Given a proof of* $\Gamma \vdash A$ *in* IIL*, *a proof of* $\Gamma \vdash A$ *can be constructed in* IIL.

Proof. By induction on IIL* proofs using Propositions 6.3.65 and 6.2.64. ∎

The other direction of the equivalence of IIL and IIL* is somewhat more complicated. Our original argument involved *depth reduction* (see Section 6.6.1). Here we adapt an argument due to Dyckhoff [26] by introducing Lemma 6.3.67 and modifying the definition of the weight of a sequent used to justify the induction in Lemma 6.3.68.

Consider an $\mathbf{L}\supset$ inference in an IIL proof with a principal antecedent formula of the form $p \supset A$. Let $\Gamma \vdash C$ be the conclusion sequent of the inference. The inference is said to be *backward* if p does not occur in Γ. A *forward* proof is one with no backward inferences. These names are chosen to be reminiscent of forward and backward chaining.

Lemma 6.3.67 *Any cut-free, contraction-free* IIL *proof* Π *of size* n *can be transformed to a cut-free, contraction-free forward proof* Θ *of size no more than* n *with the same conclusion as* Π.

Proof. The proof is by induction on the size of the cut-free, contraction-free proof Π.

Now applying Proposition 6.2.58 to the proof of the right hand hypothesis, we are able to obtain a proof of $\Gamma, A \supset B, B \vdash C$ which is smaller than the original proof, and thus by induction we may assume that $A \supset B$ may be eliminated from the context, and we have a proof of $\Gamma, B \vdash C$ of size no more than n. ∎

Proposition 6.2.64 *For all* IIL *formulas* A, B, C *the sequent* $(A \supset B) \supset C \vdash B \supset C$ *is provable in* IIL.

Proof. By the following IIL proof:

$$\dfrac{\dfrac{\dfrac{}{(A \supset B) \supset C, B, A \vdash B}\mathbf{I}}{(A \supset B) \supset C, B \vdash (A \supset B)}\mathbf{R}\supset \qquad \dfrac{}{(A \supset B) \supset C, B, C \vdash C}\mathbf{I}}{(A \supset B) \supset C, B \vdash C}\mathbf{L}\supset$$
$$\dfrac{(A \supset B) \supset C, B \vdash C}{(A \supset B) \supset C \vdash B \supset C}\mathbf{R}\supset$$

∎

6.3 IIL and IIL*

We now introduce an interesting optimization of IIL called IIL*, and prove that cut-free, contraction-free IIL proofs are easily transformed to proofs in IIL*. The proof rules for IIL* are given in Figure 6.6. Similar optimizations have been studied by others [90, 79, 44, 26].

Note that the identity rule is only applicable to atomic propositions, and that weakening is only allowed at the leaves of a proof, *i.e.*, at an application of identity. Most important, however, is the property that the principal formula is not duplicated in the premises of any of the rules in IIL*.

Proposition 6.3.65 *Given a proof of* $\Gamma \vdash B$ *of size* n *in* IIL*, *we can produce a proof of* $\Gamma, A \vdash B$ *of size* n *in* IIL*.

Proof. We simply add the IIL* formula A to each sequent in the entire IIL* derivation. That is, by case analysis on the rules in IIL*, we see that adding a formula A to the context (left hand side) of the hypotheses and conclusions of all the rules of IIL* leaves us with a correctly formed IIL* proof of $\Gamma, A \vdash B$. ∎

Proof. The argument here is a straightforward adaptation of the cut-elimination proof for G3 that appears in [51]. ∎

Proposition 6.2.61 *Any proof of* $\Gamma \vdash A$ *in* IIL *can be transformed into a proof of* $\Gamma \vdash A$ *in* IIL *that does not employ the contraction or cut rules.*

Proof. By application of Proposition 6.2.60 and then Proposition 6.2.59. Note that the latter lemma will never introduce cuts into a proof, and thus preserves the cut-free nature of proofs. ∎

Proposition 6.2.62 *Given a proof of* $\Gamma \vdash B$ *of size* n *in* IIL*, we can produce a proof of* $\Gamma, A \vdash B$ *of size* n *in* IIL*.*

Proof. We simply add the IIL formula A to each sequent in the entire IIL derivation. That is, by case analysis on the rules in IIL, we see that adding a formula A to the context (left hand side) of the hypotheses and conclusions of all the rules of IIL leaves us with a correctly formed IIL proof of $\Gamma, A \vdash B$. Note that the formula A that has been weakened in, never occurs as the principal formula of any rule in the resulting proof. ∎

Proposition 6.2.63 *Given a proof of* $\Gamma, A \supset B, B \vdash C$ *of size* n *in* IIL*, we can find a proof of* $\Gamma, B \vdash C$ *of size less than or equal to* n *in* IIL*.*

Proof. We prove this property by induction on the size of IIL proof. At each step we perform case analysis on the last rule applied.

If the last rule is identity, which is restricted to atomic propositions, we may safely remove the formula $A \supset B$ from the context.

If the last rule applied is $R\supset$, **Cut**, or **Contraction**, then by induction we have our result.

In the final case of $L\supset$, if some other implication in the context is analyzed, then by induction we have our result. If $A \supset B$ is the formula analyzed, then we know the derivation is of the form:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma, A \supset B, B \vdash A & \Gamma, A \supset B, B, B \vdash C\end{array}}{\Gamma, A \supset B, B \vdash C}\mathbf{L}\supset$$

then we may construct the following deduction:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Gamma, (A_1 \supset A_2), A_1 \vdash A_1 & \Gamma, (A_1 \supset A_2), A_1, A_2 \vdash A_2\end{array}}{\Gamma, (A_1 \supset A_2), A_1 \vdash A_2}\mathbf{L}\supset}{\Gamma, (A_1 \supset A_2) \vdash (A_1 \supset A_2)}\mathbf{R}\supset$$

The two remaining sequents in the above deduction are provable by induction. ∎

Proposition 6.2.57 *For any sequent $\Sigma \vdash A$ appearing in any* IIL *proof of $\Gamma \vdash B$, the multiset Γ is a sub-multiset of Σ.*

Proof. By induction on the size of proofs followed by a straightforward case analysis on the IIL rules. ∎

This conservation of antecedent formulas in IIL proofs provides the key to the elimination of contraction as shown by Propositions 6.2.58 and 6.2.59. The *size* of a proof is taken to be the number of inferences in it.

Proposition 6.2.58 *Given a proof of $\Gamma, A, A \vdash B$ of size n in* IIL*, we can produce a proof of $\Gamma, A \vdash B$ of size n in* IIL*.*

Proof. From Proposition 6.2.57, we know that Γ, A, A appears as a sub-multiset of each sequent in the given proof tree. By case analysis on the rules in IIL, we see that by replacing Γ, A, A with Γ, A everywhere in the derivation, we are left with a correctly formed IIL proof of $\Gamma, A \vdash B$. ∎

Proposition 6.2.59 *Given a proof of $\Gamma \vdash A$ of size n in* IIL*, we can construct a proof of $\Gamma \vdash A$ in* IIL *of size no greater than n that does not employ the contraction rule.*

Proof. By induction on proof size. If the last rule in a derivation is contraction, then we simply apply proposition 6.2.58 to its premise to achieve a smaller proof of the desired sequent. In other cases we appeal to the induction hypothesis. ∎

Proposition 6.2.60 *If there is a proof of $\Gamma \vdash A$ in* IIL *then there is a proof of $\Gamma \vdash A$ in* IIL *that does not employ the cut rule.*

$$\frac{}{\Gamma, p_i \vdash p_i}\mathbf{I}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash (A \supset B)}\mathbf{R}\supset$$

$$\frac{\Gamma, (A \supset B) \vdash A \qquad \Gamma, (A \supset B), B \vdash C}{\Gamma, (A \supset B) \vdash C}\mathbf{L}\supset$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\mathbf{Contraction}$$

$$\frac{\Gamma \vdash C \qquad \Gamma, C \vdash B}{\Gamma \vdash B}\mathbf{Cut}$$

Figure 6.5: Rules for IIL

From the logic programming perspective given in [73], the main result of this paper addresses the issue of replacing copying and reuse in intuitionistic proofs by sharing. We believe that our results, together with [41, 7], contribute to the understanding of the role of linear logic as an expressive and natural framework for describing the control structure of logic programs.

6.2 Properties of IIL

In this section, we present a series of lemmas about IIL that eventually establish the eliminability of cut and contraction, the admissibility of weakening, and the redundancy of copying in IIL proofs.

Proposition 6.2.56 shows that the rule of identity on atomic formulas can be extended to all formulas.

Proposition 6.2.56 *For all* IIL *formulas* A *and* IIL *sequents* Γ*, there exists a proof of* $\Gamma, A \vdash A$ *in* IIL*.*

Proof. We build the proof by induction on the structure of A. If A is an atomic proposition, the result is immediate. If A is an implication, that is $A = A_1 \supset A_2$,

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\Sigma', (q \supset l) \vdash (p \supset q) \qquad \Sigma', l \vdash r}\,\mathbf{L}\supset\mathbf{2}}{\Sigma', ((p \supset q) \supset l) \vdash r}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{[\Sigma']^-, [(q \supset l)]^- \vdash [(p \supset q)]^+}\multimap\mathbf{R} \qquad \vdots}{[\Sigma']^- \vdash [(q \supset l)]^- \multimap [(p \supset q)]^+} \qquad b \vdash [r]^+}{[\Sigma']^-, ([(q \supset l)]^- \multimap [(p \supset q)]^+) \multimap b \vdash [r]^+}\multimap\mathbf{L} \qquad \vdots \qquad [\Sigma']^-, [l]^- \vdash [r]^+}{[\Sigma']^-, (([(q \supset l)]^- \multimap [(p \supset q)]^+) \multimap b) \oplus [l]^- \vdash [r]^+}\oplus\mathbf{L}$$

Figure 6.4: Toward IMALL translation of example.

IIL* along with the restriction on weakening make it possible to embed IIL* in IMALL. This translation is asymmetric, *i.e.,* positive occurrences of formulas in sequents are treated differently from negative occurrences. The basic idea is that a left implication rule of IIL* is translated by a block in IMALL consisting of a $\multimap\mathbf{L}$ rule (which accounts for the principal formula) followed by a $\oplus\mathbf{L}$ rule (which accounts for the context). For instance, the translation $[(p \supset q) \supset l]^-$ will be basically $(([(q \supset l)]^- \multimap [(p \supset q)]^+) \multimap b) \oplus [l]^-$. If Σ' abbreviates $l \supset r, (q \supset r) \supset q$, the last step in the IIL* proof displayed in Figure 6.3 will be translated basically as in Figure 6.4, where the middle branch will be provable.

The actual translation is more complicated; it also involves the "locks-and-keys" technique from [59] in order to ensure faithfulness. We defer the discussion of details until Section 6.4.

In summary, we provide a transformation from IIL sequents to IMALL sequents by transforming IIL proofs. Our main result is:

Theorem 6.1.55 IIL *can be embedded into* IMALL. *The embedding preserves the structure of cut-free proofs in* IIL*.

IIL proofs are transformed by eliminating any use of the cut and contraction rules, permuting the order of the inferences, and modifying the $\mathbf{L}\supset$ rule so as to eliminate the need for copying. The resulting IIL* proofs can then be embedded in IMALL.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma, p, q, p \vdash q}\mathbf{I}}{\Sigma, p, q \vdash p \supset q}\mathbf{R}\supset \quad \dfrac{\dfrac{}{\Sigma, p, q, l \vdash l}\mathbf{I} \quad \dfrac{}{\Sigma, p, q, l, r \vdash r}\mathbf{I}}{\Sigma, p, q, l \vdash r}\mathbf{L}\supset}{\Sigma, p, q \vdash r}\mathbf{L}\supset}{\dfrac{\Sigma, p \vdash q \supset r}{\Sigma \vdash p \supset q}\mathbf{R}\supset \quad \dfrac{}{\Sigma, p, q \vdash q}\mathbf{I}}\mathbf{L}\supset \quad \dfrac{\dfrac{}{\Sigma, l \vdash l}\mathbf{I} \quad \dfrac{}{\Sigma, l, r \vdash r}\mathbf{I}}{\Sigma, l \vdash r}\mathbf{L}\supset}{\Sigma \vdash r}\mathbf{L}\supset$$

Figure 6.2: Modified Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A, B, p, q \vdash q}\mathbf{I} \quad \dfrac{\dfrac{}{B, p, q, l \vdash l}\mathbf{I} \quad \dfrac{}{B, p, q, l, r \vdash r}\mathbf{I}}{A, B, p, q, l \vdash r}\mathbf{L}\supset\mathbf{1}}{A, D, B, p, q \vdash r}\mathbf{L}\supset\mathbf{1}}{\dfrac{A, D, B, p \vdash q \supset r}{A, D, (q \supset r) \supset q, p \vdash q}\mathbf{R}\supset \quad \dfrac{}{A, D, p, q \vdash q}\mathbf{I}}\mathbf{L}\supset\mathbf{2}}{A, D, (q \supset r) \supset q \vdash p \supset q}\mathbf{R}\supset \quad \dfrac{\dfrac{}{C, l \vdash l}\mathbf{I} \quad \dfrac{}{C, l, r \vdash r}\mathbf{I}}{A, C, l \vdash r}\mathbf{L}\supset\mathbf{1}}{\Sigma \vdash r}\mathbf{L}\supset\mathbf{2}$$

Figure 6.3: "Linearized" Proof in IIL* where A is $l \supset r$, B is $r \supset q$, C is $(q \supset r) \supset q$, D is $q \supset l$.

The advantage of IIL* is that there is no copying of principal formulas.[1] An antecedent principal formula of the form $(A \supset B) \supset C$ is replaced by the simpler formula $B \supset C$ in one of the premises of the $\mathbf{L} \supset \mathbf{2}$ rule. Let A, B, C, and D label the formulas $l \supset r$, $r \supset q$, $(q \supset r) \supset q$, and $q \supset l$, respectively. With these new rules, the above proof can be transformed to an IIL* proof as in Figure 6.3.

The absence of contraction and the absence of copying of principal formulas in

[1] Grigori Mints directed our attention to IIL*. Observe that after depth-reduction (see Section 6) IIL* provides a direct proof-theoretic explanation for the membership in PSPACE of the decision problem for propositional intuitionistic logic. Cut-free proofs in IIL* have a height that is linear in the number of connectives in the conclusion sequent. An alternating Turing machine can therefore generate and check the proof of a given sequent in a nondeterministic manner within polynomial time.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma, p, q, p \vdash q}\mathbf{I}}{\Sigma, p, q \vdash p \supset q}\mathbf{R}\supset \quad \dfrac{}{\Sigma, p, q, l \vdash l}\mathbf{I}}{\Sigma, p, q \vdash l}\mathbf{L}\supset \quad \dfrac{}{\Sigma, p, q, r \vdash r}\mathbf{I}}{\Sigma, p, q \vdash r}\mathbf{L}\supset}{\Sigma, p \vdash q \supset r}\mathbf{R}\supset$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Sigma, p \vdash q \supset r} \quad \dfrac{}{\Sigma, p, q \vdash q}\mathbf{I}}{\Sigma, p \vdash q}\mathbf{L}\supset}{\Sigma \vdash p \supset q}\mathbf{R}\supset \quad \dfrac{}{\Sigma, l \vdash l}\mathbf{I}}{\Sigma \vdash l}\mathbf{L}\supset \quad \dfrac{}{\Sigma, r \vdash r}\mathbf{I}}{\Sigma \vdash r}\mathbf{L}\supset$$

Figure 6.1: Proof of $\Sigma \vdash r$ in IIL where Σ is $l \supset r, (p \supset q) \supset l, (q \supset r) \supset q$

One clear difficulty in translating that proof into IMALL is that the multiset Σ appears in every sequent in the proof. In IMALL, a formula can appear as the principal formula of at most one inference along any branch of the proof. In the above proof, the copying of the principal formula of an inference into the premises seems essential. The formulas $(p \supset q) \supset l$ and $l \supset r$ appear twice as principal formulas, and in both cases, these duplicate occurrences are along the same branch of the proof. We can deal with the duplicate use of $l \supset r$ by rearranging the above proof as in Figure 6.2.

The next step is to deal with the copying of the formula $(p \supset q) \supset l$. For this purpose, we modify the $\mathbf{L} \supset$ rule of IIL to the following two rules:

$$\dfrac{\Gamma \vdash p \quad \Gamma, B \vdash C}{\Gamma, (p \supset B) \vdash C}\mathbf{L}\supset\mathbf{1}$$

$$\dfrac{\Gamma, (B \supset C) \vdash (A \supset B) \quad \Gamma, C \vdash D}{\Gamma, ((A \supset B) \supset C) \vdash D}\mathbf{L}\supset\mathbf{2}$$

We also discard the cut and contraction rules and call the resulting system IIL*.

A *reduction* is the process of applying a rule to a sequent matching the conclusion of the rule in order to generate the corresponding premises. The *principal formula* of the rule is then said to be *reduced* by the reduction. The occurrence of an instance of a rule in a proof is said to be an *inference*. The proper subformulas of a principal formula of a rule that appear in the premises of the rule are called the *side formulas*. A proof is represented as a tree rooted at its conclusion sequent at the bottom and with the leaves at the top. Given this orientation, the notion of a rule occuring above or below another rule should be clear.

The main result of this paper is an efficient embedding of the implicational fragment of propositional intuitionistic logic (IIL) in the intuitionistic fragment of multiplicative-additive linear logic (IMALL). We provide a transformation of an IIL sequent σ to an IMALL sequent ρ so that IMALL proves ρ exactly when IIL proves σ. The sequents σ and ρ are then said to be *equiprovable*. The system IIL is given by a fairly standard sequent formulation of the intuitionistic implicational logic shown in Figure 6.5 in Section 2. These rules are similar to those of Kleene's G3 [51]. The target system, IMALL, is shown in Figure 6.10 in Section 4. Note that the rules for negation, par, and the constant 0 are absent. Because the presentation is in terms of two sided sequents, cut-elimination for IMALL holds despite these omissions. Cut-elimination is of course a crucial tool in many of our proofs.

The main distinction between IIL and IMALL is in their treatment of the structural rules. IIL has an explicit rule of contraction and the rule of weakening is implicitly built into the **I** rule. Furthermore, the principal formula of an $\mathbf{L}\supset$ rule is *copied* into the premise sequents of each IIL rule. IMALL, on the other hand, has neither contraction nor weakening, and expressly forbids the copying of the principal formula of any rule into a premise. What IMALL does allow is the sharing of the non-principal formulas between the two premises of an additive inference rule. The cut rule and the contraction rule of IIL can be shown to be eliminable. In order to further bridge the gap between these two systems, it is important to establish control over the use of structural rules in IIL proofs so that any copying of the principal formulas into the premises is made inessential. Consider the IIL proof of the sequent $\Sigma \vdash r$ in Figure 6.1, where Σ denotes $l \supset r, (p \supset q) \supset l, (q \supset r) \supset q$.

[41, 7]. Furthermore, our result addresses the issue of replacing copying and reuse by sharing as discussed below.

A first indication that copying and reuse of hypotheses in intuitionistic logic might be replaceable by sharing is the contraction-free formulation of intuitionistic logic, given by the system G3 in Section 80 of [51] (see also the formulation of a purely implicational fragment considered in Section E6, Chapter 5 of [23]). A similar formulation is given in Section 2 below. We concentrate on the purely implicational fragment, which suffices because of the reduction discussed in [86]. However, a central role in our approach is played by a further reformulation of intuitionistic logic suggested by the methods used in [90] and [79]. The corresponding calculus, presented in Section 3 below, is the actual source calculus for our translation of cut-free proofs into the "intuitionistic" version of multiplicative additive linear logic. Our translation is exponential in the implication depth, but polynomial on formulas of bounded implication depth. (In fact, it suffices to consider only implications of depth at most 2, see, *e.g.*, [76]. This depth reduction dates back to [94].) A preliminary version of this work was reported in [62].

6.1 Overview

We only consider propositional systems of intuitionistic and linear logic. We use the following notations that are common to both the intuitionistic and linear formalisms:

l_i, p_i, q_i, r_i	Propositional literals
A, B, C	Arbitrary formulas
Σ, Γ	Arbitrary multisets of formulas
$\Gamma \vdash \Sigma$	Sequent with *antecedent* Γ and *consequent* Σ

We entirely omit the linear negation operation of MALL. Note that a sequent is represented in terms of two multisets, not sets, of formulas. For the intuitionistic sequent calculi, the consequent multiset is either a singleton or is empty. When we speak of a formula in a sequent, we are really referring to an occurrence of the formula.

corollary of the negative interpretation of classical logic in intuitionistic logic and the undecidability of classical first-order logic, both of which may be found, *e.g.,* in [51]). Therefore it is impossible to have a desired embedding for first-order quantifiers.

Another, more subtle obstruction to obtaining a very "logical" embedding is the discrepancy in complexity on the level of cut-elimination (proof normalization). Already for the purely implicational fragment of propositional intuitionistic logic, cut-elimination is hyperexponential (the equivalent fact about normalization in the simple typed lambda calculus is usually one of the first exercises in a graduate course in the subject). In contrast, cut-elimination for MALL is known to be much lower, at most exponential. In fact, this is true not just in the propositional case, but also for first-order and for second-order MALL. The required bounds are given by the Small Normalization Theorem in [30]. Hence a translation that preserves normalization of proofs with cut would have to be hyperexponential. (However, it may be possible to use an optimized presentation of intuitionistic logic such as [44] to give a triple exponential translation that preserves normalization of optimized proofs with cut.)

These results do leave open the possibility of an efficient syntactic translation of propositional intuitionistic logic into MALL so that such a translation does preserve *cut-free* proofs of a certain optimized form. In this chapter we construct such a translation. Our translation is an instance of what Girard terms an "asymmetrical interpretation," that is, positive occurrences of formulas are translated differently from negative occurrences [36]. It can therefore only be viewed as a translation on cut-free proofs, unlike Girard's symmetric translation of intuitionistic logic into linear logic.

In precise technical terms, the target of our translation is an "intuitionistic" version of MALL, presented by two-sided sequents with at most one consequent formula. Similar "intuitionistic" versions of various fragment of linear logic are considered in relationship to computer science, *e.g.,* in [35, 53, 9, 28, 37, 1, 58].

Apart from the foundational interest, we believe that the result of this chapter, which is theoretical in nature, contributes to the understanding of the role of linear logic as an expressive and natural framework for describing control structure of logic programs. This logic programming perspective is based on [73]; related work is in

Chapter 6

Linearizing Intuitionistic Implication

In this chapter we revisit the encoding of intuitionistic implication into linear logic pioneered by Girard [30]. In his original translation, which partially motivated the development of linear logic, Girard used the following:

$$A \Rightarrow B \qquad \rightarrow \qquad !(A) \multimap B$$

This encoding is quite beautiful, showing that intuitionistic implication is not necessarily an atomic primitive operation. Girard's embedding works at many levels of proof theory: formulas, proofs, and cut-elimination steps. Furthermore, this embedding extends naturally to first-order and second-order logic [30].

However, the complexity results of Chapter 5 raise the possibility of an improved translation. Statman [86] has shown that propositional intuitionistic logic, as well as its purely implicational fragment, is PSPACE-complete. Hence a natural question arises whether (beyond an immediate Turing reduction) there exists a "logical" embedding of intuitionistic logic into linear logic that does not rely on the modalities.

Let us be realistic. One cannot hope to have such an embedding which would be too "logical", because on the one hand, first-order multiplicative additive linear logic is decidable (basically because of a linear bound on the depth of cut-free proofs). On the other hand, first-order intuitionistic logic is undecidable (this is an immediate

the NP-completeness of multiplicative linear logic, but sharpens the result to fragments without propositions. The hardness proof relies on subtle aspects of 3-Partition, an NP-complete problem. The other main result presented earlier in this chapter is that MALL is PSPACE complete, a point which is exploited in the following chapter.

Another type of simplification can be achieved with other encodings of a 3-Partition problem. Consider the earlier encoding of 3-Partition in full multiplicative linear logic:

$$[(k \multimap c^{S(A_1)}) \otimes \cdots \otimes (k \multimap c^{S(A_{3M})}) \otimes (c^B \multimap j)^M] \multimap (k^3 \multimap j)^M$$

Constant-only encodings can be generated by replacing c by bottom, and k by $k^{\langle C \rangle}$ for some integer C. A value of C that is particularly interesting is $C = \sum_{a \in A} s(a)$. Although they are still polynomial, such encodings tend to be larger than the one advocated above, and result in somewhat less complicated proofs of soundness. The case of $C = 1$ is an incorrect encoding, and one may consider the "bottom only" encoding proved sound and complete above to be generated from the case $C = 0$.

5.2.4 Constant-Only Multiplicative Linear Logic is NP-Complete

From the preceding, we immediately achieve our stated result.

Theorem 5.2.53 (COMLL NP-COMPLETE) *The decision problem for constant-only multiplicative linear logic is np-complete.*

Also, with an easy conservativity result, we find that this NP-Hardness proof suffices for multiplicative linear logic as well.

Theorem 5.2.54 (Conservativity) *Multiplicative linear logic is conservative over constant-only multiplicative linear logic.*

Proof. By induction on cut-free MLL proofs. ∎

5.3 Summary of Chapter

In this chapter we have considered decision problems for several fragments of linear logic. Perhaps the highlight of this chapter is the NP-completeness of the constant-only fragment of multiplicative linear logic. This result follows Kanovich's results [49] on

problem instances satisfying $B/4 < s(a) < B/2$. For $B = 5$, all elements must be equal to 2, and thus there are no possible solutions. For $B = 8$, all elements must be equal to 3, and thus there are also no possible solutions in this case. For $B = 3$, all allowable problem instances have all elements equal to 1, and thus this case is solvable in polynomial (constant) time (report "YES"). For $B = 6$, similarly, all elements have size 2, and the answer is trivially "YES". For $B = 7$, all elements have size 2 or 3, and thus all partitions must be made up of two elements of size 2 and one element of size 3. The obvious counting algorithm thus solves this case in polynomial time. Thus for all cases where B is less than or equal to 8 the 3-Partition problem is solvable in polynomial time, and thus 3-Partition remains NP-complete with the further constraint that $B > 8$.

One may also consider the following looser specification of 3-Partition, which we will call 3-Partition'.

Instance: Set A' of $3m$ elements, a bound $B' \in Z^+$, and a size $s(a') \in Z^+$ for each $a' \in A'$

Question: Can A' be partitioned into m disjoint sets $A'_1, A'_2, \cdots, A'_m$ such that, for $1 \leq i \leq m$, $\sum_{a' \in A'_i} s(a') = B'$ such that each set contains exactly 3 elements from A'?

3-Partition' does not have a priori restrictions on the sizes of elements, but instead has an explicit specification that only partitions of three elements are allowed. One can immediately restrict $s(a')$ for all $a' \in A'$ to be $\leq B'$, for otherwise there is no solution, since all sizes are nonnegative.

From an instance of 3-Partition', one may generate an instance of 3-Partition by adding $B' + 1$ to the size of each element of A'. The instance of 3-Partition then is asked with $B = 4B' + 3$, and the size of each element satisfies the condition $B/4 \leq s(a) \leq B/2$, since $B = 4B' + 3$, $s(a') < B'$, and $s(a) = s(a') + B' + 1$. By adding more than $B' + 1$ to the size each element, one can create instances of 3-Partition where elements are as close to $B/3$ as desired. Thus one could avoid the complications involved in "reshuffling" the groups of four and two elements above by restricting the 3-Partition problem accordingly.

In the case that $n = 3$, we have $\sum_{1 \leq i \leq 3} Si = B$, and thus this sequent identifies a partition of the original problem meeting the requirement that the sum equal B.

Note that since there are exactly $3M$ elements, and $\sum_{a \in A} Sa = mB$, there are exactly the same number of groups with four elements as there are groups with two elements. Also note that if $n = 2$, we have by the above constraint that $S1 + S2 = B - 1$, and if $n = 4$, then $S1 + S2 + S3 + S4 = B + 1$.

Further, we may analyze by cases to show that if there are any groups of four, then $B = 4C + 3$ for some integer C. If there are any groups of four, and $B = 4C$ for some C, then the smallest allowable element is $C + 1$, since the size of each element must strictly dominate $B/4$. However, taking four elements of size $C + 1$, the constraint $S1 + S2 + S3 + S4 = B + 1$ is violated. Similarly for $B = 4C + 1$ and $B = 4C + 2$. Thus if there is a group of four elements, then $B = 4C + 3$ for some C, and by simple algebra, the elements of any group of four elements all have size $C + 1$, and the elements of any group of two elements both have size $2C + 1$. Noting that there are exactly as many groups of two as groups of four, we may rearrange the elements into groups of three by taking two elements from the group of four and one element from the group of two. Both resulting groups of three have total size $4C + 3$, which happily is equal to B.

Thus, given any proof of $\theta(\langle A, M, B, S \rangle)$, we first see that one may identify M branches, each of which is of the form $\vdash (1^{\langle S1 \rangle} \otimes \bot), (1^{\langle S2 \rangle} \otimes \bot), \cdots, (1^{\langle Sn \rangle} \otimes \bot), (\bot^B \parr 1^{\langle 3 \rangle})$. From these M branches, we may identify M partitions of three elements of the associate 3-Partition problem, some partitions directly from branches of the proof, and some partitions from a simple "reshuffling" of groups of four and two elements. In other words, from any proof of the given sequent, one may construct a solution to the 3-partition problem. ∎

Additional Constraints On 3-Partition

There are simplifying assumptions one can make about a 3-Partition problem which will alleviate the above minor difficulties mentioned above.

One may consider only those instances of 3-Partition where $B > 8$. One may show this by cases. If $B = 0$, or $B = 1$, $B = 2$, or $B = 4$ there are no possible

$\Sigma \bigcup \Delta \bigcup (1^{\langle Si \rangle} \otimes \bot)$ being equal to the conclusion:

$$\frac{\begin{array}{cc} \vdots & \vdots \\ \vdash \Sigma, \bot & \vdash \Delta, 1^{\langle Si \rangle} \end{array}}{\vdash (1^{\langle S1 \rangle} \otimes \bot), (1^{\langle S2 \rangle} \otimes \bot), \cdots, (1^{\langle S3M \rangle} \otimes \bot), (\bot^{B} \parr 1^{\langle 3 \rangle})^{M}} \otimes \mathbf{R}$$

But $1^{\langle Si \rangle}$ which occurs in one hypothesis has measure > 2. Therefore, the formula with negative measure, $(\bot^{B} \parr 1^{\langle 3 \rangle})^{M}$, must occur in Δ. Now consider the other hypothesis, which must contain $\bot$, and other formulas Σ from the conclusion sequent. If any formulas of the form $(1^{\langle Sj \rangle} \otimes \bot)$ are included in Σ, the measure of that hypothesis is greater than 1. If no such formulas are included, then the sequent has measure 0. In either case, by Girard's condition that sequent is not provable. Thus the assumption that one of the $(1^{\langle Sj \rangle} \otimes \bot)$ formulas is principal must be in error, and $(\bot^{B} \parr 1^{\langle 3 \rangle})^{M}$ must be principal.

Thus if $M > 1$, the only possible next proof step is $\otimes$, with principal formula $(\bot^{B} \parr 1^{\langle 3 \rangle})^{M}$. We may then focus on the case when $M = 1$. We claim that each remaining branch in such a proof corresponds (more or less) to one solution partition of the original partition problem. That is, we claim that when $M = 1$, we must be left with a sequent of the form:

$$\vdash (1^{\langle S1 \rangle} \otimes \bot), (1^{\langle S2 \rangle} \otimes \bot), \cdots, (1^{\langle Sn \rangle} \otimes \bot), (\bot^{B} \parr 1^{\langle 3 \rangle})$$

There are exactly $n + B - 1$ occurrences of $\otimes$ in this sequent, and $\sum_{1 \leq i \leq n} Si + 3$ ones in this sequent. By Girard's condition, $(\sum_{1 \leq i \leq n} Si + 3) - (n + B - 1) = 1$, or equivalently $\sum_{1 \leq i \leq n} Si = n + B - 3$.

If $n = 0$, we have $0 = B - 3$, which is false by our assumption that $B > 8$. If $n = 1$, we have $S1 = B - 2$, but the sizes are bounded by $B/2$, and with the assumption that $B > 8$, there is a contradiction. Also, considering cases of $n > 4$, we have $\sum_{1 \leq i \leq n} Si = n + B - 3$, and the assumptions that $B > 8$ and $S_i > B/4$, thus we have $n(B/4) > n + B - 3$, which implies that $fracn - 3(n/4) - 1 > B$ and from this and $B > 8$, we have $n < 5$. This leaves the $n = 2$, $n = 3$, and $n = 4$ cases.

Thus we have a proof with M branches, each of which represents either two, three, or four elements.

Completeness

Lemma 5.2.52 (Completeness) *For A, M, B, and S satisfying the constraints of 3-Partition, if there is a proof of the* COMLL *formula $\theta(\langle A, M, B, S\rangle)$, then the 3-Partition problem $\langle A, M, B, S\rangle$ is solvable.*

Proof.

To simplify this direction of the proof, we use the extra assumption that the "bin size" B is greater than 8. For a justification of this assumption, see Section 5.2.3. The following makes heavy use of Girard's condition on COMLL.

Assuming we have a proof of

$$\vdash (1^{\langle S1\rangle} \otimes \bot) \parr (1^{\langle S2\rangle} \otimes \bot) \parr \cdots \parr (1^{\langle S3M\rangle} \otimes \bot) \parr (\bot^B \parr 1^{\langle 3\rangle})^M$$

we show that the corresponding 3-Partition problem is solvable.

If there is a proof of this sequent, then there is a cut-free proof, by the cut elimination theorem (Theorem 2.3.4). By repeated applications of Property 2.6.7, if there is a proof of this sequent, then there is a proof of $\vdash (1^{\langle S1\rangle} \otimes \bot), (1^{\langle S2\rangle} \otimes \bot), \cdots, (1^{\langle S3M\rangle} \otimes \bot), (\bot^B \parr 1^{\langle 3\rangle})^M$

We then perform complete induction on M.

If $M > 1$, the proof of this sequent must end in $\otimes$, since all formulas have main connective $\otimes$. We next show that the principal formula of that rule application must be $(\bot^B \parr 1^{\langle 3\rangle})^M$.

First, we note that each formula $(1^{\langle Sj\rangle} \otimes \bot)$ has measure $Sj - 1$. Since we are assuming $B > 8$, the initial conditions of the 3-Partition problem ensure that for all j, $Sj > 2$, and therefore $Sj - 1 > 1$. There is only one formula, $(\bot^B \parr 1^{\langle 3\rangle})^M$, with negative measure.

If we assume that one of the $(1^{\langle Si\rangle} \otimes \bot)$ formulas is principal in an application of $\otimes$, by Girard's condition, each hypothesis sequent must have measure one. In this case we have the following supposed proof for some Σ and Δ with the multiset union

We will use the last form of this formula, since it contains no implicit negations (linear implication). One may see this formula satisfies Girard's measure condition if there are $3 * M$ elements, and the sum of the sizes equals $M * B$, side conditions on the statement of 3-Partition (Garey+Johnson).

The claim is that these formulas are provable in the multiplicative fragment of linear logic if and only if the 3-Partition problem is solvable.

Soundness

Lemma 5.2.51 (Soundness) *If a 3-Partition problem $\langle A, M, B, S\rangle$ is solvable, then we are able to find a proof of the* COMLL *formula $\theta(\langle A, M, B, S\rangle)$.*

Proof.

The proof is straightforward. For each group of three elements in the assumed solution to the 3-Partition problem, one forms the following subproof, assuming the elements of the group are numbered x, y, and z.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots \qquad \overline{\vdash 1}^{1}}{\vdash 1^{\langle Sx\rangle}, 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, \bot^{B} \quad \overline{\vdash 1, \bot}^{\bot}}\otimes \qquad \overline{\vdash 1}^{1}}{\vdash (1^{\langle Sx\rangle} \otimes \bot), 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, 1, \bot^{B} \qquad \overline{\vdash 1, \bot}^{\bot}}\otimes \qquad \overline{\vdash 1}^{1}}{\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), 1^{\langle Sz\rangle}, 1, 1, \bot^{B} \qquad \overline{\vdash 1, \bot}^{\bot}}\otimes}{\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1, 1, 1, \bot^{B}}\parr$$

$$\dfrac{\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1^{\langle 2\rangle}, 1, \bot^{B}}{\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1^{\langle 3\rangle}, \bot^{B}}\parr$$

$$\dfrac{\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), (1^{\langle 3\rangle} \parr \bot^{B})}{}\parr$$

The elided proof of $\vdash 1^{\langle Sx\rangle}, 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, \bot^{B}$ is guaranteed to exist by the conditions on the solution to 3-Partition. That is, since x, y, and z are from the same partition, the sum of Sx, Sy, and Sz must be equal to B.

Given the M proofs constructed as above from each of the M groups of elements, one combines them with $\otimes$ into a proof of

$$\vdash (1^{\langle S1\rangle} \otimes \bot), \cdots, (1^{\langle S3M\rangle} \otimes \bot), (1^{3} \parr \bot^{B})^{M}$$

The proof can then be completed with $3M$ applications of $\parr$. ∎

however, since the complexity of most larger linear logics have already been completely characterized [60].

Encoding

We recall the definition of 3-Partition:
(as stated in Garey+Johnson page 224)

Instance:	Set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$
Question:	Can A be partitioned into m disjoint sets $A_1, A_2, \cdots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly 3 elements from A)?
Reference:	[Garey+Johnson, 1975].
Comment:	NP-complete in the strong sense.

We will write $S1$ for $S(a_1)$ to improve readability of the following discussion.

Given an instance of 3-Partition equipped with a set $A = \{a_1, \cdots, a_{3M}\}$, an integer B, and a unary function S, presented as a tuple $\langle A, M, B, S \rangle$, we define the encoding function θ as $\theta(\langle A, M, B, S \rangle) =$

$$[(\bot \multimap \bot^{S1}) \otimes \cdots \otimes (\bot \multimap \bot^{S3M})] \multimap (\bot^3 \multimap \bot^B)^M$$

Notation: $x^Y = \overbrace{x \otimes x \otimes \cdots \otimes x \otimes x}^{Y \text{ copies}}$ $\quad$ $x^{\langle Y \rangle} = \overbrace{x \parr x \parr \cdots \parr x \parr x}^{Y \text{ copies}}$

Using the contrapositive ($A \multimap B \quad \equiv \quad B^\bot \multimap A^\bot$), we can develop a "1 only" encoding:

$$[(1^{\langle S1 \rangle} \multimap 1) \otimes (1^{\langle S2 \rangle} \multimap 1) \otimes \cdots \otimes (1^{\langle S3M \rangle} \multimap 1)] \multimap (1^{\langle B \rangle} \multimap 1^{\langle 3 \rangle})^M$$

Eliminating the linear implication in favor of $\parr$ these formulas become:

$$(1^{\langle S1 \rangle} \otimes \bot) \parr (1^{\langle S2 \rangle} \otimes \bot) \parr \cdots \parr (1^{\langle S3M \rangle} \otimes \bot) \parr (\bot^B \parr 1^{\langle 3 \rangle})^M$$

5.2.3 Constant-Only Case

Some time ago, Girard developed a necessary condition for the provability of constant multiplicative linear expressions:

$$\begin{aligned} M(1) &= 1 \\ M(\bot) &= 0 \\ M(A \parr B) &= M(A) + M(B) \\ M(A \otimes B) &= M(A) + M(B) - 1 \end{aligned}$$

If a formula A is provable in multiplicative linear logic and contains no propositions, then $M(A) = 1$. In other words, the number of tensors is one less than the number of ones in any provable constant only MLL (COMLL) formula. Avron (and others) have studied generalizations of this "semantic" measure to include propositions (where a proposition p is given value 1, and $p^\bot$ is given value 0) yielding a necessary condition for MLL provability. One may go even further, achieving a necessary condition for MALL provability, using min for & and max for $\oplus$, and plus and minus infinity for the additive constants. For the latter case, the condition becomes if a formula A is provable in MALL, then $M(A) \geq 1$. Also, one may generalize these conditions somewhat, replacing all instances of 1 with any arbitrary constant c, and allowing propositions to have different (although fixed) values, where p has value v_p, and $p^\bot$ has value $c - v_p$ [11].

Although the above condition is necessary, there has been a question as to whether some form of simple "truth table" or numerical evaluation function like the above could yield a necessary and sufficient condition for provability of constant multiplicative (COMLL) expressions. The main result of this paper shows that even this multiplicative constant evaluation or circuit evaluation problem is NP-complete.

We will encode an NP-complete problem, 3-Partition, in MLL, and show that our encoding is sound and complete. The main idea is that the small-proof property of MLL allows us to encode "resource distribution" problems naturally. Since linear logic treats propositions as resources natively, it has been called "resource-consciousness" [13]. Note that since full linear logic is conservative over MLL, our encoding remains sound and complete even in larger fragments. This does not lead to new results,

balancing argument on c's, the sum $Sx + Sy + Sz = B$, as required by 3-Partition. Thus we have shown that each branch of the proof where $M = 1$ corresponds exactly to one partition containing three elements whose sum of weights is B.

Thus by complete induction on M, one can show that from any proof of the given sequent, one can construct a solution to the 3-Partition problem.

Soundness

If the given 3-Partition problem $\langle A, M, B, S\rangle$ is solvable, then we are able to find a proof of the MLL formula $[\langle A, M, B, S\rangle]$.

The proof is straightforward. For each group of three elements in the assumed solution to the 3-Partition problem, one forms the following subproof, assuming the elements of the group are numbered x, y, and z.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{c^{Sx}, c^{Sy}, c^{Sz} \vdash c^B} \quad \overline{k \vdash k}^{I}}{(k \multimap c^{Sx}), c^{Sy}, c^{Sz}, k \vdash c^B}\multimap\mathbf{L} \quad \overline{k \vdash k}^{I}}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), c^{Sz}, k, k \vdash c^B}\multimap\mathbf{L} \quad \overline{k \vdash k}^{I}}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k, k, k \vdash c^B}\multimap\mathbf{L}}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k^2, k \vdash c^B}\otimes\mathbf{L}}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k^3 \vdash c^B}\otimes\mathbf{L}}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}) \vdash (k^3 \multimap c^B)}\multimap\mathbf{R}$$

The elided proof of $c^{Sx}, c^{Sy}, c^{Sz} \vdash c^B$ is guaranteed to exist by the conditions on the solution to 3-Partition, and consists entirely of $\otimes\mathbf{R}$ and $\mathbf{I}$ rules.

Given the M proofs constructed as above from each of the M groups of elements, one combines them with $\otimes\mathbf{R}$ into a proof of

$$(k \multimap c^{S1}), \cdots, (k \multimap c^{S3M}) \vdash (k^3 \multimap c^B)^M$$

The proof can then be completed with $3M - 1$ applications of $\otimes\mathbf{L}$, and ends with one application of $\multimap\mathbf{R}$.

Informally, every $\otimes\mathbf{R}$ proof step applied to a formula $(k^3 \multimap c^B)^M$ corresponds to dividing a group of elements.

Formally, we now consider in detail the part of the above proof:

$$\frac{\vdots}{(k \multimap c^{S1}), \cdots, (k \multimap c^{S3M}) \vdash (k^3 \multimap c^B)^M}$$

We now perform complete induction on M.

We show that if $M \neq 1$, the only possible next proof step is $\otimes\mathbf{R}$. If any other proof rule is applied to this sequent, one or both of the hypotheses will be unbalanced, and therefore unprovable. We show this by cases. The only formulas which may be principal in a rule other than $\otimes\mathbf{R}$ are of the form $k \multimap c^{Si}$. If the $\multimap$L rule is applied, we have the following supposed proof for some Σ and Δ with the multiset union $\Sigma \bigcup \Delta \bigcup (k \multimap c^{Si})$ being equal to the left hand side of the conclusion:

$$\frac{\begin{array}{cc}\vdots & \vdots \\ \Sigma \vdash k & \Delta, c^{Si} \vdash (k^3 \multimap c^B)^M\end{array}}{(k \multimap c^{S1}), \cdots, (k \multimap c^{S3M}) \vdash (k^3 \multimap c^B)^M} \multimap L$$

However, given that Σ is made up of formulas from the conclusion, $\Sigma \vdash k$ cannot be balanced. The formula k has positive polarity, and the only formulas which contain k with negative polarity occur in $(k^3 \multimap c^B)^M$. Thus if $M \neq 1$, the only possible next proof step is $\otimes\mathbf{R}$.

We may then focus on the case when $M = 1$. We claim that each remaining branch in such a proof corresponds exactly to one solution partition of the original partition problem. That is, we claim that when $M = 1$, we must be left with a sequent of the form:

$$(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}) \vdash (k^3 \multimap c^B)$$

Where $Sx + Sy + Sz = B$. That is, exactly three formulas of the form $k \multimap c^{Sx}$ in the antecedent, consequent $k^3 \multimap c^B$, and the sum of the weights is equal to B, as required by 3-Partition. To show this, note that we know the consequent is $k^3 \multimap c^B$. By counting k's, there must be exactly three formulas of the form $k \multimap c^{Sx}$. By another

We will write $S1$ for $S(a_1)$ to improve readability of the following discussion.

Encoding

Given an instance of 3-Partition equipped with a set $A = \{a_1, \cdots, a_{3M}\}$, an integer B, and a unary function S, presented as a tuple $\langle A, M, B, S\rangle$, we define the encoding function θ as $\theta(\langle A, M, B, S\rangle) =$

$$[(k \multimap c^{S1}) \otimes \cdots \otimes (k \multimap c^{S3M})] \multimap (k^3 \multimap c^B)^M$$

Reminder of notation: k, c propositions, $x^Y = \overbrace{x \otimes x \otimes \cdots \otimes x \otimes x}^{Y \text{ copies}}$

The claim is that this formula is provable in the multiplicative fragment of linear logic if and only if the 3-Partition problem is solvable.

Completeness

This is the more difficult direction of the proof, and requires that we show that if there is a proof of the linear logic formula $\theta(\langle A, M, B, S\rangle)$, then the 3-Partition problem $\langle A, M, B, S\rangle$ has a solution. We make critical use of the proofs-are-balanced property of MLL, and use two simple permutability properties of linear logic:

- $\otimes$**L** permutes down (except if principle)
- $\multimap$**R** permutes down (except if principle)

If there is a proof of $\Theta(\langle A, M, B, S\rangle)$, then there is a cut-free proof, by the cut-elimination theorem 2.3.4. Given a cut-free proof, we first permute all applications of $\otimes$L and $\multimap$**R** down as far as they will go. We are then left with some proof of the form:

$$\dfrac{\dfrac{\dfrac{\vdots}{(k \multimap c^{S1}), \cdots, (k \multimap c^{S3m}) \vdash (k^3 \multimap c^B)^M}\otimes L}{\vdots}}{\dfrac{(k \multimap c^{S1}) \otimes \cdots \otimes (k \multimap c^{S3M}) \vdash (k^3 \multimap c^B)^M}{\vdash (k \multimap c^{S1}) \otimes \cdots \otimes (k \multimap c^{S3m}) \multimap (k^3 \multimap c^B)^M} \multimap R}$$

Proof. We proceed by induction on the depth of assumed proof. In the base case, all axioms are balanced, as are the axioms for the multiplicative constants. Inductively, by case analysis, we find that if all hypotheses of an IMLL or CMLL inference rule are balanced, then the conclusion must also be balanced. ∎

Note that this property fails for other fragments of linear logic such as IMALL which include the additive connectives and constants: $\&$, $\oplus$, $\top$, and 0, and also fails to hold in the presence of exponential connectives ! and ?.

Also note that this property follows immediately from Girard's proof net conditions on MLL formulas. Axiom links in a proof net must connect each proposition with exactly one proposition of opposite polarity, and thus all provable conclusions must be balanced.

We will encode an NP-complete problem, 3-Partition, in MLL, and show that our encoding is sound and complete. The main idea is that the small-proof property of MLL allows us to encode "resource distribution" problems naturally. Since linear logic treats propositions as resources natively, it has been called "resource-consciousness" [13]. Note that since full linear logic is conservative over MLL, our encoding remains sound and complete even in larger fragments. This does not lead to new results, however, since the complexity of most larger linear logics have already been completely characterized [60].

3-Partition

We use the NP-completeness of 3-Partition:
(as stated in Garey+Johnson page 224)

Instance:	Set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$
Question:	Can A be partitioned into m disjoint sets $A_1, A_2, \cdots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly 3 elements from A)?
Reference:	[Garey+Johnson, 1975].
Comment:	NP-complete in the strong sense.

5.2.2 IMLL is NP-Hard

In this section we analyze the inherent difficulty of deciding IMLL sequents. We show that one can encode a 3-Partition problem in IMLL using unary notation. Since 3-Partition is NP-complete in the strong sense, we thus have the result that IMLL is NP-hard. We show that our encoding is correct by relying on a property of IMLL (which also holds of CMLL) which greatly restricts the space of possible IMLL proof trees. This "balanced" property is related to the tremendously useful "lock and key" mechanism used in [60, 62] for MALL and IMALL.

We begin with a few definitions. We recall the definition of *polarity* of a formula from Section 2.5

$$\begin{array}{rcl}
[A \multimap B]^+ & = & [A]^- \multimap [B]^+ \\
[A \otimes B]^+ & = & [A]^+ \otimes [B]^+ \\
[\Sigma, A]^+ & = & [\Sigma]^+, [A]^+ \\
[A^\perp]^+ & = & ([A]^-)^\perp \\
[A \multimap B]^- & = & [A]^+ \multimap [B]^- \\
[A \otimes B]^- & = & [A]^- \otimes [B]^- \\
[\Sigma, A]^- & = & [\Sigma]^-, [A]^- \\
[A^\perp]^- & = & ([A]^+)^\perp
\end{array}$$

The polarity of an instance of a formula A in a sequent $\Sigma \vdash \Delta$ is given by the sign of the superscript on A in $[\Sigma]^-$ or $[\Delta]^+$. That is, if an instance of formula A ends up as $[A]^+$, then it is of *positive* polarity. If an instance of formula A ends up as $[A]^-$, then it is of *negative* polarity.

We define a sequent to be *balanced* if the number of occurrences of p_i with positive polarity and negative polarity are equal. Otherwise, we say a sequent is *unbalanced*. We show that all provable IMLL and CMLL sequents are balanced, and therefore all unbalanced sequents are not provable. This property was previously discussed in [47, 11], and many other places.

Proposition 5.2.50 (proofs are balanced) *An* IMLL *or* CMLL *sequent is provable only if it is balanced.*

5.2 Multiplicative Linear Logic is NP-Complete

In this section it is shown that the decision problem for propositional multiplicative linear logic is NP-complete. An argument for the NP-hardness of this fragment was first sketched by Max Kanovich in electronic mail [48]. Together with the earlier result [60] that the multiplicatives are in NP, Kanovich's result showed that this decision problem is NP-complete. Kanovich later updated his argument to show that the "Horn fragment" of the multiplicatives is also NP-complete [47, 49], using a novel computational interpretation of this fragment of linear logic. This section continues this trend by providing a proof that evaluating expressions in *and*, *or*, *true*, and *false* in multiplicative linear logic is NP-complete. That is, even without propositions, multiplicative linear logic is NP-complete.

This section begins with a proof that intuitionistic and classical Multiplicative Linear Logic are in NP. Then it is shown that IMLL is NP-hard. Then it is demonstrated that the multiplicative circuit evaluation problem is NP-complete.

5.2.1 IMLL and CMLL Are In NP

Informally, the argument showing membership in NP is simply that every connective in a multiplicative linear logic formula is analyzed exactly once in any cut-free proof. Thus an entire proof, if one exists, can be guessed and checked in nondeterministic polynomial time.

Theorem 5.2.49 (Small-Proofs) *Every connective is analyzed exactly once in any cut-free* CMLL *or* IMLL *proof.*

From Theorem 2.3.4 and Theorem 5.2.49, we know that given a CMLL or IMLL sequent of size n, if there is any proof of this sequent, then there is a proof with exactly n total applications of inference rules. Since each application of an inference rule may be represented in space linear in n, we may simply guess and check an entire n^2 representation of a proof tree in nondeterministic polynomial time.

5.1.7 IMALL 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, showing that intuitionistic MALL is also PSPACE-complete.

Corollary 5.1.48 IMALL *provability is* PSPACE*-complete.*

$$\dfrac{\dfrac{\vdots}{\vdash \langle I\rangle, x_m, q_{m-1}, [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g}}{\vdash \langle I\rangle, (x_m \parr q_{m-1}), [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g}\parr$$
$$\dfrac{}{\vdash \langle I\rangle, ((x_m \parr q_{m-1}) \oplus (x_m^{\perp} \parr q_{m-1})), [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g}\oplus$$

Then by the induction hypothesis applied to the proof of the sequent

$$\vdash \langle I\rangle, x_m, q_{m-1}, [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g$$

we get $I, \neg X_m \models Q_{m-1}X_{m-1}\ldots Q_0X_0 : M$, and hence $I \models \exists X_m Q_{m-1}X_{m-1}\ldots Q_0X_0 : M$.

The argument is similar when the right $\oplus$ reduction is applied in the given cut-free proof.

The proof when $Q_m \equiv \forall$ is also similar. ∎

When $m = n$ in Lemma 5.1.46, it follows that a closed QBF G is valid iff $\sigma(G)$ is provable in MALL. Since σ is a log-space encoding of a given QBF, the final result below follows immediately from Theorems 5.1.46 and 5.1.37.

Theorem 5.1.47 MALL *provability is* PSPACE*-complete.*

compound formulas in the conclusion sequent $\vdash q_m, \langle I \rangle, [\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g, g$. From the MALL rules, it is clear that the only applicable reduction in a cut-free proof search would be an application of the $\otimes$-rule. Hence for some k, we can partition the formulas other than $q_k^\perp \otimes A_k$ in the conclusion sequent into Γ and Δ to get a deduction of the conclusion sequent of the following form.

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash \Gamma, q_k^\perp & \vdash A_k, \Delta\end{array}}{\vdash \Gamma, (q_k^\perp \otimes A_k), \Delta}\otimes$$

Suppose for the sake of deriving a contradiction that $k < m$. Recall that there are no constants in the encoding. The formula $q_{k+1}^\perp \otimes A_{k+1}$ must either occur in Γ or Δ, and definitely not in both. Since the only occurrences of q_k are within A_{k+1}, by Lemma 5.1.45, if $q_{k+1}^\perp \otimes A_{k+1}$ occurs in Δ, then we cannot complete the proof $\vdash q_k, \Gamma$. Thus we assume $q_{k+1}^\perp \otimes A_{k+1}$ occurs in Γ. It is easy to see by inspection of the form of A_{k+1} that the only occurrences of q_k in A_{k+1} have the form $q_k \circ B$ or the form $B \circ q_k$, where $\circ$ is either $\oplus$, $\&$, or $\otimes$. Therefore, again by Lemma 5.1.45, $\vdash q_k^\perp, \Gamma$ is not provable.

Thus it follows that $k \not< m$.

When $k = m$, we can apply Lemma 5.1.45 to infer that $\Gamma \equiv q_m$, since otherwise, Γ would not contain any occurrences of q_m as immediate arguments to $\otimes$, $\&$ or $\oplus$. If $Q_m \equiv \exists$, this yields the deduction

$$\dfrac{\overline{\vdash q_m, q_m^\perp}\,\mathbf{I} \qquad \begin{array}{c}\vdots \\ \vdash \langle I \rangle, ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), [\![Q_{m-1} X_{m-1} \ldots Q_0 X_0 : M]\!]_g, g\end{array}}{\vdash \langle I \rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1}))), [\![Q_{m-1} X_{m-1} \ldots Q_0 X_0 : M]\!]_g, g}\otimes$$

For the same reason as before, the remaining subgoal cannot be reduced by applying the $\otimes$-rule to a formula $q_i^\perp \otimes A_i$ since all of the occurrences of q_i remain as immediate arguments to $\parr$. The only possible reduction then is to "unwind" the quantifier encoding for $Q_m X_m$ as in the ($\Leftarrow$) direction of the proof until q_{m-1} is introduced as a sequent formula. If the left $\oplus$ reduction is applied in the given cut-free proof, we have

$$\dfrac{\dfrac{}{\vdash q_m, q_m^\perp}\mathbf{I} \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m^\perp, q_{m-1}, [\![G]\!]_g, g\end{array}}{\vdash \langle I\rangle, ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), [\![G]\!]_g, g}\oplus}{\vdash \langle I\rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1}))), [\![G]\!]_g, g}\otimes$$

Since $\langle I, X_m\rangle = \langle I\rangle, x_m^\perp$, the induction hypothesis can be applied to show that the remaining subgoal of the above deduction is provable.

When $I, \neg X_m \models G$, the proof construction only differs from the above one on the $\oplus$ rule corresponding to the quantifier encoding.

If $Q_m \equiv \forall$, then

$$[\![\forall X_m\colon G]\!]_g = (q_m^\perp \otimes ((x_m \parr q_{m-1})\&(x_m^\perp \parr q_{m-1}))), [\![G]\!]_g$$

Since $I \models \forall X_m\colon G$, it follows that $I, X \models G$ and $I, \neg X \models G$. The following deduction can be constructed.

$$\dfrac{\dfrac{}{\vdash q_m, q_m^\perp}\mathbf{I} \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m, q_{m-1}, [\![G]\!]_g, g\end{array} \qquad \begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m^\perp, q_{m-1}, [\![G]\!]_g, g\end{array}}{\vdash \langle I\rangle, ((x_m \parr q_{m-1})\&(x_m^\perp \parr q_{m-1})), [\![G]\!]_g, g}\&}{\vdash \langle I\rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1})\&(x_m^\perp \parr q_{m-1}))), [\![G]\!]_g, g}\otimes$$

Since $\langle I, X_m\rangle$ is $\langle I\rangle, x_m^\perp$ and $\langle I, \neg X_m\rangle$ is $\langle I\rangle, x_m$, the two remaining subgoals in the deduction are provable by the induction hypotheses.

Induction step $\Leftarrow$: This is the critical step in the proof. We are given that $m > 0$ and that the conclusion sequent $\vdash q_m, \langle I\rangle, [\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g, g$ is provable. Theorem 5.1.36 can be applied to construct a cut-free proof of $\vdash q_m, \langle I\rangle, [\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g, g$. We show that this cut-free proof respects the quantifier ordering, i.e., the reduction of the encoding of $Q_m X_m$ occurs below any other step in the proof.

It is easy to see that every formula in the multiset $[\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g$ is of the form $q_i^\perp \otimes A_i$, for $0 \le i \le m$, with $A_0 \equiv [M]_g$, and $A_{j+1} \equiv ((x_{j+1} \parr q_j) \circ (x_{j+1}^\perp \parr q_j))$. The connective written as $\circ$ can be either & or $\oplus$. The formulas $q_i^\perp \otimes A_i$ are the only

The following lemma demonstrates the correctness of the MALL encoding of Boolean quantifiers. Each Q_i in the statement below is either $\forall$ or $\exists$.

Lemma 5.1.46 (Main Induction) *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 $X_{m+1}, \ldots, X_n$, the relation $I \models Q_m X_m \ldots Q_1 X_1 : M$ holds iff the sequent $\vdash q_m, \langle I \rangle, [\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g, g$ is provable in* MALL.

Proof. The proof is by induction on m between 0 and n. Note that I is universally quantified in the induction hypothesis.

Base case $\Rightarrow$: Here $m = 0$. Then $[\![M]\!]_g \equiv q_0^\perp \otimes [M]_g$, and we can easily construct the following deduction of the required conclusion $\vdash q_0, \langle I_0 \rangle, [\![M]\!]_g, g$.

$$\dfrac{\dfrac{}{\vdash q_0, q_0^\perp}\mathbf{I} \qquad \begin{array}{c}\vdots \\ \vdash \langle I \rangle, [M]_g, g\end{array}}{\vdash \langle I \rangle, q_0, q_0^\perp \otimes [M]_g, g}\otimes$$

The proof of the remaining subgoal $\vdash \langle I \rangle, [G]_g, g$, follows from Lemma 5.1.40.

Base case $\Leftarrow$: The deduction shown above is the only possible one in a cut-free proof of $\vdash \langle I \rangle, q_0, q_0^\perp \otimes [M]_g, g$ since $q_0^\perp \otimes [G]$ is the only compound formula in the conclusion. So if $\vdash \langle I \rangle, q_0, q_0^\perp \otimes [M]_g, g$ is provable, by Theorem 5.1.36, it must have a cut-free proof containing a proof of $\vdash \langle I \rangle, [M]_g, g$. By Lemma 5.1.43, we get $I \models M$.

Induction step $\Rightarrow$: Assume $0 < m \leq n$. Let G abbreviate $Q_{m-1} X_{m-1} \ldots Q_1 X_1 : M$. We must prove the lemma for $Q_m X_m G$. If $Q_m \equiv \exists$, then

$$[\![Q_m X_m : G]\!]_g = (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1}))), [\![G]\!]_g$$

If $I \models \exists X_m : G$, then either $I, X_m \models G$ or $I, \neg X_m \models G$. In the former case, the following deduction of the required conclusion can be constructed.

quantifier encoding in any cut-free proof. To achieve this, we need to argue that the key q_i needed to unlock the ith quantifier encoding is only made available when the $(i+1)$st quantifier encoding has been reduced. In order for the ith quantifier encoding, which has the form $q_i^\perp \otimes U_i$, to be reduced before the $(i+1)$st quantifier encoding, a subgoal of the form $\vdash q_i^\perp, \Gamma$ would have to be provable. The only occurrences of q_i are in the subformula U_{i+1} given by $(q_i \parr x_{i+1}) \circ (q_i \parr x_{i+1}^\perp)$, where $\circ$ may be either $\oplus$ or $\&$. If U_{i+1} occurs in Γ, then the only occurrences of q_i in Γ are as immediate arguments to a $\parr$. By exploiting the absence of an unrestricted weakening rule in MALL, it can be shown that in the absence of constants, $\vdash q_i^\perp, \Gamma$ is not provable when all of the occurrences of q_i in Γ appear as immediate arguments to $\parr$. Therefore, regardless of whether U_{i+1} occurs in Γ, the sequent $\vdash q_i^\perp, \Gamma$ would not be provable, thus making it impossible to reduce the ith quantifier encoding below the $(i+1)$st quantifier encoding in a cut-free proof.

Lemma 5.1.45 *If q is a positive or negative literal and the sequent $\vdash q, \Gamma$ contains no constants, then $\vdash q, \Gamma$ is provable only if either $\Gamma \equiv q^\perp$ or Γ contains at least one occurrence of a subformula either of the form $q^\perp \circ A$, or the form $A \circ q^\perp$, where $\circ$ may be either $\oplus$, $\&$, or $\otimes$.*

Proof. We fix $\circ$ to be either $\oplus$, $\&$, or $\otimes$ for this proof. The proof is by induction on cut-free MALL proofs of $\vdash q, \Gamma$. In the base case, for a cut-free proof of depth 0, the sequent $\vdash q, \Gamma$ must be a MALL axiom, and $\Gamma \equiv q^\perp$ holds.

In the induction step, when in the given cut-free proof of $\vdash q, \Gamma$, the conclusion sequent is derived by an application of either a $\otimes$, $\&$ or a $\oplus$ rule, then at least one premise must be of the form $\vdash q, \Delta$. We know by the induction hypothesis for the proof of $\vdash q, \Delta$, either $\Delta = q^\perp$ or Δ either contains a subformula of the form $q^\perp \circ A$, or the form $A \circ q^\perp$. In either case, Γ contains one of the forms, $q^\perp \circ A$ or $A \circ q^\perp$.

If in the cut-free proof of $\vdash q, \Gamma$, the conclusion sequent is derived by an application of the $\parr$ rule, the premise sequent must be of the form $\vdash q, \Delta$, where Δ is not a single formula, Then by the induction hypothesis on the proof of $\vdash q, \Delta$, the sequence Δ must contain one of the forms, $q^\perp \circ A$ or $A \circ q^\perp$. Since every subformula of Δ is a subformula of Γ as well, Γ must also contain one of the forms $q^\perp \circ A$ or $A \circ q^\perp$. ∎

that K agrees with K_1 on $[N]_a$ and with K_2 on $[P]_b$. Each disjunct in $\text{AND}(a, b, g)$ expanded as

$$\begin{array}{ll} (a \otimes b \otimes g^{\perp}) & \oplus \\ (a^{\perp} \otimes b^{\perp} \otimes g) & \oplus \\ (a \otimes b^{\perp} \otimes g) & \oplus \\ (a^{\perp} \otimes b \otimes g) & \end{array}$$

is falsified by K. As already observed, the COPY formulas are all classically false, and thus K falsifies $\langle I \rangle, [M]_g$. Since in this case, $I \not\models N \wedge P$, the second part of the conclusion is also satisfied.

The remaining cases are similar. ∎

Lemma 5.1.43 *If I is an assignment for the variables in a given Boolean formula M, then*

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

Proof. By Lemma 5.1.41, we know that if $\vdash \langle I \rangle, [M]_g, g$ is provable, then no assignment can simultaneously falsify $\langle I \rangle$, $[M]_g$, and g under the classical interpretation. By Lemma 5.1.42, we can find an assignment K which falsifies $\langle I \rangle$ and $[M]_g$ such that $K(g) = T$ iff $I \models M$. Since K cannot also falsify g, $K(g) = T$ and hence $I \models M$.

Similarly, when $\vdash \langle I \rangle, [M]_g, g^{\perp}$ is provable, we can, by Lemmas 5.1.42 and 5.1.41, find an assignment K such that $K(g) = F$ and as a consequence, $I \not\models M$. ∎

Lemma 5.1.44 *$\vdash \langle I \rangle, [M]_g, g$ is provable iff $I \models M$.*

Proof. Follows immediately from Lemmas 5.1.40 and 5.1.43. ∎

So far, we have demonstrated the correctness of the encoding of the Boolean matrix of a given quantified Boolean formula. The remainder of the proof deals with the encoding of Boolean quantifiers. The next lemma states the crucial reason why the MALL encoding of quantifiers is faithful to the quantifier orderings. As observed in Subsection 5.1.4, the goal is to ensure that in any successful proof search, the ith quantifier encoding is reduced after, i.e, above, the reduction of the $(i + 1)$st

Proof. The proof is by induction on the construction of $[M]_g$. Note that the induction is parametric in I and g (I and g are universally quantified in the induction hypothesis), so that when $M \equiv (N \wedge P)$, the induction hypothesis on N has I/N replacing I and a replacing g, where a labels the output of N.

Base case: $M \equiv X$. Then $[M]_g = (x^\perp \otimes g) \oplus (x \otimes g^\perp)$. If $I \models X$, then $I(X) = T$ and $\langle I \rangle = x^\perp$, and $\langle I \rangle$ is falsified if K assigns T to x. $[M]_g$ is falsified if K assigns T to g, and the second part of the conclusion, $K(g) = T$ also follows. If $I \not\models X$, then $I(X) = F$. Let K assign F to x and F to g to falsify both $\langle I \rangle$ and $[M]_g$. Then $K(g) = F$ as required.

Induction step: Observe first that the formula $\textsc{copy}(x)$ defined as $(x \otimes (x^\perp \parr x^\perp)) \oplus (x^\perp \otimes (x \parr x))$ is classically false.

When $M \equiv \neg N$, the encoding $[M]_g$ is $\textsc{not}(a, g), [N]_a$. By the induction hypothesis, we have an assignment K_1 falsifying $\langle I \rangle, [N]_a$ such that $K_1(a) = T$ iff $I \models N$. Suppose $K_1(a) = T$, and hence $I \models N$. The formula $\textsc{not}(a, g)$ is $(a \otimes g) \oplus (a^\perp \otimes g^\perp)$. Let K be $K_1\{g \leftarrow F\}$. Since g does not occur in $\langle I \rangle$ or $[N]_a$, K agrees with K_1 on $\langle I \rangle, [N]_a$. The assignment K also falsifies $\textsc{not}(a, g)$, thus falsifying $\langle I \rangle, [M]_g$. Note that $K(g) = F$ as required, since $I \not\models M$.

If $K_1(a) = F$, then $I \not\models N$. Letting K be $K_1\{g \leftarrow T\}$ falsifies $\langle I \rangle, [M]_g$.

When $M \equiv (N \wedge P)$, then by the induction hypotheses for N and P, there exists

1. K_1 falsifying $\langle I/N \rangle, [N]_a$ such that $K_1(a) = T$ iff $I/N \models N$, and
2. K_2 falsifying $\langle I/P \rangle, [P]_b$ such that $K_2(a) = T$ iff $I/P \models P$

The encoding $\langle I \rangle$ is a sequence of literals such that no two distinct literals in $\langle I \rangle$ share a common atom. Since $\langle I/N \rangle$ and $\langle I/P \rangle$ are subsets of $\langle I \rangle$, there is no literal x such that x is in $\langle I/N \rangle$ and $x^\perp$ is in $\langle I/P \rangle$. Formulas $[N]_a$ and $[P]_b$ have no atoms in common outside of those in $\langle I \rangle$. Then the union of the assignments $K_1 \cup K_2$, is still an assignment, i.e., it assigns a unique truth value to each *atom* in $\langle I \rangle, [N]_a, [P]_b$.

Suppose that $K_1(a) = T$, and hence $I/N \models N$, and $K_2(b) = F$, so that $I/P \not\models P$. Let K be $(K_1 \cup K_2)\{g \leftarrow F\}$. Note that g does not occur in $\langle I \rangle$, $[N]_a$ or $[P]_b$ so

$A \parr B$ and $A \oplus B$ are read as classical disjunction. A sequent is interpreted as the classical disjunction of the formulas contained in it.

Lemma 5.1.41 *If $\vdash \Gamma$ is a provable* MALL *sequent, then for any assignment of truth values to the atoms in Γ, there exists a formula A in the sequence Γ such that A is true under the classical interpretation.*

Proof. The proof is by a straightforward induction on cut-free MALL proofs. Clearly, for axioms $\vdash x, x^\perp$, one of x or $x^\perp$ must evaluate to T in a given truth assignment. In the induction case, suppose that the last step in the proof of Γ is a $\otimes$-rule of the form

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash B, \Gamma_1 & \vdash C, \Gamma_2\end{array}}{\vdash (B \otimes C), \Gamma_1, \Gamma_2}\otimes$$

By the induction hypothesis, the sequence B, Γ_1 contains a formula A_1, and the sequence C, Γ_2 contains a formula A_2, and both A_1 and A_2 are true. If A_1 is different from B, then A_1 occurs in the conclusion sequent yielding the required A, and similarly, when A_2 is different from C. Otherwise, the formula $(B \otimes C)$ is $(A_1 \otimes A_2)$ and is hence true under the classical interpretation of $\otimes$ as conjunction. The induction arguments corresponding to the other connectives are similar. ∎

The main intuition behind Lemma 5.1.42 is that by appropriately assigning truth values to the literals in $\langle I \rangle$ and $[M]_g$, it is possible to mimic the evaluation of the Boolean formula M under I. Due to our use of one-sided sequents and the form of our encoding, there is exactly one truth value falsifying each formula in $\langle I \rangle$ and $[M]_g$. This assignment turns out be the appropriate one, i.e., the value of g under this assignment is T exactly when $I \models M$. For example, if I is $\{X \leftarrow F\}$ and M is $\neg X$, then $\langle I \rangle$ is $x^\perp$ and $[M]_g$ is $(x \otimes g) \oplus (x^\perp \otimes g^\perp)$. The only falsifying assignment here is $\{x \leftarrow F, g \leftarrow T\}$.

Lemma 5.1.42 *Let M be a Boolean formula and I be an assignment for the variables in M. There exists an assignment K of truth values to the atoms in $\langle I \rangle$ and $[M]_g$ such that for every formula A in the sequence $\langle I \rangle, [M]_g$, assignment K falsifies A under the classical interpretation, and $K(g) = T$ iff $I \models M$.*

Base case: $M \equiv X$. Suppose $I(X) = T$, then $I \models M$ and $\langle I \rangle = x^\perp$. The following proof can then be constructed, expanding the definition of $[M]_g$

$$\dfrac{\dfrac{\overline{\vdash x^\perp, x}^{\mathbf{I}} \quad \overline{\vdash g^\perp, g}^{\mathbf{I}}}{\vdash x^\perp, (x \otimes g^\perp), g}\otimes}{\vdash x^\perp, (x^\perp \otimes g) \oplus (x \otimes g^\perp), g}\oplus$$

The case when $I(X) = F$ is similarly straightforward.

Induction step: There are a number of cases here corresponding to the definition of $[M]_g$. We consider a typical case and leave the remaining ones to the reader.

Let $M \equiv N \wedge P$, and suppose that $Var(N) \cap Var(P) \neq \emptyset$. Consider the case when $I/N \models N$ and $I/P \not\models P$, so that $I \not\models N \wedge P$. Expanding $[M]_g$, $\textsc{and}(a, b, g)$, and using Lemma 5.1.39, the following deduction can be constructed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots \qquad\qquad \vdots}{\dfrac{\overline{\vdash g, g^\perp}^{\mathbf{I}} \qquad \dfrac{\vdash \langle I/N \rangle, [N]_a, a \quad \vdash \langle I/P \rangle, [P]_b, b^\perp}{\vdash \langle I/N \rangle, \langle I/P \rangle, (a \otimes b^\perp), [N]_a, [P]_b}\otimes}{\vdash \langle I/N \rangle, \langle I/P \rangle, (a \otimes b^\perp \otimes g), [N]_a, [P]_b, g^\perp}\otimes}{\vdash \langle I/N \rangle, \langle I/P \rangle, \textsc{and}(a, b, g), [N]_a, [P]_b, g^\perp}\oplus}{\vdots}}{\dfrac{\vdash \langle I \rangle, \textsc{and}(a, b, g), \textsc{copyall}(Var(N) \cap Var(P)), [N]_a, [P]_b, g^\perp}{\vdash \langle I \rangle, \textsc{and}(a, b, g) \parr \textsc{copyall}(Var(N) \cap Var(P)) \parr [N]_a \parr [P]_b, g^\perp}\parr}$$

Applying the induction hypothesis to I/N, N, and a, and to I/P, P, and b, we can establish that the remaining subgoals of the deduction are provable.

The remaining subcases in the evaluation of $N \wedge P$ are similar, as are the remaining cases in the induction argument. ∎

The next step is to establish the converse of Lemma 5.1.40. The classical interpretation of the MALL connectives may be used to give a relatively easy proof. In the classical interpretation, truth values, T and F, are assigned to the MALL atoms, $A^\perp$ is read as classical negation, $A \otimes B$ and $A \& B$ are read as classical conjunction, and

Each pass would use only log-space, and the remainder of the algorithm may be performed in log-space.

5.1.6 Proof of PSPACE-hardness of MALL

The main theorem is that for any closed QBF G, G is valid if and only if $\sigma(G)$ is a provable MALL sequent. The first set of lemmas demonstrates that the encoding of Boolean formulas works correctly. The second set of lemmas demonstrates that the Boolean quantifiers have been correctly encoded.

If I is a truth value assignment for the Boolean variables $X_1, \ldots, X_n$, then I is encoded as $\langle I \rangle$, where

$$\begin{aligned} \langle I \rangle &= \langle X_1 \rangle_I, \ldots, \langle X_n \rangle_I \\ \langle X_i \rangle_I &= \begin{cases} x_i^\perp & \text{if } I(X_i) = T \\ x_i, \text{ otherwise} \end{cases} \end{aligned}$$

If I is an assignment for a set of variables $\overline{Y}$, and $\overline{X} \subseteq \overline{Y}$, then $I/\overline{X}$ is the assignment I restricted to the subset $\overline{X}$, and by abuse of notation I/M is $I/Var(M)$. The following lemma is stated without proof.

Lemma 5.1.39 *Given sets of variables $\overline{X}$ and $\overline{Y}$, and an assignment I for $\overline{X} \cup \overline{Y}$, there is a deduction of the sequent $\vdash \langle I \rangle, \text{COPYALL}(\overline{X} \cap \overline{Y}), \Gamma$ from the sequent $\vdash \langle I/\overline{X} \rangle, \langle I/\overline{Y} \rangle, \Gamma$.*

Lemma 5.1.40 *Let M be a Boolean formula and I an assignment for the variables in M, then*

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

Proof. By induction on the structure of M, as follows. The cases in the proof correspond closely to those in the definition of $[M]_g$.

Definition 5.1.38

$$
\begin{array}{rcll}
\sigma(G) & = & \vdash q_n, [\![G]\!]_g, g & q_n, g \ new \\
[\![(\forall X_{i+1} : G)]\!]_g & = & (q_{i+1}^{\perp} \otimes ((x_{i+1} \parr q_i) \& (x_{i+1}^{\perp} \parr q_i))), [\![G]\!]_g & q_{i+1} \ new \\
[\![(\exists X_{i+1} : G)]\!]_g & = & (q_{i+1}^{\perp} \otimes ((x_{i+1} \parr q_i) \oplus (x_{i+1}^{\perp} \parr q_i))), [\![G]\!]_g & q_{i+1} \ new \\
[\![M]\!]_g & = & (q_0^{\perp} \otimes [M]_g) & q_0 \ new \\
[X]_g & = & (x^{\perp} \otimes g) \oplus (x \otimes g^{\perp}) & \\
[\neg N]_g & = & \textsc{not}(a, g) \parr [N]_a & a \ new \\
[N \wedge P]_g & = & \left\{ \begin{array}{rl} \textsc{and}(a, b, g) & \parr \\ \textsc{copyall}(Var(N) \cap Var(P)) & \parr \\ [N]_a & \parr \\ [P]_b & \end{array} \right. & a, b \ new \\
[N \wedge P]_g & = & \textsc{and}(a, b, g) \parr [N]_a \parr [P]_b & a, b \ new \\
\textsc{copyall}(\overline{X}) & = & \parr_{X_i \in \overline{X}} \textsc{copy}(x_i) &
\end{array}
$$

The lower (simpler) definition of $[N \wedge P]_g$ is only applicable in the case that $Var(N) \cap Var(P) = \emptyset$. In the other case, where $Var(N) \cap Var(P) \neq \emptyset$, the COPYALL definition must be used.

Note that the sequent $\sigma(G)$ contains no MALL constants. The complexity of computing $\sigma(G)$ is at most quadratic in the size of G since the encoding function is defined inductively over the structure of the formula, and the intersection operation can be performed in linear time with a bit-vector representation of sets, where the length of each bit-vector is the number of distinct Boolean variables occurring in G. The cost of constructing the COPY formulas at each step in the recursion is also linear in the size of G. The cost of each NOT and AND formula is fixed with respect to the representation of the literals, and the literals can be represented with a cost that is logarithmic in the size of G.

The encoding may be computed in log-space, although the algorithm described above uses more than log-space, because of the work space required to save the set of variables that must be copied when encoding a conjunction. The encoding algorithm could be modified to make a number of passes over the input to determine the number of occurrences of each variable and generate the required number of COPY formulas.

matrix. $Var(M)$ is the set of variables occurring in the Boolean formula M. Overlined syntactic variables such as $\overline{X}$ and $\overline{Y}$ range over sets of Boolean variables.

The mall sequent encoding a QBF G is represented by $\sigma(G)$. We need to be careful about keeping literals distinct. The annotation "a new" in the definition indicates that the literal a is a freshly chosen one that has not been used elsewhere in the encoding.

The sequent $\sigma(G)$ consists of the encoding of the QBF $[\![G]\!]_g$, where g labels the output signal, the key q_n, and the output value g. The definition of $[\![G]\!]_g$ constructs the quantifier encodings by induction on the length of the quantifier prefix. The definition of $[M]_g$ is by induction on the structure of M, so that $[N \wedge P]_g$ is constructed by

- choosing the fresh labels a and b for the outputs of subformulas N and P, respectively
- defining the relation between a, b, and g by $\textsc{and}(a, b, g)$
- if needed, providing a copying formula for each Boolean variable common to both N and P
- and recursively constructing $[N]_a$ and $[P]_b$

To be precise, we provide the following definition of the encoding.

The idea here is that the quantifier encoding for $\exists X_1$ hides the "key" q_1 that is needed to unlock the quantifier encoding for $\forall X_2$. If we now attempt to violate the quantifier dependencies, the following would be one possible deduction.

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

In the above deduction, we are left with a subgoal of the form $\vdash q_1^\perp, q_1, x_1$, and since x_1 is not a constant, we cannot reduce this sequent to a MALL axiom. (Recall that MALL lacks an unrestricted weakening rule.) Other deductions attempting to violate the quantifier ordering also fail. On the other hand, the deduction which does respect the order of the quantifier encodings can be performed as shown below. The quantifier encoding for $\exists X_1$ provides the key q_1 for unlocking the quantifier encoding of $\forall X_2$.

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

The formal definition of the polynomial time encoding of QBF validity in terms of MALL provability is given in Section 5.1.5. In Section 5.1.6, we demonstrate the correctness of the encoding.

5.1.5 Formal Definition of the Encoding

For our purpose, a Boolean formula is constructed from Boolean variables using the Boolean connectives $\neg$ and $\wedge$. All quantified variables are assumed to occur in the

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

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash x_2^\perp, \Gamma & \vdash x_2, \Gamma\end{array}}{\vdash (x_2 \& x_2^\perp), \Gamma}\,\&$$

However, with this mapping of quantifiers, the MALL encoding of G and H would be identical and provable, but H is not a valid QBF.

A correct encoding of $\exists X_1 \forall X_2 : M$ should ensure that if the encoding is provable in MALL, then there is a proof in which the choice of a truth value for X_1 is independent of whether X_2 is T or F. The order of reductions below show how the choice of a truth value for $\exists X_1$ in a proof of the MALL encoding can depend on the quantifier $\forall X_2$.

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash x_1, x_2, \Gamma & \vdash x_1^\perp, x_2^\perp, \Gamma\end{array}}{\vdash (x_1 \oplus x_1^\perp), x_2, \Gamma}\,\oplus \qquad \dfrac{\vdots}{\vdash (x_1 \oplus x_1^\perp), x_2^\perp, \Gamma}\,\oplus}{\vdash (x_1 \oplus x_1^\perp), (x_2 \& x_2^\perp), \Gamma}\,\&$$

In this ordering of the reductions, $(x_1 \oplus x_1^\perp)$ is reduced differently on the x_2 and $x_2^\perp$ branches of the proof leading to distinct witnesses for X_1 according to whether X_2 is T or F. The solution to this quantifier order problem is to encode the quantifier dependencies in the MALL formula so that if there is any proof, then there is some proof of the encoding in which $(x_1 \oplus x_1^\perp)$ is reduced *below* $(x_2 \& x_2^\perp)$, thus ensuring that the truth value of X_1 has been chosen independently of the truth value for X_2. For this purpose, we introduce new MALL atoms q_0, q_1, q_2, and encode $\exists X_1 \forall X_2 : M$ as

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

be constructed.

$$\dfrac{\dfrac{\vdots}{\vdash x_1^\perp, x_2^\perp, \Gamma_1, c} \quad \dfrac{\vdots}{\vdash x_1^\perp, x_2^\perp, \Gamma_2, f}\otimes}{\dfrac{\dfrac{}{\vdash g, g^\perp}\mathbf{I} \quad \vdash x_1^\perp, x_1^\perp, x_2^\perp, x_2^\perp, (c \otimes f), \Gamma_1, \Gamma_2}\otimes}{\dfrac{\dfrac{}{\vdash x_2, x_2^\perp}\mathbf{I} \quad \vdash x_1^\perp, x_1^\perp, x_2^\perp, x_2^\perp, (c \otimes f \otimes g^\perp), \Gamma_1, \Gamma_2, g}\otimes}{\dfrac{\dfrac{}{\vdash x_1, x_1^\perp}\mathbf{I} \quad \vdash x_1^\perp, x_1^\perp, x_2^\perp, \mathrm{AND}(c, f, g), x_2 \otimes (x_2^\perp \parr x_2^\perp), \Gamma_1, \Gamma_2, g}\otimes}{\dfrac{\vdash x_1^\perp, x_2^\perp, \mathrm{AND}(c, f, g), x_1 \otimes (x_1^\perp \parr x_1^\perp), \mathrm{COPY}(x_2), \Gamma_1, \Gamma_2, g}{\vdash x_1^\perp, x_2^\perp, \mathrm{AND}(c, f, g), \mathrm{COPY}(x_1), \mathrm{COPY}(x_2), \Gamma_1, \Gamma_2, g}\oplus}{\vdash x_1^\perp, x_2^\perp, \mathrm{AND}(c, f, g) \parr \mathrm{COPY}(x_1) \parr \mathrm{COPY}(x_2) \parr \Gamma_1 \parr \Gamma_2, g}\parr$$

In summary, we have informally described the encoding in MALL of the evaluation of Boolean formulas under an assignment. The connectives $\parr$, $\otimes$, and $\oplus$ were used to represent the truth tables of $\neg$ and $\wedge$, and MALL literals were used to represent the "signals" in the Boolean formula. The duplication of input signals forms a crucial part of the encoding since MALL lacks a rule of contraction.

5.1.4 Encoding Boolean Quantification

Recall that G is the formula $\forall X_2 \exists X_1 : M$, and H is the formula $\exists X_1 \forall X_2 : M$, where M is $\neg(\neg X_1 \wedge X_2) \wedge \neg(\neg X_2 \wedge X_1)$. Intuitively, it is useful to separate the encoding of the Boolean quantifier prefix as separately encoding the individual quantifiers and the dependencies between quantifiers. Given the above encoding for assignments and Boolean formulas, an almost correct way to encode Boolean quantifiers would be to encode $\exists X_1$ as the formula $(x_1 \oplus x_1^\perp)$, and $\forall X_2$ as $(x_2 \& x_2^\perp)$. The encoding of G would then be given by the sequent

$$\vdash (x_2 \& x_2^\perp), (x_1 \oplus x_1^\perp), [M]_g, g.$$

The formula $(x_1 \oplus x_1^\perp)$ behaves like existential quantification in proof search since a nondeterministic choice can be made between

$$\dfrac{\dfrac{\vdots}{\vdash x_1^\perp, \Gamma}}{\vdash (x_1 \oplus x_1^\perp), \Gamma}\oplus \qquad \text{and} \qquad \dfrac{\dfrac{\vdots}{\vdash x_1, \Gamma}}{\vdash (x_1 \oplus x_1^\perp), \Gamma}\oplus$$

the formula

$$\textsc{and}(c, f, g) \parr \textsc{implies}(x_1, x_2, a, b, c) \parr \textsc{implies}(x_2, x_1, d, e, f).$$

The validity of M under the assignment $\{X_1 \leftarrow T, X_2 \leftarrow T\}$ would then be represented by

$$\vdash x_1^\perp, x_2^\perp, \textsc{and}(c, f, g) \parr \textsc{implies}(x_1, x_2, a, b, c) \parr \textsc{implies}(x_2, x_1, d, e, f), g. \qquad (5.7)$$

The following deduction represents one attempt to prove sequent (5.7).

$$\dfrac{\dfrac{\vdots \qquad\qquad \vdots}{\dfrac{\vdash x_1^\perp, x_2^\perp, \textsc{implies}(x_1, x_2, a, b, c), c \qquad \vdash \textsc{implies}(x_2, x_1, d, e, f), f}{\vdash x_1^\perp, x_2^\perp, (c \otimes f), \textsc{implies}(x_1, x_2, a, b, c), \textsc{implies}(x_2, x_1, d, e, f)}\otimes \qquad \dfrac{}{\vdash g^\perp, g}\mathbf{I}}{\dfrac{\vdash x_1^\perp, x_2^\perp, (c \otimes f \otimes g^\perp), \textsc{implies}(x_1, x_2, a, b, c), \textsc{implies}(x_2, x_1, d, e, f), g}{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c, f, g), \textsc{implies}(x_1, x_2, a, b, c), \textsc{implies}(x_2, x_1, d, e, f), g}\oplus}\otimes$$
$$\dfrac{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c, f, g), \textsc{implies}(x_1, x_2, a, b, c), \textsc{implies}(x_2, x_1, d, e, f), g}{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c, f, g) \parr \textsc{implies}(x_1, x_2, a, b, c) \parr \textsc{implies}(x_2, x_1, d, e, f), g}\parr$$

Since MALL lacks a rule of contraction, each of the assignment literals, $x_1^\perp$ and $x_2^\perp$, can appear in only one premise of a $\otimes$ rule. As a result, one of the remaining subgoals in the above deduction lacks the required input literals. We therefore need to be able to explicitly duplicate the assignment literals in the sequent (5.7) to match the number of duplicate occurrences of X_1 and X_2 in M. The formula $\textsc{copy}(x_1)$ defined as

$$(x_1 \otimes (x_1^\perp \parr x_1^\perp)) \oplus (x_1^\perp \otimes (x_1 \parr x_1))$$

serves to duplicate an instance of x_1 or $x_1^\perp$. If M is now encoded as

$$\textsc{and}(c, f, g) \parr \textsc{copy}(x_1) \parr \textsc{copy}(x_2) \parr \Gamma_1 \parr \Gamma_2$$

where Γ_1 abbreviates $\textsc{implies}(x_1, x_2, a, b, c)$, and Γ_2 abbreviates $\textsc{implies}(x_2, x_1, d, e, f)$, the desired deduction of (5.7) can then

$$\textsc{and}(x,y,b) = \left[\begin{array}{ll} (x \otimes y \otimes b^{\perp}) & \oplus \\ (x^{\perp} \otimes y^{\perp} \otimes b) & \oplus \\ (x \otimes y^{\perp} \otimes b) & \oplus \\ (x^{\perp} \otimes y \otimes b) & \end{array}\right] \tag{5.5}$$

Sequent (5.6) represents $X, Y \models (X \wedge Y)$:

$$\vdash x^{\perp}, y^{\perp}, \textsc{and}(x,y,b), b. \tag{5.6}$$

Sequent (5.6) has the proof

$$\dfrac{\dfrac{}{\vdash x^{\perp}, x}\mathbf{I} \qquad \dfrac{\dfrac{}{\vdash y^{\perp}, y}\mathbf{I} \qquad \dfrac{}{\vdash b^{\perp}, b}\mathbf{I}}{\vdash y^{\perp}, (y \otimes b^{\perp}), b}\otimes}{\dfrac{\vdash x^{\perp}, y^{\perp}, (x \otimes y \otimes b^{\perp}), b}{\vdash x^{\perp}, y^{\perp}, \textsc{and}(x,y,b), b}\oplus}\otimes$$

As with sequent (5.4), the MALL sequent representing the false assertion $\neg X, Y \models (X \wedge Y)$ is given by

$$\vdash x, y^{\perp}, \textsc{and}(x,y,b), b$$

and is not provable since it can be falsified by the classical interpretation assigning F to x and b, and T to y.

The next step is to construct the encoding of the Boolean formula M given at the beginning of this section, from the encodings of the Boolean connectives. The formula M is thought of as a Boolean circuit with the distinctly labeled signals. The encoding $[(\neg X_1 \wedge X_2)]_b$ is given by the formula $\textsc{and}(a, x_2, b) \parr \textsc{not}(x_1, a)$. Let $\textsc{implies}(x,y,u,v,w)$ represent the formula

$$\textsc{not}(v,w) \parr \textsc{and}(u,y,v) \parr \textsc{not}(x,u),$$

then $\textsc{implies}(x_1, x_2, a, b, c)$ is the encoding $[\neg(\neg X_1 \wedge X_2)]_c$. The literals a, b and c are the distinct literals labeling the output signals of the Boolean gates.

We now consider the problem that the input signals in M have a fanout greater than one. An almost correct encoding in MALL of the Boolean formula M is given by

For literals x and y, the definition of $\textsc{not}(x, y)$ is just the representation of the truth table for negation within MALL, as shown below:

$$\textsc{not}(x, y) = (x \otimes y) \oplus (x^{\perp} \otimes y^{\perp}). \tag{5.1}$$

$\textsc{not}(x_1, a)$ is simply the linear negation of the formula

$$(x_1 \multimap a^{\perp}) \& (x_1^{\perp} \multimap a)$$

which is more perspicuous in describing a as the Boolean negation of x_1. The sequent

$$\vdash x_1, \textsc{not}(x_1, a), a \tag{5.2}$$

encodes the situation where the input X_1 is F, and asserts (correctly) that the output $\neg X_1$ is T.

The sequent (5.2) is easily seen to have the MALLproof

$$\dfrac{\dfrac{\dfrac{}{\vdash x_1, x_1^{\perp}}\mathbf{I} \qquad \dfrac{}{\vdash a^{\perp}, a}\mathbf{I}}{\vdash x_1, (x_1^{\perp} \otimes a^{\perp}), a}\otimes}{\vdash x_1, (x_1 \otimes a) \oplus (x_1^{\perp} \otimes a^{\perp}), a}\oplus$$

Similarly, the sequent (5.3) representing $\{X_1 \leftarrow T\} \not\models \neg X_1$ is also provable.

$$\vdash x_1^{\perp}, \textsc{not}(x_1, a), a^{\perp}. \tag{5.3}$$

On the other hand, the sequent

$$\vdash x_1^{\perp}, \textsc{not}(x_1, a), a \tag{5.4}$$

asserts (falsely) that $\{X_1 \leftarrow T\} \models \neg X_1$. To see why sequent (5.4) is not provable, we observe that MALL is a refinement of classical logic in which no classically falsifiable sequents are provable. The sequent $\vdash x_1^{\perp}, \textsc{not}(x_1, a), a$ is falsified by assigning T to x_1 and F to a, while interpreting $\otimes$ and $\&$ as classical conjunction and $\oplus$ and $\parr$ as classical disjunction. A sequent is interpreted classically as the disjunction of the sequence of formulas that it contains.

The encoding for conjunction, $[X \wedge Y]_b$ is given by $\textsc{and}(x, y, b)$ as defined below.

$B_1, \ldots, B_n$ has the one-sided form $\vdash A_1^\perp, \ldots, A_m^\perp, B_1, \ldots, B_n$. Thus, a formula $A \multimap B$ on the left of a two-sided sequent becomes $A \otimes B^\perp$ in a one-sided sequent. Similarly, the provable two-sided sequent $A, A \multimap B \vdash B$ becomes $\vdash A^\perp, A \otimes B^\perp, B$. While one-sided sequents simplify the technical arguments considerably, the reader might gain further insight by rewriting parts of our encoding in a two-sided form.

5.1.3 Encoding Boolean Evaluation

The encoding of the Boolean connectives and quantifiers in MALL is described here by means of an example. The full definition of the encoding appears in Section 5.1.5. The encoding from QBF validity to MALL provability makes no use of the MALL constants. Consider the *valid* QBF G given by

$$\forall X_2 \exists X_1 : \neg(\neg X_1 \wedge X_2) \wedge \neg(\neg X_2 \wedge X_1).$$

The matrix M of G is essentially a restatement of $(X_1 \iff X_2)$. Let H be the falsifiable formula $\exists X_1 \forall X_2 : M$ that is obtained from G by reversing the order of the quantifiers. It is crucial that the encodings of G and H in MALL respect the ordering of quantifiers so that the encoding of G is provable but the encoding of H is not.

The encoding of the Boolean matrix describes the formula as a circuit with signals labeled by MALL literals. Let the assignment I be encoded by a sequence of MALL formulas $\langle I \rangle$, and $[M]_a$ be 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

$$\vdash \langle I \rangle, [M]_a, a^\perp.$$

Since we are using one-sided sequents, we encode the assignment $X_1, \neg X_2$ by $x_1^\perp, x_2$. The MALL literals encoding the assignment are to be seen as the input signals to the encoding of the Boolean formula.

We first consider the Boolean connectives $\neg$ and $\wedge$, then construct the full encoding of M. The encoding $[\neg X_1]_a$ of $\neg X_1$ with output labeled a is the formula $\mathrm{NOT}(x_1, a)$.

variables and negated Boolean variables. For example, the assignment $X_1, \neg X_2, X_3$ maps X_1 to T, X_2 to F, and X_3 to T. The assignment I, X assigns T to X, but behaves like I, otherwise. If I is an assignment for the free variables in G, we use the standard notation $I \models G$ to indicate that G is valid under I, and write $I \not\models G$ if I falsifies G. Note that

$$I \models \forall X\colon G \text{ iff } I, X \models G \qquad \text{and} \qquad I, \neg X \models G$$
$$I \models \exists X\colon G \text{ iff } I, X \models G \qquad \text{or} \qquad I, \neg X \models G$$

If G is a QBF and I is an assignment for the free variables in G, we say G *is valid under* I exactly if $I \models G$. If G is a closed QBF, then G is said to be valid if it is valid under the empty assignment. The validity of a closed QBF G is represented as $\models G$. The QBF validity problem is: *Given a closed QBF G, is G valid?*

We demonstrate the PSPACE-hardness of MALL provability by defining a succinct encoding of a QBF as a MALL sequent that is provable *exactly* when the given QBF is valid.

The transformation of the QBF validity problem to MALL provability takes place in two steps:

- Given a quantifier-free Boolean formula M and an assignment I for the free variables in M, we show that there is a MALL sequent encoding M and I which is provable exactly when M is valid under I. This essentially demonstrates that the process of evaluating Boolean functions can be represented by the process of cut-free proof search in the MALL sequent calculus.

- Given a QBF G and an assignment I for the free variables in G, there exists a MALL sequent encoding the quantifier prefix and the Boolean matrix of G so that the MALL sequent is provable exactly when G is valid under I. The idea here is to simulate the Boolean quantifiers $\exists$ and $\forall$ by using the additive connectives $\oplus$ and $\&$.

Two-sided vs. one-sided sequents. We use a formulation of MALL with one-sided sequents to simplify the proofs. In linear logic, a two-sided sequent $A_1, \ldots, A_m \vdash$

MALL proof tree is at most linear in the length of the final sequent of the proof. An alternating Turing machine [19] may guess and check a cut-free proof in linear time, using OR-branching to nondeterministically guess a reduction in the cut-free proof, and AND-branching to generate and check the proofs of both premises of a two premise rule in parallel.

Membership in PSPACE can also be proved without reference to alternation. A nondeterministic Turing machine can be defined to generate and check a cut-free sequent proof in a depth-first manner. Given the linear bound on the depth of any cut-free proof with respect to the size of the conclusion sequent, the search stack need contain no more than a linear number of sequents. Since each sequent in a cut-free proof is no larger than the conclusion sequent, we get a quadratic bound on the stack size. ■

5.1.2 Informal Outline of PSPACE-hardness of MALL

Since there are a number of technical details to the proof of PSPACE-hardness, we will illustrate the key intuitions by means of an example; the details of the proof are given in Subsection 5.1.6.

The PSPACE-hardness of MALL provability is demonstrated by a transformation from the validity problem for quantified Boolean formulas (QBF). A *quantified Boolean formula* has the (prenex) form $Q_m X_m \ldots Q_1 X_1 : M$, where

1. each Q_i is either $\forall$ or $\exists$,
2. M is a quantifier-free *Boolean matrix* containing only the connectives $\neg$ and $\wedge$, and *Boolean variables*.

A *closed* QBF contains no free variables. Our conventions in this section are that G and H range over quantified Boolean formulas; M and N range over quantifier-free Boolean formulas; U, V, X, Y, Z range over Boolean variables; and I ranges over truth value assignments. For expository convenience, we refer to quantifier-free Boolean formulas simply as Boolean formulas.

An *assignment* I for a set of Boolean variables $\{X_1, \ldots, X_n\}$ maps each X_i to a truth value from $\{T, F\}$. An assignment is represented by a sequence of Boolean

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

An important property of the sequent calculus formulation of MALL is cut-elimination. This property follows from Theorem 2.3.4, but since we are restricting our attention to the one sided case, we restate the theorem explicitly.

Theorem 5.1.36 *Any sequent provable in* MALL *is provable without the cut rule.*

Proof. Since MALL is a fragment of linear logic, we may use the cut-elimination procedure from Theorem 2.3.4 to convert a MALL proof to a cut-free proof in linear logic. By the subformula property (Theorem 2.4.5), such a cut-free proof of a MALL sequent contains only MALL formulas. Since all the rules which apply to MALL formulas are already in MALL, any cut-free proof of a MALL sequent must already be a MALL proof. ∎

Membership in PSPACE is straightforward, given cut elimination, but we include a short sketch to illustrate the importance of Theorem 5.1.36. The proof of PSPACE-hardness is more technical. Proof search in the cut-free sequent calculus is crucial to the proof. The primitive step in proof search is a ıreduction, namely the application of an inference rule to transform a sequent matching the conclusion of the rule to the collection of sequents given by the corresponding premises of the rule. A reduction is the inverse of an inference rule, and drives conclusions to premises. Proof search is the process of constructing a cut-free proof in a bottom-up manner by nondeterministically applying reductions starting from the conclusion sequent.

5.1.1 Membership in PSPACE

Theorem 5.1.37 *The provability in* MALL *of a given sequent can be decided by a polynomial-space bounded Turing machine.*

Proof. By Theorem 5.1.36, a provable MALL sequent has a cut-free MALL proof. In a cut-free MALL proof, there are at most two premises to each rule, and each premise is strictly smaller than the consequent. Therefore, the depth of a cut-free

Chapter 5

Decidable Fragments of Linear Logic

This chapter briefly covers decidable fragments of linear logic for which complexity results are known. The main results are that without exponentials or non-logical theory axioms, linear logic is PSPACE-complete, and the multiplicative fragment of linear logic is NP-complete.

5.1 MALL is PSPACE-complete

In this section, we analyze the complexity of the fragment of propositional linear logic without the modal storage operator ! and its dual ?, but including all the remaining connectives and constants of linear logic. Earlier it was shown that with the addition of nonlogical theories to MALL the decision problem is recursively unsolvable. Here we study the complexity of the decision problem in the absence of theory axioms.

In this section we restrict our attention to the right hand side of the sequent arrow $\vdash$. The results of this section immediately carry over to the two sided version of the calculus at the expense of greater case analysis at many steps in the following proofs. Technically, we must restate the **Cut** rule as follows, essentially incorporating the negation rule into the **Cut** rule.

replaced by **Cut2**. Somewhat more concretely, the following shows a deduction of a sequent which is not derivable in CL:

$$\dfrac{\dfrac{\dfrac{}{\vdash p_1, p_1^\perp}\mathbf{I} \quad \dfrac{}{\vdash p_2, p_2^\perp}\mathbf{I}}{\vdash p_1, (p_1^\perp \otimes p_2^\perp), p_2}\otimes \qquad \dfrac{}{\vdash p_1, p_1^\perp}\mathbf{I}}{\vdash (p_1^\perp \otimes p_2^\perp), p_1, p_2}\mathbf{Cut2}$$

Notice in the final conclusion that p_1 and p_2 have changed places, in a way impossible without the use of the **Cut2** rule in CL. Thus using **Cut2** we could prove any sequent which is provable in the commutative fragment of linear logic corresponding to CL. However, it would be impossible to prove some such sequents in CL without **Cut2**, and thus cut-elimination fails in this logic.

However, since there is a proof of a sequent in this logic if and only if there is a proof of that sequent in (commutative) linear logic, we may as well use linear logic, which does have a cut-elimination theorem.

4.7 Summary of Chapter

In this chapter the undecidability of a small fragment of propositional noncommutative linear logic was demonstrated. This logic contains fewer connectives than those required for the proof of undecidability of (commutative) linear logic. The proof uses semi-Thue systems, although perhaps a more intuitive proof could use standard Turing machines, viewing the left of the turnstile as a direct representation of the Turing tape. The instructions must be reusable, and so are marked by !, and here we assume that all three structural rules (exchange, contraction, and weakening) are allowed for ! formulas.

The interest in this logic stems from work by Lambek [55] predating the introduction of linear logic. Lambek's work considered one of the possible variants of noncommutative linear logic (and did not contain exponentials, and so is decidable). The later sections of this chapter consider some other alternative formulations of noncommutative linear logic. Although these proof-theoretic investigations are suggestive, perhaps the best source of insight into the correct axiomatization would be semantic considerations and applications such as Lambek's.

full linear logic suffices to eliminate cuts from CL with unrestricted exchange. Cut-elimination for CL with $\otimes$ replaced by $\otimes\mathbf{3}$ is possible to prove directly, although the principal $\otimes\mathbf{3}$ versus $\parr$ case is quite difficult. Cut-elimination in this case may be accomplished with the addition of an "intermingling cut" rule which along with the nonintermingling cut rule may be eliminated from any proof. The key reason this lemma holds is that $\otimes$ and $\parr$ are the only binary connectives of CL and allowing $(A \otimes B)$ to be equivalent to $(B \otimes A)$ in this context causes $(A \parr B)$ to be equivalent to $(B \parr A)$.

Corollary 4.6.35 *A sequent $\vdash \Gamma$ is provable in the system obtained by replacing $\otimes$ by $\otimes\mathbf{3}$ in CL augmented with additives and constants if and only if that sequent is provable in the system obtained by adding the unrestricted exchange rule to CL augmented with additives and constants.*

This corollary follows from the fact that the constants and additive connectives are inherently commutative, and may be proven by induction on the length of proofs.

4.6.2 Intermingling Cut

A problem similar to that which occurs with $\otimes\mathbf{3}$ arises if we allow the **Cut** rule to interleave its conclusion. Define **Cut2**:

$$\textbf{Cut2}\quad \frac{\vdash \Sigma, A \qquad \vdash \Gamma, A^{\perp}}{\vdash \Delta} \quad \text{where } \Delta \text{ is some interleaving of } \Sigma \text{ and } \Gamma$$

As for the previous alteration, the system CL with **Cut** replaced by **Cut2** would be commutative. We can achieve the effect of the unrestricted exchange rule using the **Cut2** rule:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \vdash \Sigma, \Gamma, A & \vdash A, A^{\perp}\end{array}}{\vdash \Sigma, A, \Gamma}\textbf{Cut2}$$

Note that for any formula A, there is always a proof of $\vdash A, A^{\perp}$ in noncommutative (as well as commutative) linear logic. The above partial deduction shows that unrestricted exchange may be simulated in noncommutative linear logic with **Cut**

4.5.5 Mix and Match

The above modifications of CL do not interfere with cut-elimination, nor with the basic undecidability result for CL. It is also the case that even in combination the above three modifications, (**R** versus embedding, ?**E** versus ?**C2**, and $\otimes$ versus $\otimes$**2**) do not interact. That is, any combination of these modifications retains the character of CL, including the properties of being undecidable, and having a cut-elimination theorem.

4.6 Degenerate Noncommutative Linear Logics

Some variations on the CL system are not as benign as the above. In fact, it is much easier to create nonsense than a coherent logic by altering proof rules haphazardly.

The main focus of this section is to consider plausible but degenerate variants of the rules based on interleaving the circular orders of hypotheses.

4.6.1 Intermingling $\otimes$

At first glance, it might seem interesting to study the systems obtained when binary rules in CL are replaced with rules which allow intermingling of the hypotheses in the conclusion. For example,

$$\otimes\mathbf{3}\quad \frac{\vdash \Sigma, A \qquad \vdash B, \Gamma}{\vdash \Delta, (A \otimes B)} \quad \text{where } \Delta \text{ is some interleaving of } \Sigma \text{ and } \Gamma$$

Somewhat surprisingly, the system obtained by replacing $\otimes$ with $\otimes$**3** in CL is equivalent to a commutative version of CL.

Lemma 4.6.34 *A sequent $\vdash \Gamma$ is provable in the system obtained by replacing $\otimes$ by $\otimes$**3** in CL if and only if that sequent is provable in the system obtained by adding the unrestricted exchange rule to CL.*

This lemma follows by induction on the length of cut-free proofs. Formally, we need a cut-elimination procedure for both logics. The cut-elimination procedure for

Corollary 4.5.30 *The provability problem for CL without the* **?E** *rule, and with the* **?C** *rule replaced by* **?C2** *rule is recursively unsolvable.*

4.5.4 Alternate $\otimes$

There are two quite reasonable versions of the $\otimes$ rule in noncommutative linear logic, one as used above in CL, and the other using a different sequent order in the conclusion:

$$\otimes \quad \frac{\vdash \Sigma, A \qquad \vdash B, \Gamma}{\vdash \Sigma, (A \otimes B), \Gamma} \qquad\qquad \frac{\vdash \Sigma, A \qquad \vdash \Gamma, B}{\vdash \Sigma, \Gamma, (A \otimes B)} \quad \otimes\mathbf{2}$$

The two formulations are equivalent in the presence of unrestricted exchange (commutative linear logic), but are subtlely different in the context of noncommutative linear logic. In a noncommutative linear logic with $\otimes$ replaced by $\otimes\mathbf{2}$, the definition of negation must change. In particular, the negation of the multiplicative connectives would be defined as follows:

$$(A \otimes B)^{\perp} \triangleq A^{\perp} \parr B^{\perp} \qquad (A \parr B)^{\perp} \triangleq A^{\perp} \otimes B^{\perp}$$

We define the translation $\sigma(\Gamma)$ of a sequent Γ to be the sequent Γ with all occurrences of formulas of the form $A \otimes B$ replaced by $B \otimes A$.

Lemma 4.5.31 *A sequent* $\vdash \Gamma$ *is provable in CL if and only if* $\sigma(\Gamma)$ *is provable in CL with the* $\otimes$ *rule replaced by the* $\otimes\mathbf{2}$ *rule*

Lemma 4.5.32 *A sequent* $\vdash \sigma(\Gamma)$ *is provable in CL if and only if* Γ *is provable in CL with the* $\otimes$ *rule replaced by the* $\otimes\mathbf{2}$ *rule*

This lemma follows by induction on the height of proofs.

Lemma 4.5.33 *The provability problem for CL with the* $\otimes$ *rule replaced by the* $\otimes\mathbf{2}$ *rule and alternate definition of negation is recursively unsolvable.*

This lemma follows from the above two, which simply state that by reversing the order of all tensor ($\otimes$) formulas, we pass from CL to this new logic, and back again. Thus a decision procedure for one implies a decision procedure for the other, and by Lemma 4.4.24, we know there is no decision procedure for CL.

the system derived from CL by removing the **R** rule, and replacing all other rules by their embedded equivalents, and adding a symmetric identity rule.

Lemma 4.5.27 *A sequent* $\vdash \Gamma$ *is provable in CL if and only if it is provable in* E*CL.*

This lemma follows by induction on the length of proofs, and from this lemma we obtain undecidability for this system.

Corollary 4.5.28 *The provability problem for CL without the* **R** *rule, and with every other rule replaced by its embedded equivalent is recursively unsolvable.*

4.5.3 CL without ?E

The earlier proof of undecidability fails in CL without the ?**E** rule, since some of the requisite lemmas about theories fail. However, we may omit this rule, and replace **?C** with the following **?C2** rule to restore our results, and many other properties of noncommutative linear logic.

$$\textbf{?C}\quad \frac{\vdash \Sigma, ?A, ?A}{\vdash \Sigma, ?A} \qquad\qquad \frac{\vdash \Sigma, ?A, \Gamma, ?A}{\vdash \Sigma, \Gamma, ?A}\quad \textbf{?C2}$$

This contraction rule essentially states that what may be proven from two *not necessarily contiguous* assumptions of a reusable formula, may be proven from one assumption of that reusable formula. It is the case that the **?C2** rule is derivable from the **?E** and **?C** rules in CL.

Lemma 4.5.29 *A sequent* $\vdash \Gamma$ *is provable in CL if and only if it is provable in CL without the* **?E** *rule, and with the* **?C** *rule replaced by* **?C2**.

This lemma may be proven by induction on the length of proofs. Essentially, in CL one may contract and then exchange the reusable formula to any desired position, while in the other system one may contract the formula directly into position. On the other hand, to permute a reusable formula in CL, one simply applies exchange, while in the other system one must contract the formula into position, and then weaken away the formula in its previous position. Using this lemma, we may obtain the following undecidability result.

These rules contain the two novel rules of rotation (**R**) and restricted exchange (**!E**). This system is very close to that studied by Yetter [97], although Yetter's extra modalities k and K have been omitted.

The one-sided calculus CL is a conservative extension over the two-sided intuitionistic calculus FICL. In order to make this precise, we require some definitions. The notation $[\Gamma]^{\perp}$ stands for the negation of a sequence of formulas, which is defined to the sequence of negations of formulas in Γ, in reverse order.

Lemma 4.5.26 *Any well-formed FICL sequent $\Gamma \vdash C$, is provable in FICL if and only if $\vdash [\Gamma]^{\perp}, C$ is provable in CL.*

A broader class of conservativity results are available, going beyond the small fragment FICL, but the logic CL is not conservative over the whole two-sided intuitionistic logic ICL.

In all formulations of noncommutative linear logic the key rule is $\otimes$. In a sense which we make more precise later, the constants and additive connectives of linear logic are inherently commutative. Also, the $\parr$ rule follows the $\otimes$ rule in its commutativity. Thus noncommutative linear logic is quite sensitive to the exact formulation of $\otimes$. However, there are some minor variations on the syntactic presentation of the other proof rules which first bear some notice.

4.5.2 Rotate Rule versus Embedding

Without the **R** rule we would have to modify the formulation of other rules, such as the $\parr$ rule, to allow its application within a sequent, instead of requiring its application at one end of a sequent. To see this, compare the original version of $\parr$ on the left with the modified version on the right below:

$$\parr \quad \frac{\vdash \Sigma, A, B}{\vdash \Sigma, (A \parr B)} \qquad \frac{\vdash \Sigma, A, B, \Gamma}{\vdash \Sigma, (A \parr B), \Gamma} \quad \parr\mathbf{2}$$

We will call the $\parr\mathbf{2}$ rule the embedded equivalent of the $\parr$ rule. The use of $\parr\mathbf{2}$ in noncommutative linear logic without the **R** rule directly corresponds to the use of $\parr$ in CL with **R** rule considered part of the system. We will use ECL to stand for

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\mathbf{R} \qquad \frac{\vdash \Gamma, A}{\vdash A, \Gamma}$$

$$!\mathbf{E} \qquad \frac{\vdash \Gamma, ?A, \Sigma}{\vdash \Gamma, \Sigma, ?A}$$

Theorem 4.4.24 *The provability problem for FICL is recursively unsolvable.*

Corollary 4.4.25 *The provability problem for ICL is recursively unsolvable.*

This corollary follows from Theorem 4.4.24 by a conservativity result which is easily derived from the cut-elimination and subformula properties of FICL.

4.5 Other Noncommutative Logics

As mentioned previously, there is a family of logics which share a strong resemblance to FICL. All of the ones we can sensibly imagine have undecidable decision problems.

The first generalization that one might make is to loosen the intuitionistic restriction of FICL, allowing multiple formulas to appear on the right hand side of a sequent.

We will now consider various possibilities of formalizations of such a system, where all formulas appear on the right of the $\vdash$. When specific rules are mentioned, it is their use on the right that is meant. For example, in the remainder of this chapter, the $\otimes$ **R** rule will be referred to as the $\otimes$ rule.

4.5.1 One-sided Noncommutative Linear Logic: CL

The one-sided rules of noncommutative linear logic (CL are given below.

Thus, by induction, given a sequence of reductions which solves a word problem, we may simulate the solution in FICL. ∎

The following concrete example illustrates the intended simulation. Assume that the first rule applied in the sequence of reductions is $\langle cd \rightarrow xy \rangle$. Then g in the above schema is cd, and g' is xy. Also, $[g]$ is p_c, p_d, $[g']$ is p_x, p_y, $[g']^2$ is $(p_x \otimes p_y)$, and $\gamma([g'])$ is $(p_x \otimes p_y)$. Also assume that r is $p_a p_b$, s is $p_e p_f$, and $[V]$ is p_n.

$$\dfrac{\dfrac{}{p_c, p_d \vdash (p_y \otimes p_x)}\mathbf{T} \quad \dfrac{\vdots}{\dfrac{p_a, p_b, p_x, p_y, p_e, p_f \vdash p_n}{p_a, p_b, (p_x \otimes p_y), p_e, p_f \vdash p_n}\otimes\mathbf{L}}}{p_a, p_b, p_c, p_d, p_e, p_f \vdash p_n}\mathbf{Cut}$$

Lemma 4.4.23 *The word problem P is solvable with productions $\mathcal{T}$ if $\tau(P)$ is provable in the theory derived from $\mathcal{T}$.*

Proof. We prove this lemma by induction on the size of the normalized directed proof of $\tau(P)$.

Inspecting the rules of FICL, we see that most rules are inapplicable, since the conclusion sequent, which is a translation $\tau(P)$ of a word problem, contains only occurrences propositions.

In the case that $\Gamma \vdash C$ is equal to $[V] \vdash [V]$, for some V, (Recall that the target word V of the word problem is a singleton — this proof may be a single application of the identity rule) then the solution to the word problem is trivial, i.e., no productions are required, since $U \equiv V$.

In the induction case where $\Gamma \vdash C$ is not proven by identity, the only remaining case is **Cut**. By cut-standardization, we know that one hypothesis is an axiom, and the cut-formula in that axiom is not a negative literal. Inspecting the axioms in the theory corresponding to the productions of a semi-Thue system, we see that the cut-formula must always be a formula $[g']^2$. This formula is built up from positive literals connected by $\otimes$. By the normalization property, we know that we may now apply $\otimes$ **L** until we are left again with the inductive case of some $\tau(P')$.

Thus the semi-Thue system may mimic the FICL proof by applying the production corresponding to the theory axiom applied, and we have our result by induction. ∎

this problem into a FICL sequent as:

$$[U] \vdash [V]^2$$

Now, we show that the word problem P is solvable within system $\mathcal{T}$ if and only if the translation $\tau(P)$ is provable in the theory derived from $\mathcal{T}$. We state the two parts of the equivalence as Lemmas 4.4.22 and 4.4.23.

Lemma 4.4.22 *The word problem P is solvable in the semi-Thue system $\mathcal{T}$ only if $\tau(P)$ is provable in the theory derived from $\mathcal{T}$.*

Proof. We proceed by induction on the length of the derivation of $U \Longrightarrow^* V$. If the derivation is trivial, that is $U \equiv V$, then we must show that the sequent

$$\vdash [V], [V]^\perp$$

is provable. Since we assume the word problem has the restricted form with V a singleton, this sequent actually has the form $\vdash [n], [n]^\perp$, which by definition of notation is $\vdash p_n^\perp, p_n$. This is provable by identity.

Suppose the derivation of $U \Longrightarrow^* V$ is a nonempty sequence:

$$U \Longrightarrow U_1 \Longrightarrow U_2 \Longrightarrow \cdots \Longrightarrow U_n \Longrightarrow V.$$

Since $U \Longrightarrow U_1$, there is some rule $\langle g \to g' \rangle$ in $\mathcal{T}$, and possibly null words r and s such that $[U] = [rgs]$ and $[U_1] = [rg's]$. By induction we assume that we have a proof of $[U_1] \vdash [V]$, and construct from this a proof of the following form:

$$\dfrac{\dfrac{[g] \vdash [g']^{\mathbf{T}} \qquad \dfrac{\dfrac{\vdots}{[r], [g'], [s] \vdash [V]}}{\dfrac{[r], \gamma([g']), [s] \vdash [V]}{\vdots}\otimes\mathbf{L}}\otimes\mathbf{L}}{[r], [g'], [s] \vdash [V]}\,\mathbf{Cut}$$

In this partial deduction there are as many applications of the $\otimes$ **L** rule as there are separate formulas in $[g']$.

is known to be undecidable [80]. The problem remains undecidable if we add the condition that V be a singleton (word of length one) n such that n does not occur in U or in the right hand side of any production, and only appears as a singleton on the left hand side of productions. This restriction is analogous to requiring that a Turing machine have a unique final state without any outgoing transitions.

To show that the above restrictions preserve undecidability it suffices to give a transformation from a general word problem to a word problem of the restricted class. Specifically, given a word problem from U to V, with set of productions $\mathcal{T}$, we may add the production $V \to n$ where n is a new symbol which is added to the alphabet Σ. We then ask the new word problem from U to n (with the new and the original productions), in the alphabet $\Sigma \cup \{n\}$. This problem is solvable if and only if the original problem is. However the new problem is of the restricted class defined above.

4.4 From Semi-Thue Systems to Noncommutative Linear Logic

We overload the definition of translation [ ] to include the case of words — the translation of a word $[ab\cdots z]$ is the list of FICL formulas $p_a, p_b, \cdots, p_z$. We also define $[ab\cdots z]^2$ to be the FICL formula $p_a \otimes p_b, \cdots \otimes p_z$. Finally, as a notational convenience, we let $\gamma(\Gamma)$ designate ambiguously any formula which could be derived from Γ by applications of the $\otimes$ **L** rule. In other words, $\gamma(\Gamma)$ is the result of replacing some number of commas separating formulas in Γ by $\otimes$.

Given a Semi-Thue system $\mathcal{T} = \{\langle a_1 \to b_1\rangle, \langle a_2 \to b_2\rangle \cdots \langle a_k \to b_k\rangle\}$, we define the FICL theory derived from $\mathcal{T}$ as the following set of sequents:

$$\begin{array}{c} [a_1] \vdash [b_1]^2 \\ [a_2] \vdash [b_2]^2 \\ \vdots \\ [a_k] \vdash [b_k]^2 \end{array}$$

For a word problem P consisting of the pair U, V we define the translation $\tau(P)$ of

For each axiom $t_i \equiv A_1, A_2, \cdots, A_n \vdash B$, we may prove $\vdash [t_i] \equiv \vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B)$ by several applications of $\otimes$ **L**, one application of $\multimap$ **R**, and one application of **!S**, as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A_1, A_2, \cdots, A_n \vdash B}T}{\otimes L}}{\vdots}}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B}\otimes L}{\vdash (A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B}\multimap R$$
$$\dfrac{}{\vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B)}!S$$

By cutting this proof against the given proof of $\Gamma, [T] \vdash C$, we obtain a proof of $\Gamma, [T - \{t_i\}] \vdash C$, where $T - \{t_i\}$ is the multiset difference of T and $\{t_i\}$. Thus by induction on the number of axioms, cutting each one out in turn, we can derive $\Gamma \vdash C$ in theory T. ∎

4.3.2 Semi-Thue Systems

A *semi-Thue system* $\mathcal{T}$ over alphabet Σ is a set of pairs $\langle x \to y \rangle$, where x and y are strings over Σ. Each pair in $\mathcal{T}$ is called a *production*, and we use them as rewrite rules. We call x the *left hand side* and y the *right hand side* of a production $\langle x \to y \rangle$. U *rewrites* to V in system $\mathcal{T}$ with a production $\langle g \to g' \rangle$ if U and V are words over alphabet Σ, and there exist possibly null words r and s such that $U = rgs$ and $V = rg's$. We write

$$U \Longrightarrow V$$

if U rewrites to V using some production. We use the notation

$$U \Longrightarrow^* V$$

if there exists a (possibly empty) sequence of words $U_1, \cdots, U_n$ such that

$$U \Longrightarrow U_1 \Longrightarrow U_2 \Longrightarrow \cdots \Longrightarrow U_n \Longrightarrow V$$

The *word problem* for a semi-Thue system $\mathcal{T}$ the problem of determining, for a given pair of words U and V, whether or not $U \Longrightarrow^* V$ in system $\mathcal{T}$. This problem

all the $[t_i]$ formulas:

$$\frac{}{p_i \vdash p_i}I \quad \Longrightarrow \quad \cfrac{\cfrac{\cfrac{\cfrac{}{p_i \vdash p_i}I}{p_i, [t_1] \vdash p_i}!W}{p_i, [t_1], [t_2] \vdash p_i}!W \\ \vdots \\ \cfrac{\cfrac{}{p_i, [t_1], [t_2], \cdots, [t_{k-1}] \vdash p_i}!W}{p_i, [T] \vdash p_i}!W}$$

We then continue by adding $[T]$ to every sequent in the entire proof tree. At every application of $\otimes R$, $-\!\circ L$, and **Cut**, we extend the proof tree with an extra copy of the conclusion sequent of the binary rule, to which we add an extra copy of $[T]$. Then we extend the proof further, adding one contraction step for each $[t_i]$ between that sequent and the original conclusion of that binary rule.

$$\cfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta \vdash A & \Gamma \vdash B\end{array}}{\Theta, \Gamma \vdash (A \otimes B)}\otimes \quad \Longrightarrow \quad \cfrac{\cfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta, [T] \vdash A & \Gamma, [T] \vdash B\end{array}}{\Theta, [T], \Gamma, [T] \vdash (A \otimes B)}\otimes}{\Theta, \Gamma, [T] \vdash (A \otimes B)}!C$$

Thus we have given a construction which builds a proof of $\Theta, [T] \vdash \Sigma$ without any nonlogical axioms from a given proof of $\Theta \vdash \Sigma$ using axioms from T. ∎

The following theorem closely corresponds to Theorem 2.7.9.

Theorem 4.3.21 (Theory $\Leftarrow$) *For any finite set of axioms T, $\Gamma \vdash C$ is provable in theory T if $\Gamma, [T] \vdash C$ is provable without nonlogical axioms.*

Proof. Assuming we have a proof of $\Gamma, [T] \vdash C$, we immediately have a proof of $\Gamma, [T] \vdash C$ in theory T, since any proof in pure linear logic is also a proof in the logic extended with axioms. We can also build proofs of $\vdash [t_i]$ in the theory T for each axiom t_i. By cutting these proofs against the given proof of $\Gamma[T] \vdash C$, we obtain a proof of $\Gamma \vdash C$ in theory T.

Theorem 4.3.20 (Theory $\Rightarrow$) *For any finite set of FICL axioms T, $\Gamma \vdash C$ is provable in theory T only if $\Gamma, [T] \vdash C$ is provable in FICL without nonlogical axioms.*

Proof. Given some proof of $\Gamma \vdash C$ in theory T, we have a linear logic proof tree with axioms of T at some leaves. For each leaf of the proof tree of the form $\Theta \vdash D$, where $\Theta \vdash D$ is some axiom t_i, we replace that leaf with a small proof of $\Theta, [T] \vdash D$ which does not use theory axioms (see below). For each leaf sequent which was originally an application of identity we extend the proof by adding weakenings for all the $![t_i]$ formulas. We then add $[T]$ to every sequent in the body of the entire proof tree. At every application of $\otimes R$, $-\circ L$, and **Cut**, we extend the proof tree with contractions on each formula in $[T]$.

The first mentioned proof tree, of $\Theta, [T] \vdash D$, will be constructed from the proof tree for $\Theta, [t_i] \vdash D$. Since each formula in $[T]$ begins with !, we may weaken in the remainder of $[T]$, and thus with some number of weakening steps we have $\Theta, [T] \vdash D$. For example, if there are k axioms, and $\Theta \vdash D$ is the axiom $t_1 = q_1 \vdash (q_2 \otimes a)$, then we know $[t_1] = !(q_1 -\circ (q_2 \otimes a))$. We then perform the following transformation:

$$\frac{}{q_1 \vdash (q_2 \otimes a)}{\scriptstyle T} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q_2 \vdash q_2}{\scriptstyle I} \quad \dfrac{}{a \vdash a}{\scriptstyle I}}{q_2, a \vdash (q_2 \otimes a)}{\scriptstyle \otimes R}}{(q_2 \otimes a) \vdash (q_2 \otimes a)}{\scriptstyle \otimes L}}{q_1 \vdash q_1}{\scriptstyle I} \quad \dfrac{}{q_1, (q_1 -\circ (q_2 \otimes a)) \vdash (q_2 \otimes a)}{\scriptstyle -\circ L}}{q_1, [t_1] \vdash (q_2 \otimes a)}{\scriptstyle !D}}{q_1, [t_1], [t_2] \vdash (q_2 \otimes a)}{\scriptstyle !W}}{\vdots}}{q_1, [t_1], [t_2], \cdots, [t_{k-1}] \vdash (q_2 \otimes a)}{\scriptstyle !W}}{q_1, [T] \vdash (q_2 \otimes a)}{\scriptstyle !W}$$

For each leaf sequent which was originally an application of identity we weaken in

premise of a directed cut where the cut-formula in that axiom is not a negative literal a *principal axiom* of that directed cut. By definition, all directed cuts have at least one principal axiom. A cut between two axioms is always directed, and if the cut-formula of such a cut is non-atomic, that cut has two principal axioms. A *directed* or *standardized* proof is a proof with all cuts directed.

Lemma 4.3.18 (FICL Cut Standardization) *If there is a proof of* $\Gamma \vdash C$ *in theory* T *in FICL, then there is a directed proof of* $\Gamma \vdash C$ *in theory* T *in FICL.*

This lemma may be proven in nearly the same way as cut-standardization in full linear logic (Lemma 2.7.10). In fact there are fewer cases here since the constants and additive connectives are not present in FICL.

An $\otimes$ **L** normalized directed proof is a directed proof where each application of $\otimes$ **L** has been permuted as far down the proof tree as possible.

Lemma 4.3.19 ($\otimes$ Normalization) *If there is a proof of* $\Gamma \vdash C$ *in theory* T *in FICL, then there is a* $\otimes$ **L** *normalized directed proof of* $\Gamma \vdash C$ *in theory* T *in FICL.*

This lemma may be proven by appealing to Lemma 4.3.18 and a permutability lemma, which is not proven here, but which is directly analogous to the permutability of $\otimes$ **L** (or $\parr$ **R**) down in a sequent proof (see Section 2.6).

4.3.1 Theories into FICL

It happens that the translation of the theories into pure (commutative) linear logic may also be employed to encode FICL theories in pure FICL. We recall the definition of the translation $[T]$ of a theory T with k axioms into a multiset of formulas by

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

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

$$[\, G_1, G_2, \cdots, G_n \vdash F_1, F_2, \cdots, F_m \,] \triangleq\ !((G_1 \otimes G_2 \otimes \cdots \otimes G_n) \multimap (F_1 \parr F_2 \parr \cdots \parr F_m))$$

The following theorem is directly analogous to Theorem 2.7.8, but is restated and proved here for the FICL case.

sort of and-branching. This situation is analogous to that for commutative versus noncommutative semi-Thue systems, where the noncommutative version allows the encoding of a zero test leading to undecidability, whereas the commutative version is unable to simulate zero test and has been shown to be decidable [52]. In fact, since FICL closely resembles semi-Thue systems, we will demonstrate undecidability of FICL by a reduction from semi-Thue systems.

Although the reduction is intuitively simple, the proof of its correctness requires some elaborate machinery. In particular, a cut-elimination theorem is required.

Lemma 4.2.16 (Cut Elimination Revisited) *If there is a proof of sequent* $\Gamma \vdash C$ *in FICL, then there is a cut-free proof of* $\Gamma \vdash C$ *in FICL.*

This lemma may be proven in the same manner as in the full logic 2.3.4.

Corollary 4.2.17 (Subformula Property Revisited) *Any formula in any cut-free FICL proof of* $\Gamma \vdash C$ *is a subformula of* Γ *or of* C.

4.3 FICL Theories

We will define theories as for the commutative case (see Section 2.7), and show that cut-standardization (see Lemma 2.7.10) again holds in this logic.

Formally, a *FICL axiom* may be any FICL sequent of the form $p_{i_1}, p_{i_2}, \cdots p_{i_n} \vdash C$, where C is any FICL formula not including modal operators (? or !), and the remainder of the sequent is made up of negative literals. Any finite set of FICL axioms is a *FICL theory*. For any theory T, we say that a sequent $\Gamma \vdash C$ is provable in T exactly when we are able to derive $\Gamma \vdash C$ using the standard set of FICL proof rules and FICL axioms from T. Thus each axiom of T is treated as a reusable sequent which may occur as a leaf of a proof tree. As before we will write $\overline{\Gamma \vdash C}^{\mathbf{T}}$ for a leaf sequent which is a member of the theory T.

We recall the definition of a *directed* cut:

A *directed* cut is one where at least one premise is an axiom in T, and the cut-formula in the axiom is not an atomic literal of negative polarity. We call any axiom

The $\forall$ **Right** and $\exists$ **Left** rules only apply if y is not free in Γ, and any nonlogical theory axioms.

Although ICL is undecidable, a more intriguing result is that a small fragment of ICL is also undecidable. We call this multiplicative-exponential fragment FICL.

$$\textbf{Identity}\quad \frac{}{A \vdash A} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2, A, \Gamma_3 \vdash C}{\Gamma_2, \Gamma_1, \Gamma_3 \vdash C}\quad \textbf{Cut}$$

$$\otimes\ \textbf{Left}\quad \frac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma, (A \otimes B), \Gamma_2 \vdash C} \qquad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash (A \otimes B)}\quad \otimes\ \textbf{Right}$$

$$\multimap\ \textbf{Left}\quad \frac{\Gamma_1 \vdash A \qquad B, \Gamma_2 \vdash C}{\Gamma_1, (A \multimap B), \Gamma_2 \vdash C} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash (A \multimap B)}\quad \multimap\ \textbf{Right}$$

$$!\ \textbf{W}\quad \frac{\Gamma_1, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C} \qquad \frac{\Gamma_1, !A, !A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}\quad !\ \textbf{C}$$

$$!\ \textbf{D}\quad \frac{\Gamma_1, A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\quad !\ \textbf{S}$$

$$!\ \textbf{E1}\quad \frac{\Gamma_1, A, !B, \Gamma_2 \vdash C}{\Gamma_1, !B, A, \Gamma_2 \vdash C} \qquad \frac{\Gamma_1, !B, A, \Gamma_2 \vdash C}{\Gamma_1, A, !B, \Gamma_2 \vdash C}\quad !\ \textbf{E2}$$

However, one may simplify this system further, by considering the logic with nonlogical theories, but without the explicit modal operators ! and ?. One may encode such a logic in FICL using the ! and ?.

4.2 FICL is Undecidable

We will show the word problem for semi-Thue systems has a straightforward encoding in FICL. Since we have already shown that full linear logic is undecidable, the fact that full noncommutative linear logic is undecidable is not too surprising. But since FICL is a fragment of noncommutative linear logic which does not contain the additive connectives, the earlier construction of and-branching two-counter machines in full linear logic would fail in FICL. However, the and-branching used in that construction was required in order to encode zero-test in a commutative setting. In a noncommutative setting a zero test operation may be encoded easily without any

Identity	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A \qquad \Gamma_2, A, \Gamma_3 \vdash C}{\Gamma_2, \Gamma_1, \Gamma_3 \vdash C}$	**Cut**
$\otimes$ **Left**	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma, (A \otimes B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash (A \otimes B)}$	$\otimes$ **Right**
$\multimap$ **Left**	$\dfrac{\Gamma_1 \vdash A \qquad B, \Gamma_2 \vdash C}{\Gamma_1, (A \multimap B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash (A \multimap B)}$	$\multimap$ **Right**
$\circ\!\!-$ **Left**	$\dfrac{\Gamma_2 \vdash B \qquad \Gamma_1, A \vdash C}{\Gamma_1, (A \circ\!\!- B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash (B \circ\!\!- A)}$	$\multimap$ **Right**
$\oplus$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	$\&$ **Right**
$\&$ **Left1**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right1**
$\&$ **Left2**	$\dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right2**
! W	$\dfrac{\Gamma_1, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1, !A, !A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	**! C**
! D	$\dfrac{\Gamma_1, A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	$\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}$	**! S**
? D	$\dfrac{\Gamma \vdash A}{\Gamma \vdash ?A}$	$\dfrac{!\Gamma, A \vdash ?C}{!\Gamma, ?A \vdash ?C}$	**? S**
? W	$\dfrac{\Gamma \vdash}{\Gamma \vdash ?A}$		
! E1	$\dfrac{\Gamma_1, A, !B, \Gamma_2 \vdash C}{\Gamma_1, !B, A, \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1, !B, A, \Gamma_2 \vdash C}{\Gamma_1, A, !B, \Gamma_2 \vdash C}$	**! E2**
0 **Left**	$\Gamma, 0 \vdash C$	$\Gamma \vdash \top$	$\top$ **Right**
1 **Left**	$\dfrac{\Gamma \vdash C}{\Gamma, 1 \vdash C}$	$\vdash 1$	1 **Right**

4.1 Non-commutative Rules

The logic we work with in this chapter is intuitionistic noncommutative linear logic, where the modal ! is assumed to be allowed to commute.

and thus should be permitted the freedom of exchange, even in the noncommutative versions of linear logic. In this formulation reusable formulas are therefore allowed to permute.

There are a whole family of logics which could result from various additions of restricted exchange to noncommutative linear logic. The main point of difference within this family is the exact formulation of the rules of inference. However, most members of this family of logics have an undecidable validity problem.

In fact, the multiplicative and reuse operators are sufficient to encode undecidable problems in most of these logics. In other words, the constants and additive connectives are not necessary in order to simulate a Turing machine in noncommutative linear logic, although they appear to be necessary in commutative linear logic. Below we present the detailed proof of undecidability for a particular logic we will call FICL, which is the multiplicative and exponential fragment of ICL, a member of the noncommutative linear logic family.

David Yetter [97] has also studied a variant of noncommutative linear logic. In his work, he considered a system with two new modalities, k and K, which are related to ? and !. The k modality essentially marks those formulas which are free to be permuted both left and right through a sequent, despite the noncommutativity of the logic in general. The reusable formulas (marked with ! on the left, or ? on the right) are allowed to permute, but are also allowed the freedom of contraction and weakening, while the k and K formulas are not. The undecidability result of this chapter therefore holds for Yetter's formulation of noncommutative linear logic as well.

We now focus on FICL, a fragment of ICL sufficient to encode Turing machines. This logic includes only the multiplicative and exponential connectives of linear logic, excluding the additives and constants, but including restricted versions of the exchange rule which only applies to ! formulas.

Chapter 4

Noncommutative Propositional Linear Logic

The following is called the *unrestricted exchange* rule:

$$\textbf{Exch. R} \quad \frac{\Theta \vdash \Sigma, A, B\Gamma}{\Theta \vdash \Sigma, B, A, \Gamma}$$

Since this rule is present in full linear logic, sequents are often treated as *multisets* of formulas, and the exchange rules are often considered implicit in the display of sequent proofs. This structural rule allows sequents to be permuted arbitrarily, making linear logic a commutative logic. More specifically, $\vdash (A \otimes B) \multimap (B \otimes A)$ is derivable in linear logic using exchange, as are the analogous sequents for all the other binary connectives of linear logic ($\parr$, $\oplus$, $\&$). However, the absence of the **E** rule (treating sequents as *sequences* of formulas, without any rule of exchange) drasticly alters the set of provable sequents in linear logic. In fact, without the exchange rule, $\vdash (A \otimes B) \multimap (B \otimes A)$ is not derivable.

Noncommutative propositional linear logic is linear logic where the unrestricted exchange rule is omitted, or equivalently, where sequents are treated as being lists instead of multisets. This family of logics is somewhat speculative, though all are very closely related to work by Lambek [55],Yetter [97], and others. However, the immediate resulting system is unsatisfying in that the reusable formulas (those marked by ?) are exactly the ones which can be contracted and weakened in linear logic,

3.4 Summary of Chapter

In this section it was shown that propositional linear logic is undecidable. This fact follows from a computational reading of some linear logic proofs as (ACM) computations. The reduction from the halting problem for ACMs to a decision problem in linear logic is factored through decision problems in linear logic augmented with nonlogical theory axioms.

This undecidability result is perhaps somewhat surprising, but should not be viewed as a negative result: linear logic is an extremely expressive logic, and finding that the full (propositional) logic is undecidable merely implies that linear logic embodies notions closer to computing machinery than previously envisioned.

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

Figure 3.1: Zero-test proof

$$\dfrac{\dfrac{q_I \vdash (q_2 \otimes a)}{}\delta'_1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q_3, a \vdash q_F}\delta'_2 \quad \dfrac{}{q_F \vdash q_F}I}{q_3, a \vdash q_F}Cut \qquad \vdots}{z_B, a \vdash q_F}}{(z_B \oplus q_3), a \vdash q_F}\oplus\mathbf{L} \qquad \dfrac{q_2 \vdash (z_B \oplus q_3)}{}\delta'_3}{q_2, a \vdash q_F}Cut}{(q_2 \otimes a) \vdash q_F}\otimes\mathbf{L}}{q_I \vdash q_F}Cut$$

Figure 3.2: Proof corresponding to computation

The proof shown in Figure 3.2 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 elided proof of $z_B, a \vdash q_F$ appears in Figure 3.1, and corresponds to the verification that the B counter is zero. The application of **Cut**, theory axiom, and identity correspond to the final decrement instruction of the computation, and complete the proof.

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 3.1 and 3.2, 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 3.1 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 & 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.

a simple computation of an ordinary two counter machine *with* zero-test instruction, a corresponding ACM computation, and a corresponding linear logic proof.

Repeating from the introduction, a key insight is that searching for a directed proof of a linear logic sequent in a theory is analogous to searching for an accepting ACM computation. The product of a successful search is an accepting computation.

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. 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** $A\ Q_F$	$q_3 \vdash a, q_F$
δ'_3	::=	Q_2 **Fork** Z_B, Q_3	$q_2 \vdash (z_B \oplus q_3)$
δ'_4	::=	Z_B **Decrement** $A\ Z_B$	$z_B \vdash a, z_B$
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

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case a **Fork** instruction), and then continuing as dictated by the two inductive cases. ∎

From Lemmas 3.1.11, 2.7.8, 2.7.9, 3.2.12, and 3.2.13 of this section, we easily obtain our main result:

Theorem 3.2.14 *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.

In any theory derived from an ACM M, there is only one positive formula in any theory axiom. Also, throughout a directed proof of $\theta(s)$ in such a theory, the only positive atom which appears outside a theory axiom is q_F. Thus any directed proof of $\theta(s)$ in a theory derived from M is in the intuitionistic fragment of linear logic, and along with a conservativity result not proven here, we have the following:

Corollary 3.2.15 *The provability problem for propositional intuitionistic linear logic is recursively unsolvable.*

In the proof of this corollary we make use of the conservativity property of full linear logic over the intuitionistic fragment for any sequents occurring in a directed proof of a translation of an ACM machine configuration. This conservativity is a weaker property than full conservativity since sequents in such a directed proof have a special form. In particular, they have no constants, and the right hand side is always a single formula.

3.3 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 above. We present

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case a **Decrement** A instruction), and then continuing as dictated by the inductive case.

$q_i \vdash a, q_j$: Analogous arguments apply.

$q_i \vdash (q_j \oplus q_k)$: If the last axiom applied is $q_i \vdash (q_j \oplus q_k)$, which corresponds to a **Fork** instruction, then by standardization, we know the cut-formula must be $(q_j \oplus q_k)$ in the axiom, and that the proof must look like

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \begin{array}{c}\vdots\\ (q_j \oplus q_k), a^x, b^y \vdash q_F\end{array}}{q_i, a^x, b^y \vdash q_F}Cut$$

Since each other linear logic rule besides $\oplus$ L, cut, identity, or axiom introduces some symbol which does not occur in $(q_j \oplus q_k), a^x, b^y \vdash q_F$, the derivation of this sequent must end in one of these rules. Furthermore, there are two formulas in the sequent which are not negative literals, so this sequent is not derivable using only an axiom. Identity could not lead to this sequent, since the sequent contains a non-atomic formula. By our standardization procedure, we know that each cut must involve an axiom from the theory, and the cut-formula in the axiom is not a negative literal. Inspecting the various types of axioms in the theory derived from M, we see that all axioms contain one top level negative atomic formula q_i for some i. Since q_i cannot be the cut-formula in a principal axiom of a directed cut, it must appear in the conclusion of that application of cut. However, there is no such top level q_i in the sequent in question. Thus this sequent may only be derived by the application of the $\oplus$ L rule. Thus we know the derivation to be of the form:

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \dfrac{\begin{array}{c}\vdots\\ q_j, a^x, b^y \vdash q_F\end{array} \qquad \begin{array}{c}\vdots\\ q_k, a^x, b^y \vdash q_F\end{array}}{(q_j \oplus q_k), a^x, b^y \vdash q_F}\oplus\mathbf{L}}{q_i, a^x, b^y \vdash q_F}Cut$$

The proofs of $q_j, a^x, b^y \vdash q_F$ and $q_k, a^x, b^y \vdash q_F$ can be simulated on the machine by induction, since one is a sequent which corresponds to the triple $\langle Q_j, x, y\rangle$, the other corresponds to $\langle Q_k, x, y\rangle$, and each has a proof in linear logic of smaller size.

are not negative literals, so this sequent is not derivable using only an axiom. Identity could not lead to this sequent, since the sequent contains a non-atomic formula. By our standardization procedure, we know that each cut must involve an axiom from the theory, and the cut-formula in the axiom is not a negative literal. Inspecting the various types of axioms in the theory derived from M, we see that all axioms contain one top level negative atomic formula q_i for some i. Since q_i cannot be a directed cut-formula in a principal axiom, it must appear in the conclusion of that application of cut. However, there is no such top level q_i in the sequent in question. Thus this sequent may only be derived by the application of the $\otimes$ L rule. Therefore, we know the derivation must have the form:

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \quad \dfrac{\vdots}{\dfrac{q_j, a^{x+1}, b^y \vdash q_F}{(q_j \otimes a), a^x, b^y \vdash q_F}\otimes\mathbf{L}}}{q_i, a^x, b^y \vdash q_F}Cut$$

We know that the proof of $q_j, a^{x+1}, b^y \vdash q_F$ may be simulated by the ACM by induction, since it is the sequent $\theta(\langle Q_j, x+1, y\rangle)$, which corresponds to the triple $\langle Q_j, x+1, y\rangle$, and has a proof in linear logic of smaller size.

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case an **Increment** A instruction), and then continuing as dictated by the inductive case.

$q_i \vdash (q_j \otimes a)$**:** Analogous arguments apply.

$q_i, a \vdash q_j$**:** If the last axiom applied is $q_i, a \vdash q_j$, which corresponds to a **Decrement** A instruction, then by standardization, we know the cut-formula must be q_j in the axiom, and that the proof must be of the form

$$\dfrac{\dfrac{}{q_i, a \vdash q_j}T \quad \dfrac{\vdots}{q_j, a^x, b^y \vdash q_F}}{q_i, a^{x+1}, b^y \vdash q_F}Cut$$

By induction, the proof of $q_j, a^x, b^y \vdash q_F$ can be simulated, since it is the sequent $\theta(\langle Q_j, x, y\rangle)$, which corresponds to the triple $\langle Q_j, x, y\rangle$, and has a shorter proof in linear logic.

We assume that we are given a proof of each element of the set $\theta(s)$, and we analyze one of the proofs, all of which end in a conclusion corresponding to a machine triple $\langle Q_i, x, y\rangle$.

$$\vdots$$
$$q_i, a^x, b^y \vdash q_F$$

Since this sequent is simply a list of atomic propositions, the only linear logic rules which can apply to any such sequent are identity, some axiom, and cut.

Identity is only applicable when both x and y are zero, and $q_i = q_F$. In this case, $q_F \vdash q_F$ already corresponds to the accepting triple $\langle Q_F, 0, 0\rangle$.

The only axioms which are identical to a sequent in $\theta(s)$ are those which correspond to some δ which is a decrement instruction that ends in q_F. In this case, since each decrement axiom in $[\delta]$ contains exactly one occurrence of a or b, $x = 1$ and $y = 0$, or $x = 0$ and $y = 1$. In either case, the ACM machine M need only perform the decrement instruction δ, and this branch of computation reaches an accepting triple.

The final possibility is cut, and by our standardization procedure, we know that one hypothesis of that cut is an axiom from the theory derived from M, and furthermore that the cut-formula in that axiom is not a negative literal.

Since there are only five types of instructions in an ACM; **Increment** A or B, **Decrement** A or B, and **Fork**, there are only five different types of axioms in a theory derived from any ACM M. We now perform case analysis on the type of axiom that was last applied in a proof.

$q_i \vdash (q_j \otimes a)$: If the last axiom applied is of the form $q_i \vdash (q_j \otimes a)$, then it corresponds to an **Increment** A instruction, and by standardization, we know the cut-formula must be $(q_j \otimes a)$ in the axiom, and that the proof must look like

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \qquad \begin{array}{c}\vdots\\ (q_j \otimes a), a^x, b^y \vdash q_F\end{array}}{q_i, a^x, b^y \vdash q_F}Cut$$

Since each other linear logic rule besides $\otimes$ L, cut, identity, or axiom introduces some symbol which does not occur in $(q_j \otimes a), a^x, b^y \vdash q_F$, the derivation of this sequent must end in one of these rules. Furthermore, there are two formulas in this sequent which

Q_i **Fork** Q_j, Q_k**:** Here, the halting computation begins with the step

$$\{\cdots\langle Q_i, A, B\rangle\cdots\} \rightarrow \{\cdots\langle Q_j, A, B\rangle, \langle Q_k, A, B\rangle\cdots\}$$

We assume by induction that we have a proof of $q_j, a^A, b^B \vdash q_F$,
and of $q_k, a^A, b^B \vdash q_F$, and we extend those proofs into a proof of $q_i, a^A, b^B \vdash q_F$.

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \dfrac{\begin{array}{cc}\vdots & \vdots \\ q_j, a^A, b^B \vdash q_F & q_k, a^A, b^B \vdash q_F\end{array}}{(q_j \oplus q_k), a^A, b^B \vdash q_F}\oplus\mathbf{L}}{q_i, a^A, b^B \vdash q_F}Cut$$

Here $q_i \vdash (q_j \oplus q_k)$ is the axiom which corresponds to the fork instruction. ∎

Lemma 3.2.13 (Machine $\Leftarrow$) *An and-branching counter machine M accepts from ID s if every sequent in the set $\theta(s)$ is provable in the theory derived from M.*

Proof.

Given a set of proofs of the elements of $\theta(s)$ in the theory derived from M, we claim that a halting computation of the ACM M from state s can be extracted from those proofs. We achieve this with the aid of the cut standardization Lemma 2.7.10, 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 standardized proof.

By Lemma 2.7.10, it suffices to consider standardized proofs. We show that a set of standardized proofs of $\theta(s)$ may be mimicked by the ACM M to produce an accepting computation from state s.

This proof is by induction on the sum of the sizes (number of proof rules applied) of standardized proofs. Since an ACM state is given by a finite set of triples, and all proofs are finite, we know that this measure is well founded. We assume that any smaller set of proofs which all end in conclusions which correspond to a triple $\langle Q_i, A, B\rangle$ can be simulated by machine M.

We consider the proof of a single element of $\theta(s)$ at a time.

If $s = \{\cdots\langle Q_i, x, y\rangle\cdots\}$, then $\theta(s) = \{\cdots q_i, a^x, b^y \vdash q_F \cdots\}$.

Q_i **Increment A** Q_j**:** In this case, the first step in the halting computation has the form

$$\{\cdots\langle Q_i, A, B\rangle\cdots\} \rightarrow \{\cdots\langle Q_j, A+1, B\rangle\cdots\}$$

We assume by induction that we have a proof of $\theta(\langle Q_j, A+1, B\rangle) = q_j, a^{A+1}, b^B \vdash q_F$. We extend this proof into a proof of $\theta(\langle Q_i, A, B\rangle) = q_i, a^A, b^B \vdash q_F$ by adding a cut with an axiom, as follows.

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \quad \dfrac{\vdots}{\dfrac{q_j, a^{A+1}, b^B \vdash q_F}{(q_j \otimes a), a^A, b^B \vdash q_F}\otimes\mathbf{L}}}{q_i, a^A, b^B \vdash q_F}Cut$$

Note that the axiom $q_i \vdash (q_j \otimes a)$ is precisely the translation of the transition taken by the machine, and therefore is an axiom of the theory.

Q_i **Increment B** Q_j**:** Analogous to above.

Q_i **Decrement A** Q_j**:** Since the A counter of the machine must be positive for this instruction to apply, we know that the halting computation begins with the transition

$$\{\cdots\langle Q_i, A+1, B\rangle\cdots\} \rightarrow \{\cdots\langle Q_j, A, B\rangle\cdots\}$$

We assume by induction that we have a proof of $q_j, a^A, b^B \vdash q_F$. As in the **Increment** A case, we extend this to a proof of $q_i, a^{A+1}, b^B \vdash q_F$ by adding a cut with the axiom corresponding to the transition taken by the machine.

$$\dfrac{\dfrac{}{q_i, a \vdash q_j}T \quad \dfrac{\vdots}{q_j, a^A, b^B \vdash q_F}}{q_i, a^{A+1}, b^B \vdash q_F}Cut$$

Q_i **Decrement B** Q_j**:** Analogous to above.

The translation of an ACM ID is simply the set of translations of the elements of the ID:

$$\theta(\{E_1, E_2, \cdots, E_m\}) = \{\theta(E_1), \theta(E_2), \cdots, \theta(E_m)\}$$

We claim that an ACM M accepts from ID s if and only if every element of $\theta(s)$ is provable in the theory corresponding to the transition function of the machine. We prove each half of this equivalence in separate lemmas.

Lemma 3.2.12 (Machine $\Rightarrow$) *An and-branching counter machine M accepts from ID s only if every sequent in $\theta(s)$ is provable in the theory derived from M.*

Proof. Given a halting computation of an ACM machine M from s we claim we can build a proof of every sequent in $\theta(s)$ in the theory derived from M.

M accepts from s only if there is some finite sequence of transitions from this ID to an accepting ID. We proceed by induction on the length of that sequence of transitions.

If there are no transitions in the sequence, then by the definition of accepting ID, s consists entirely of $\langle Q_F, 0, 0\rangle$. We must show that the sequent

$$q_F, a^0, b^0, \vdash q_F$$

is provable in linear logic. This is immediate: we have 0 A's and 0 B's, that is, none at all. Thus by one application of identity (per sequent) $q_F \vdash q_F$, we have our (set of) proof.

If there is at least one transition in the sequence, we have to show that $\theta(s)$ is provable. Since M accepts from ID $\{\cdots\langle Q_i, A, B\rangle\cdots\}$, and there is at least one transition in the sequence, we know that there is some transition in M such that $ID \rightarrow ID'$, and M accepts from ID'. We assume by induction that there is a linear logic proof which corresponds to the accepting computation for ID'.

We now perform case analysis on the type of transition. There are five different types of instructions: **Increment** A or B, **Decrement** A or B, and **Fork**. Since the two increment and two decrement instructions are nearly identical, we will concentrate only on the cases concerning the counter A.

3.2 From Machines to Logic

We give a translation from ACMs to linear logic with theories and show that our sequent translation of a machine in a particular state is provable in linear logic if and only if the ACM halts from that state. In fact, our translation uses only MALL formulas and theories, thus with the use of our earlier encoding, Lemma 2.7.8 and Lemma 2.7.9, we will have our result for propositional linear logic without nonlogical axioms. Since an instantaneous description of an ACM is given by a list of triples, it is somewhat delicate to state the induction we will use to prove soundness.

The main idea of this encoding is to use the linear connective & to simulate and-branching behavior, along with previously described techniques for encoding petri-net like tokens using the multiplicative connectives. These machine parts may be combined to build an ACM.

Given an ACM $M = \langle Q, \delta, Q_I, Q_F \rangle$ we first define a set of propositions:

$$\{q_i | Q_i \in Q\} \bigcup \{a, , b\}$$

We then define the linear logic theory corresponding to the transition relation δ as the set of axioms determined as follows:

$$\begin{aligned}
Q_i \textbf{ Increment A } Q_j &\mapsto q_i \vdash (q_j \otimes a) \\
Q_i \textbf{ Increment B } Q_j &\mapsto q_i \vdash (q_j \otimes b) \\
Q_i \textbf{ Decrement A } Q_j &\mapsto q_i, a \vdash q_j \\
Q_i \textbf{ Decrement B } Q_j &\mapsto q_i, b \vdash q_j \\
Q_i \textbf{ Fork } Q_j, Q_k &\mapsto q_i \vdash (q_j \oplus q_k)
\end{aligned}$$

Given a triple $\langle Q_i, x, y \rangle$ of an ACM, we define the translation $\theta(\langle Q_i, x, y \rangle)$ by:

$$\theta(\langle Q_i, x, y \rangle) \triangleq q_i, a^x, b^y, \vdash q_F$$

Thus all sequents which correspond to ACM triples have exactly one positive literal, q_F, some number of as, and bs, the multiplicity of which correspond to the values of the two counters of the ACM in unary, and exactly one other negative literal, q_i, which corresponds to the state of the ACM.

Decrement B transition from Z_A to itself allows M'' to loop and decrement the counter B arbitrarily. In particular it is possible for B to be decremented to the value 0. Since Q_F has no outgoing transitions, the **Fork** instruction which moves from Z_A to Q_F and Q_F allows this branch of computation to terminate correctly if and only if both counters are zero when it is executed. Since we are considering nondeterministic ACMs, it is possible for a branch of computation which reaches Z_A to terminate if and only if the A counter is zero when it reaches Z_A. Similarly, any branch of computation reaching Z_B reaches an accepting ID if and only if the B counter is zero.

We claim that there is a halting computation for the given two counter machine M' if and only if there is one for the constructed ACM M''. This is proven by two simulations.

The and-branching machine M'' may mimic the original two counter machine in the performance of any instruction, by following any **Increment** of M' with the corresponding **Increment** instruction, and a **Decrement** with the corresponding **Decrement**. When M' executes a **Zero-Test** A instruction, M'' forks off an *and* branch which verifies that the counter A is in fact zero, and the other branch continues to follow the computation of M'.

For the converse simulation, there is always at most one and-branch of any M'' computation which corresponds to a nonfinal, non-zero-testing state in the original machine. There may be many *and* branches of the computation which are in states Z_A, Z_B, and Q_F, but at most one *and* branch is in any other state. Thus, M' may mimic M'' by following the branch of ACM computation which does not enter Z_A, Z_B, or Q_F until the final step of computation, when it enters Q_F. For every **Increment** and **Decrement** instruction in the accepting computation of M'', M' may performs the corresponding instruction. Every **Fork** instruction executed by M'' from a nonfinal, non-zero-testing state corresponds to a **Zero-Test** instruction in M', and by the above observation, if M'' forks into state Z_A, then M'' accepts only if the counter A is zero (and similarly for Z_B and the counter B). Since we are assuming an accepting M'' computation, we know that M' may execute the corresponding **Zero-Test** instruction successfully. ■

We claim without proof that M and M' accept the same set of input values, and are therefore equivalent machines.

From a nondeterministic two counter machine M' with unique final state without outgoing transitions, we construct an ACM M'' as follows. The ACM M'' will have the same set of states, and same initial and final states as M'. The transition function of M'' is built by first taking all the **Increment** and **Decrement** instructions from the transition function of M'. We then add two new states to M'', Z_A and Z_B, which are used to test for zero in each of the two counters. For Z_A, we add two instructions, (Z_A **Decrement** B Z_A), and (Z_A **Fork** Q_F, Q_F), to the transition function of M''. Similarly for Z_B, we add (Z_B **Decrement** A Z_B), and (Z_B **Fork** Q_F, Q_F). Then for each **Zero-Test** instruction of M' of the form

$$(Q_i \textbf{ Zero-Test } \text{A } Q_j)$$

we add one instruction to M'':

$$(Q_i \textbf{ Fork } Q_j,\ Z_A).$$

An important feature of M'' is that once a zero testing or final state is entered, no configuration of that branch of computation may ever leave that set of states. More specifically, where M' would test for zero, M'' will fork into two "parallel" computations. One continues in the "same" state as M' would have if the **Zero-Test** had succeeded, and the other branch "verifies" that the counter is indeed zero. While the second branch may change the value of one of the counters (the counter which is not being tested), this cannot affect the values of the counters in the "main" branch of computation. Further, the zero-testing branch of computation never enters any states other than zero-test states or the final state. This holds because there are no outgoing transitions from the final state whatsoever, and the only transitions from the two zero testing states either loop back to that state or move directly to the final state. Also note that any branch of an ACM M'' computation which arrives at the state Z_A may be part of a terminating computation if and only if the counter A is zero when the machine reaches that state. This can be seen by observing that once arriving in Z_A, there is no possibility of modifications to the counter A. The

A two counter machine accepts if it is able to reach any one of the final states in the set F with both counters at zero. It is important that these machines have a **Zero-Test** instruction since the halting problem becomes decidable otherwise, by obvious reduction to the word problem in commutative semi-Thue systems, which is decidable [69]. Since **Zero-Test** is the most difficult to encode in linear logic, we concentrate on a machine with and-branching, which provides a basic mechanism powerful enough to simulate **Zero-Test**, but which is more easily simulated in linear logic.

Using two counter machines, we show that ACM's have an undecidable halting problem.

Lemma 3.1.11 *It is undecidable whether an and-branching two counter machine without zero-test accepts from ID* $\{\langle Q_I, 0, 0\rangle\}$. *This remains so if the transition relation of the machine is restricted so that there are no outgoing transitions from the final state.*

Proof. Since ACM's may simulate zero-test with and-branching, ACM's are sufficiently powerful to simulate two counter machines, for which the halting problem is known to be recursively unsolvable [75, 56]. We will give a construction from standard two counter machines to ACMs, and argue that the construction is sound and faithful. This construction and the proof of its soundness is routine, and its steps should be familiar to anyone versed in automata theory. In our simulation of the test for zero instruction of two counter machines, we make essential use of the fact that all branches of computation terminate with both counters set to zero.

Given a nondeterministic two counter machine M we first construct an equivalent two counter machine M' with a unique final state Q_F which has no outgoing transitions. One simply adds two new states, Q_D and Q_F to M', and for each $Q_f \in F$ of M, one adds the instructions (Q_f **Increment** A Q_D) and (Q_D **Decrement** A Q_F). Note that one may simply look at these new transitions as a single nondeterministic step from each old final state to the new (unique) final state, which has no outgoing transitions. However, since there is no general "silent" move, we make the transition in two steps.

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 Q_i, Q_j, and Q_k are states in Q. The **Decrement** instructions only apply if the appropriate counter is not zero, while the **Increment** and **Fork** instructions are always enabled from the proper state.

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

3.1.1 Two Counter Machines

Standard two counter machines have a finite set of states, Q, a finite set of transitions, δ, a distinguished initial state Q_I, and a set of final states F [75, 42]. An instantaneous description of the state of a two counter machine is given by a triple $\langle Q_i, A, B\rangle$, which consists of the current state, and the values of two counters, A and B. The transitions in δ are of four kinds:

$(Q_i$ **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 $\langle Q_j, A+1, B\rangle$
$(Q_i$ **Decrement** B $Q_j)$	taking	$\langle Q_i, A, B+1\rangle$ to $\langle Q_j, A, B+1\rangle$
$(Q_i$ **Zero-Test** A $Q_j)$	taking	$\langle Q_i, 0, B\rangle$ to $\langle Q_j, 0, B\rangle$
$(Q_i$ **Zero-Test** B $Q_j)$	taking	$\langle Q_i, A, 0\rangle$ to $\langle Q_j, A, 0\rangle$

for the lack of an explicit zero test transition, and the addition of "fork" transitions. 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. However, there is no intrinsic reason to restrict the machines to two counters. All of our arguments and results generalize easily to N counters, for $N \geq 2$. 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 described below.

An *instantaneous description*, or *ID*, of an ACM M is a finite 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. Intuitively, a list of triples represents a set of machine configurations. One may think of an ACM state as some sort of parallel computation which terminates successfully only if all its concurrent computation fragments terminate successfully.

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 M *accepts from* an ID s if and only if there is some computation from s to an accepting ID. It is essential for our encoding in linear logic that both counters be zero in all elements of an accepting ID.

step used in this section relies heavily on the cut-elimination procedure for linear logic without nonlogical axioms, first sketched by Girard in [30]. A very explicit proof of cut-elimination for full propositional linear logic also appears in Chapter 2 which some readers may find helpful to skim before continuing.

It should be noted that linear logic is not the only known undecidable propositional logic — propositional relevance logic is also undecidable [88]. However, the undecidability of these logics appear to arise in very different ways; linear logic fails to have a critical distributivity property used in the proof of undecidability of relevance logic, and relevance logic fails to have one of the key connectives used in the proof of undecidability of linear logic.

The encoding of an undecidable problem in linear logic hinges on the combination of three powerful mechanisms: resource accumulation, arbitrary reuse, and and-branching. In linear logic, A, A is very different from A, and this allows us to represent counters in unary. Indefinitely reusable formulas such as $!(B \multimap C)$, (or axioms of the form $B \vdash C$) may be used to implement machine instructions. The additive connectives & and $\oplus$ may be used to test a conjunction of properties of the simulated machine, such as whether a counter is zero, and if the rest of the computation can continue normally. Together this machinery is enough to encode recursively unsolvable problems in linear sequents.

3.1 And-Branching Two Counter Machines Without Zero-Test

In this section we describe nondeterministic two counter machines with *and*-branching but without a zero-test instruction. We show that these machines have a recursively unsolvable halting problem, and then we will show how the halting problem for these machines may be encoded as a decision problem in MALL, with nonlogical axioms corresponding to the machine instructions.

The machines described here are similar to standard two counter machines except

Chapter 3

Propositional Linear Logic is Undecidable

In this chapter we show that unlike most propositional logics, the decision problem for propositional linear logic is not recursively solvable. We study this problem through the use of nonlogical axioms in the fragment of linear logic without modal operators (MALL). Since this class of axioms may be encoded in full propositional linear logic, as shown in Chapter 2, undecidability of MALL with nonlogical axioms implies the undecidability of full propositional linear logic.

The proof of undecidability of MALL with nonlogical axioms consists of a reduction from the halting problem for a form of counter machine to a decision problem in linear logic. In more detail, we begin by extending propositional linear logic with theories whose (nonlogical) axioms may be used any number of times in a proof. We then describe a form of *and*-branching two-counter machine with an 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 in the presence of nonlogical axioms. To illustrate the encoding of two-counter machines, we present an example simulation of a simple computation in Section 3.3. On first reading, the reader may wish to jump ahead to that section since it demonstrates the basic mechanism used in the undecidability proof. Also, the crucial cut-standardization

2.8 Summary of Chapter

In this chapter, we have presented several proof theoretic results. These results are relatively minor extensions of theorems already in the literature of linear logic. However, the exact form of some of these theorems, such as cut-elimination, are used to develop richer proof theoretic tools, such as cut standardization (Lemma 2.7.10) in the presence of theory axioms. These results will be used as the basis for later theorems, such as the undecidability of full propositional linear logic.

This completes the discussion of the modifications to Lemma 2.3.2 necessary to handle nonlogical axioms. Fortunately, Lemma 2.3.3 and Theorem 2.3.4 then follow without modification (although the definition of degree has changed somewhat).

Therefore, given any proof of a sequent $\Theta \vdash \Gamma$ in theory T, we can construct a directed proof of $\Theta \vdash \Gamma$ in theory T. ∎

The cut-elimination procedure in Section 2.3 introduces new rules of inference called **Cut**! and **Cut**?. If we generalized axioms to allow ? and ! formulas in axioms, we would have to generalize the notion of directed proof to include cases involving **Cut**! and **Cut**?, and a post processing step would be required to transform all directed **Cut**! and **Cut**?s into a sequence of contractions followed by a single directed **Cut**, or perhaps simply into a sequence of **Cut**s. In any event, our axioms are restricted to MALL formulas so that any cut involving an axiom is always an application of **Cut**, never of **Cut**! nor of **Cut**?.

in the right hypothesis is an axiom, we apply the following transformation:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta_1 \vdash \Sigma, C & \Theta_2 \vdash B, \Delta, p_i\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, (C \otimes B), \Delta, p_i}\otimes \quad \overline{\Theta_3 \vdash \Gamma, p_i^{\perp}}^{T}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (C \otimes B), \Delta, \Gamma}Cut$$

$$\Downarrow$$

$$\dfrac{\begin{array}{cc}\vdots & \dfrac{\begin{array}{cc}\vdots & \\ \Theta_2 \vdash B, \Delta, p_i & \overline{\Theta_3 \vdash \Gamma, p_i^{\perp}}^{T}\end{array}}{\Theta_2, \Theta_3 \vdash B, \Delta, \Gamma}Cut \\ \Theta_1 \vdash \Sigma, C & \end{array}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (C \otimes B), \Delta, \Gamma}\otimes$$

This is simply a special case of the reduction given in case 2.3.1 in Section 2.3.

Also, as a second example, the reduction given for Identity is applicable even to the axiom case:

$$\dfrac{\overline{p_i \vdash p_i}^{I} \quad \overline{\Theta, p_i \vdash \Gamma}^{T}}{\Theta, p_i \vdash \Gamma}Cut \quad \Longrightarrow \quad \overline{\Theta, p_i \vdash \Gamma}^{T}$$

Again, this is simply a special case of the reduction given in Section 2.3.

As a third and final example of specializations of reductions given in the appendix, the $\top$ R rule also applies to axioms:

$$\dfrac{\overline{\Theta_1 \vdash \top, \Sigma, p_i}^{\top} \quad \overline{\Theta_2, p_i \vdash \Gamma}^{T}}{\Theta_1, \Theta_2 \vdash \top, \Sigma, \Gamma}Cut \quad \Longrightarrow \quad \overline{\Theta_1, \Theta_2 \vdash \top, \Sigma, \Gamma}^{\top}$$

This is also simply a special case of the reduction given in the appendix.

Now, some simple analysis is required to show that there are no new cases of principal cuts involving axioms. If the cut in question is already directed, the cut has degree zero, by our modified definition, and thus we are done. Otherwise, by definition of axiom we know that the cut-formula is a negative atomic literal. There are only three rules where an atomic literal may be principal: Identity, 0 L, and $\top$ R. However, all of these cases are handled by existent reductions (two of which are restated above). One should also note that since any cut involving two axioms must be directed, we needn't provide a reduction for that case.

By case analysis, it may be seen that there is no proof of this conclusion sequent that doesn't contain cut. However, we do obtain the following result:

Lemma 2.7.10 (Cut Standardization) *If there is a proof of* $\Theta \vdash \Delta$ *in theory* T*, then there is a directed proof of* $\Theta \vdash \Delta$ *in theory* T*.*

Proof. We modify the cut-elimination procedure defined in Section 2.3 in two ways. First we alter the definition of degree to ignore the measure of directed cuts. Formally, we say that if a cut is directed, its degree is zero. Second, we modify the procedure given in Lemma 2.3.2 to handle the extra cases brought about by the presence of axioms. We must allow all the reductions as stated in Appendix 2.3 to apply to the case when one of the premises is an axiom, but we need not introduce any truly novel reductions.

We will follow the notation used in Section 2.3, where **Cut**$*$ is used to ambiguously refer to the **Cut** rule or the extra rule of inference introduced in the appendix called **Cut**!. Also, we will define all the formulas which appear in an axiom to be principal in that application of the axiom.

In Section 2.3 most of the reductions are given for some specific derivation versus any possible derivation. For example, all the non-principal cases are stated for any derivation of the "other" hypothesis of **Cut**$*$. Similarly, the Identity and $\top$ R rules are stated for any derivation of the "other" hypothesis. We simply now state that even if the other derivation involves an axiom, the reduction still applies.

For example, if the last rule applied in the left hypothesis is $\otimes$, and the last rule

induction on the number of axioms, cutting each one out in turn, we can derive $\Gamma \vdash \Delta$ in theory T. ∎

Despite the fact that one can encode arbitrary theories in linear logic, as demonstrated above, the remainder of this thesis will focus on a specific subclass of propositional, nonmodal theories. The theories we will define below are sufficient to encode recursively unsolvable problems, but yet allow a great deal of control over the shape of proofs.

2.7.2 Cut Standardization

We now focus on a special class of linear theories, where every theory axiom contains at most one formula at top level which is not a negative literal. Recall that a positive literal is one of the given propositional symbols p_i or $p_i^\perp$ which occurs with positive polarity, and a negative literal is one of these which occurs with negative polarity. Thus the class of theories of interest here contains linear logic sequents of the form $q_1, q_2, \cdots q_n \vdash C, p_1^\perp, p_2^\perp, \cdots, p_n^\perp$ or of the form $q_1, q_2, \cdots q_n, C \vdash p_1^\perp, p_2^\perp, \cdots, p_n^\perp$. For example, the sequents $q, r \vdash s, p^\perp$, $q, p \vdash (p \otimes q), p^\perp$, $(p \otimes p), q \vdash p^\perp$, and $\vdash p^\perp, q^\perp$ are all axioms in this class. However, $\vdash p_1, p_1$ and $(p_1 \otimes p_2), p_3^\perp \vdash$ are not.

A *directed* cut is one where at least one premise is an axiom in T, and the cut-formula in the axiom is not an atomic literal of negative polarity. We call any axiom premise of a directed cut where the cut-formula in that axiom is not a negative literal a *principal axiom* of that directed cut. By definition, all directed cuts have at least one principal axiom. A cut between two axioms is always directed, and if the cut-formula of such a cut is non-atomic, that cut has two principal axioms. A *directed* or *standardized* proof is a proof with all cuts directed.

When theories are added to linear logic the cut-elimination Theorem 2.3.4 no longer holds, due to the added axioms which may participate directly in cuts. Such cuts may never be removed: consider the nonlogical MALL theory consisting of two axioms: $p_1 \vdash p_2$ and $p_2 \vdash p_3$:

$$\dfrac{\dfrac{}{p_1 \vdash p_2}\mathbf{T} \qquad \dfrac{}{p_2 \vdash p_3}\mathbf{T}}{p_1 \vdash p_3}\mathbf{Cut}$$

of the conclusion sequent of the binary rule, to which we add an extra copy of $[T]$. Then we extend the proof further, adding one contraction step for each $[t_i]$ between that sequent and the original conclusion of that binary rule.

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta \vdash A, \Sigma & \Gamma \vdash B, \Delta\end{array}}{\Theta, \Gamma \vdash \Sigma, (A \otimes B), \Delta}\otimes \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta, [T] \vdash A, \Sigma & \Gamma, [T] \vdash B, \Delta\end{array}}{\Theta, [T], \Gamma, [T] \vdash (A \otimes B), \Sigma, \Delta}\otimes}{\vdots}\,{}^{!C}}{\Theta, [T], \Gamma \vdash (A \otimes B), \Sigma, \Delta}\,{}^{!C}$$

Thus we have given a construction which builds a proof of $\Theta, [T] \vdash \Sigma$ without any nonlogical axioms from a given proof of $\Theta \vdash \Sigma$ using axioms from T. ∎

Theorem 2.7.9 (Theory $\Leftarrow$) *For any finite set of axioms T, $\Gamma \vdash \Delta$ is provable in theory T if $\Gamma, [T] \vdash \Delta$ is provable without nonlogical axioms.*

Proof. Assuming we have a proof of $\Gamma, [T] \vdash \Delta$, we immediately have a proof of $\Gamma, [T] \vdash \Delta$ in theory T, since any proof in pure linear logic is also a proof in the logic extended with axioms. We can also build proofs of $\vdash [t_i]$ in the theory T for each axiom t_i. By cutting these proofs against the given proof of $\Gamma, [T] \vdash \Delta$, we obtain a proof of $\Gamma \vdash \Delta$ in theory T.

For each axiom $t_i \equiv A_1, A_2, \cdots, A_n \vdash B_1, B_2, \cdots, B_m$, we may prove $\vdash [t_i] \equiv \vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m))$ by several applications of $\otimes$ L, $\parr$ R, one application of $\multimap$ R, and one application of **!S**, as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A_1, A_2, \cdots, A_n \vdash B_1, B_2, \cdots, B_m}\,T}\otimes L}{\vdots}}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B_1, B_2, \cdots, B_m}\otimes L}\parr R}{\vdots}}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B_1 \parr B_2 \parr \cdots \parr B_m}\parr R}{\vdash (A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m)}\multimap R}{\vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m))}\,{}^{!S}$$

By cutting this proof against the given proof of $\Gamma, [T] \vdash \Delta$, we obtain a proof of $\Gamma, [T - \{t_i\}] \vdash \Delta$, where $T - \{t_i\}$ is the multiset difference of T and $\{t_i\}$. Thus by

does not use theory axioms (see below). For each leaf sequent which was originally an application of identity or 1 R, or $\bot L$, we extend the proof by adding weakenings for all the $![t_i]$ formulas. For each leaf sequent which consists of an application of $\top$ R, or 0 L, we simply add the $[T]$ formulas to the sequent. We then add $[T]$ to every sequent in the body of the entire proof tree. At every application of $\otimes R$, $\parr L$, $\multimap L$, and **Cut**, we extend the proof tree with contractions on each formula in $[T]$.

The first mentioned proof tree, of $\Theta, [T] \vdash \Sigma$, will be constructed from the proof tree for $\Theta, [t_i] \vdash \Sigma$. Since each formula in $[T]$ begins with !, we may weaken in the remainder of $[T]$, and thus with some number of weakening steps we have $\Theta, [T] \vdash \Sigma$. For example, if there are k axioms, and $\Theta \vdash \Sigma$ is the axiom $t_1 = q_1 \vdash (q_2 \otimes a)$, then we know $[t_1] = !(q_1 \multimap (q_2 \otimes a))$. We then perform the following transformation:

$$\dfrac{}{q_1 \vdash (q_2 \otimes a)}{\scriptstyle T} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{q_2 \vdash q_2}{\scriptstyle I} \quad \dfrac{}{a \vdash a}{\scriptstyle I}}{q_2, a \vdash (q_2 \otimes a)}{\scriptstyle \otimes R}}{\dfrac{}{q_1 \vdash q_1}{\scriptstyle I} \quad \dfrac{}{(q_2 \otimes a) \vdash (q_2 \otimes a)}{\scriptstyle \otimes L}}{q_1, (q_1 \multimap (q_2 \otimes a)) \vdash (q_2 \otimes a)}{\scriptstyle \multimap L}}{q_1, [t_1] \vdash (q_2 \otimes a)}{\scriptstyle !D}}{[t_1], [t_2], q_1 \vdash (q_2 \otimes a)}{\scriptstyle !W}}{\vdots}}{[t_1], [t_2], \cdots, [t_{k-1}], q_1 \vdash (q_2 \otimes a)}{\scriptstyle !W}}{[T], q_1 \vdash (q_2 \otimes a)}{\scriptstyle !W}$$

For each leaf sequent which was originally an application of identity or 1 R, or $\bot L$, we weaken in all the $[t_i]$ formulas:

$$\dfrac{}{p_i \vdash p_i}{\scriptstyle I} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{}{p_i \vdash p_i}{\scriptstyle I}}{[t_1], p_i \vdash p_i}{\scriptstyle !W}}{[t_1], [t_2], p_i \vdash p_i}{\scriptstyle !W}}{\vdots}}{[t_1], [t_2], \cdots, [t_{k-1}], p_i \vdash p_i}{\scriptstyle !W}}{[T], p_i \vdash p_i}{\scriptstyle !W}$$

We then continue by adding $[T]$ to every sequent in the entire proof tree. At every application of $\otimes R$, $\parr L$, $\multimap L$, and **Cut**, we extend the proof tree with an extra copy

2.7 Linear Logic Augmented With Theories

Essentially, a theory is a set of nonlogical axioms (sequents) that may occur as leaves of a proof tree. The use of theories described here is an extension of earlier work on multiplicative theories [39, 67].

The point of the extension of linear logic to encompass nonlogical theories is to highlight the difference between linear (non-! or MALL) formulas and resusable (!, or theory axiom) formulas. Later we will see that theory axioms capture the behavior of machine instructions such as a Turing machine's finite control, the transitions of a Petri net, or the transitions of a counter machine. The linear MALL formulas appearing in a sequent represents the current state of a machine.

This section shows that nonlogical MALL theories may be efficiently encoded in full propositional linear logic using the modal !. Thus one may study the computational significance of propositional linear logic by considering MALL augmented with nonlogical theories, knowing that any result for this system will carry over to full propositional linear logic in the end.

2.7.1 Encoding MALL Theories

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

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

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

$$[\, G_1, \cdots, G_n \vdash F_1, F_2, \cdots, F_m \,] \quad \triangleq \quad !((G_1 \otimes \cdots \otimes G_n) \multimap (F_1 \parr F_2 \parr \cdots \parr F_m))$$

Theorem 2.7.8 (Theory $\Rightarrow$) *For any finite set of axioms T, $\Gamma \vdash \Delta$ is provable in theory T only if $\Gamma, [T] \vdash \Delta$ is provable without nonlogical axioms.*

Proof. Given some proof of $\Gamma \vdash \Delta$ in theory T, we have a linear logic proof tree with axioms of T at some leaves. For each leaf of the proof tree of the form $\Theta \vdash \Sigma$, where $\Theta \vdash \Sigma$ is some axiom t_i, we replace that leaf with a small proof of $\Theta, [T] \vdash \Sigma$ which

0	various examples
1	$\vdash (A \parr B), (A^\perp \otimes B^\perp)$
2	$\vdash (A \& B), (\perp \otimes \top), A^\perp, B^\perp$
3	$\vdash (A \& B), (A^\perp \oplus B^\perp)$
4	$\vdash (1 \& A), ?A^\perp$
5	$\vdash ((!A) \& A), ?A^\perp$
6	$\vdash (A(t)^\perp \& A(u)^\perp), \exists x.A(x)$
7	$\vdash ?A, (A^\perp \otimes A^\perp)$
8	$\vdash !(A^\perp \oplus B), ?A$
9	$\vdash \forall y.(A(y) \oplus B), \exists x.A(x)^\perp$

Note that cases 4 and 5 are special: if we change the definition of permutable to "... if for any proof with C reduced immediately above C' IN ALL HYPOTHESES, then there is a proof with C reduced immediately below C", then 4 and 5 become permutable. Also, the impermutabilities involving Cut (labeled $-$) are sensitive to the exact definitions used: here the cut formula must be the same in the permuted proof, etc. With these definitions Cut versus & is special in the same sense as cases 4 and 5.

As a specific application of the above table of permutabilities, consider the following "invertability" property of the $\parr$ (right) rule.

Proposition 2.6.7 *If a sequent* $\Gamma \vdash \Delta, (A \parr B)$ *is provable in linear logic, then so is* $\Gamma \vdash \Delta, A, B$.

The proof of this property is immediate from emptyness of the $\parr$ column in the array of permutabilities.

$\forall$ must be instantiated lower in the proof tree than the $\exists$ in sequent 9 in classical as well as linear logic. However, classical logic enjoys all other possible permutabilities, while intuitionistic and linear logic have many impermutabilities.

Note that I, 1, and $\top$ are trivial cases, since they have no hypotheses. Also, $!S$ is only rarely permutable, since it requires all other formulas to be prefixed with ?, (although here only the !'d formula is considered principal in an application of !S). In all permutable cases for $!S$, except versus $?W$ and $?C$, the permutability is trivial. (There are no proofs with $\otimes$ immediately above $!S$ where the principal formulas are not subformula of one another.)

	$\otimes$	$\parr$	&	$\oplus$	$?W$	$?C$	$?D$	$!S$	$\bot$	$\forall$	$\exists$	Cut
$\otimes$								0				
$\parr$	1							0				–
&	2			3	4		5	0			6	–
$\oplus$								0				
$?W$												
$?C$	7											–
$?D$												
$!S$							8					–
$\bot$								0				
$\forall$								0			9	
$\exists$								0				
Cut								–				

2.6 Permutabilities of Linear Logic

In the previous section, the cut-elimination theorem was proved for linear logic. One may view a cut-free proof as the normal form of an infinite class of proofs with cut of the same formula. Thus the process of cut-elimination is often referred to as "proof normalization". In this section we turn our attention to a set of more specific normal form theorems for linear logic proofs. Specificly, we present a set of permutability theorems which may be used to convert proofs into various normal forms. This work builds on that of Mints [77] and Andreoli and Pareschi [7].

This class of results may be proved by applying Girard's sequentialization theorem from proof nets to the sequent calculus backward and forward [30], or by direct proof theoretic argument in the sequent calculus. The proofs in each case (using either method) are quite straightforward, and are omitted here.

Formally, the following is a list of some permutabilities and impermutabilities of linear logic. The table lists only the permutabilities for connectives as they appear on the right hand side of a sequent. From this one may easily extrapolate all possible permutabilities of two-sided linear logic, using the definition of negation to consider the behavior of the dual operator as it appears on the right.

A rule R for the main connective of a formula C is permutable below another rule R' for the main connective of formula C' (C not a subformula of C') if for any proof with C analyzed immediately above C', then there is a proof with C reduced immediately below C'. Using other terminology (see Bellin, for example), an inference R permutes below an inference R' if and only if for any proof where R occurs immediately above R', and the principal formula of R is not active in R', there is a proof where R occurs immediately below R'. It is easiest to consider cut-free proofs, although one may also study permutabilities involving cut if one is careful with the definitions.

Any numeral in the following table should be read as "the connective of this column cannot always permute below the connective of this row". For example, number 9 shows that $\exists$ cannot always permute below $\forall$. Example 9 is essentially the same example that shows quantifier impermutability in classical logic. That is, the

It is easy to see that the subformula property is not true of proofs with cut: the formula A in the hypotheses of **Cut** might not appear in the conclusion. The main historical interest in the subformula property is that it implies consistency of the logic.

2.5 Polarity

Another useful property of cut-free proofs is stated using *polarity*. We define the polarity of a formula to be the number of (implicit) negations that surround it. Formally, we define the polarity of a formula based on the following sets of transformations, beginning with $[\Sigma]^- \vdash [\Delta]^+$.

$$\begin{array}{rcl}
[A \multimap B]^+ & = & [A]^- \multimap [B]^+ \\
[A \otimes B]^+ & = & [A]^+ \otimes [B]^+ \\
[\Sigma, A]^+ & = & [\Sigma]^+, [A]^+ \\
[A^\perp]^+ & = & ([A]^-)^\perp \\
[A \multimap B]^- & = & [A]^+ \multimap [B]^- \\
[A \otimes B]^- & = & [A]^- \otimes [B]^- \\
[\Sigma, A]^- & = & [\Sigma]^-, [A]^- \\
[A^\perp]^- & = & ([A]^+)^\perp
\end{array}$$

The polarity of an instance of a formula A in a sequent $\Sigma \vdash \Delta$ is given by the sign of the superscript on A in $[\Sigma]^-$ or $[\Delta]^+$. That is, if an instance of formula A ends up as $[A]^+$, then it is of *positive* polarity. If an instance of formula A ends up as $[A]^-$, then it is of *negative* polarity.

Polarity is preserved throughout cut-free proofs, as stated formally below.

Lemma 2.5.6 *If a formula A has polarity p in an occurrence in a sequent in a cut-free proof of $\Gamma \vdash \Delta$, then A has polarity p in $\Gamma \vdash \Delta$.*

By induction, we can produce proofs of $\Theta_1 \vdash \Gamma_1, A$ and $\Theta_2, A \vdash \Gamma_2$ of degree less than d. By a single application of Lemma 2.3.2 to the resulting proof constructed from the modified hypotheses, we obtain a proof of $\Theta \vdash \Gamma$ of degree less than d. ∎

Theorem 2.3.4 (Cut-Elimination) *If a sequent is provable in linear logic, then it is provable in linear logic without using the* **Cut** *rule.*

Proof. By induction on the degree of the assumed proof. We may apply Lemma 2.3.3 at each inductive step, and at the base case the degree of the proof is zero, so therefore by definition of proof degree there are no cuts, and we have our desired cut-free proof. ∎

Note that the proof can explode hyperexponentially in size during the cut-elimination process.

2.4 Subformula Property

We have demonstrated that all cuts may be eliminated from a proof, at the possible expense of increasing the size of the proof hyperexponentially. This normalization is worthwhile, however, since cut-free proofs have useful properties. One such property is the subformula property.

Corollary 2.4.5 (Subformula Property) *Any formula of any sequent in any cut-free proof of* $\Theta \vdash \Gamma$ *is a subformula of* Θ *or* Γ.

Proof. Each rule of linear logic except **Cut** has the property that every subformula of the hypotheses is also a subformula of the conclusion. For example, in the $\otimes$ R rule, any subformula of either hypothesis is either a subformula of $\Gamma_1, \Gamma_2, \Sigma_1, \Sigma_2, A$, or B. However, any such subformula is also a subformula of the conclusion. In fact, we may have "added" a subformula: $(A \otimes B)$ is a subformula of the conclusion, but might not be a subformula of the hypotheses.

Therefore, by induction on the size of proofs, we have that any subformula of any step of a cut-free proof of a sequent is a subformula of the original sequent. ∎

1 **R versus** 1 **L**

$$\dfrac{\dfrac{}{\vdash 1}\,\mathbf{1R} \quad \dfrac{\vdots}{\dfrac{\Theta \vdash \Gamma}{\Theta, 1 \vdash \Gamma}\,\mathbf{1L}}}{\Theta \vdash \Gamma}\,\mathbf{Cut} \quad \Longrightarrow \quad \begin{array}{c}\vdots\\ \Theta \vdash \Gamma\end{array}$$

Again, we know that the **Cut**$*$ involved here is **Cut**, since the formula 1 was just introduced, and does not begin with ! or ?.

This exhausts all the cases. ∎

Thus, we have a procedure which given a proof which ends in **Cut**$*$ of degree d, and which has no applications of **Cut**$*$ in the proof of either hypothesis of degree greater than or equal to d, produces a proof of degree less than d.

Lemma 2.3.3 (Lower-Degree-Cuts) *If a sequent is provable in linear logic with a proof of degree $d > 0$, then it is provable in linear logic with a proof of degree less than d.*

Proof. By induction on the height of the derivation tree of the conclusion, we show that given any proof of degree d of $\Theta \vdash \Gamma$ in propositional linear logic, we may find a (possibly much larger) proof of $\Theta \vdash \Gamma$ in linear logic of degree less than d.

We examine the proof of $\Theta \vdash \Gamma$. Since the degree of this proof is greater than zero, there must be some **Cut**$*$ in the proof. If the last rule is not **Cut**$*$, then by induction we may form proofs of its hypotheses of degree less than d. Applying the same rule to the resulting reduced degree hypotheses produces the desired proof of degree less than d.

In the case that the last rule is **Cut**$*$, we have the following situation for some Θ_1 and Θ_2 which together (in multiset union) make up Θ, and similarly for Γ_1, Γ_2 which make up Γ: (where at most one of n, m is greater than one:

$$\dfrac{\begin{array}{cc}\vdots & \vdots\\ \Theta_1 \vdash \Gamma_1, A^n & \Theta_2, A^m \vdash \Gamma_2\end{array}}{\Theta \vdash \Gamma}\,\mathbf{Cut}*$$

then we apply the same reduction as in the non-principal $\otimes$ case (Section 2.3.1):

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A} \quad \dfrac{\vdots}{\Theta_2, !C^n \vdash B, \Delta}}{\Theta_1, \Theta_2, !C^n \vdash \Sigma, (A \otimes B), \Delta}\otimes\mathbf{R} \quad \dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\,!\mathbf{S}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma}\mathbf{Cut}*$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A} \quad \dfrac{\dfrac{\vdots}{\Theta_2, !C^n \vdash B, \Delta} \quad \dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\,!\mathbf{S}}{\Theta_2, !\Theta_3 \vdash B, \Delta, ?\Gamma}\mathbf{Cut}*}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma}\otimes\mathbf{R}$$

In the more complex case, when the cut-formulas descend from both hypotheses of $\otimes$, we use the following reduction to push the cut above the $\otimes$ rule.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1, !C^m \vdash \Sigma, A} \quad \dfrac{\vdots}{\Theta_2, !C^n \vdash B, \Delta}}{\Theta_1, \Theta_2, !C^{n+m} \vdash \Sigma, (A \otimes B), \Delta}\otimes\mathbf{R} \quad \dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\,!\mathbf{S}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma}\mathbf{Cut}!$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1, !C^m \vdash \Sigma, A} \quad \dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\,!\mathbf{S}}{\Theta_1, !\Theta_3 \vdash \Sigma, ?\Gamma, A}\mathbf{Cut}! \quad \dfrac{\dfrac{\vdots}{\Theta_2, !C^n \vdash B, \Delta} \quad \dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\,!\mathbf{S}}{\Theta_2, !\Theta_3 \vdash B, \Delta, ?\Gamma}\mathbf{Cut}!}{\Theta_1, !\Theta_3, \Theta_2, !\Theta_3 \vdash \Sigma, ?\Gamma, (A \otimes B), \Delta, ?\Gamma}\otimes\mathbf{R}$$

$$\dfrac{\Theta_1, !\Theta_3, \Theta_2, !\Theta_3 \vdash \Sigma, ?\Gamma, (A \otimes B), \Delta, ?\Gamma}{\vdots}\,!\mathbf{C}$$

$$\dfrac{\vdots}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma}\,?\mathbf{C}$$

the upper **Cut**$*$,we use the following reduction:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1, !C^m \vdash \Sigma, A}\quad\dfrac{\vdots}{\Theta_2, !C^n, A \vdash \Delta}}{\Theta_1, \Theta_2, !C^{n+m} \vdash \Sigma, \Delta}\mathbf{Cut}*\quad\dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\mathbf{!S}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma}\mathbf{Cut!}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1, !C^m \vdash \Sigma, A}\quad\dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\mathbf{!S}}{\Theta_1, !\Theta_3 \vdash \Sigma, ?\Gamma, A}\mathbf{Cut!}\quad\dfrac{\dfrac{\vdots}{\Theta_2, !C^n \vdash \Delta, A}\quad\dfrac{\dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}}{!\Theta_3 \vdash ?\Gamma, !C}\mathbf{!S}}{\Theta_2, !\Theta_3 \vdash \Delta, ?\Gamma, A}\mathbf{Cut!}}{\Theta_1, !\Theta_3, \Theta_2, !\Theta_3 \vdash \Sigma, ?\Gamma, \Delta, ?\Gamma,}\mathbf{Cut}*\,\mathbf{!C}$$

$$\vdots$$

$$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma,\ \mathbf{?C}$$

$\otimes$ R versus !S

There are two possibilities here, which correspond to whether it is necessary to split a package into two pieces. The case where the package needs to be split is again one of the most tricky aspects of the entire cut-elimination procedure.

If **Cut**$*$ is applied to formulas which may all be found in one hypothesis of $\otimes$,

Cut$*$ versus !S

There are two possibilities here, which correspond to whether it is necessary to split a package into two pieces. The case where the package needs to be split is one of the most tricky aspects of the entire cut-elimination procedure.

If the lower application of **Cut**$*$ is applied to formulas which may all be found in one hypothesis of the upper application of **Cut**$*$, then we apply the same reduction as in the non-principal **Cut** case (Section 2.3.1):

$$\dfrac{\dfrac{\vdots \quad \vdots}{\Theta_1 \vdash \Sigma, A \qquad \Theta_2, A, !C^n \vdash \Delta}\,\mathbf{Cut}* \qquad \dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}\,\mathbf{!S}}{\Theta_1, \Theta_2, !C^n \vdash \Sigma, \Delta \qquad\qquad !\Theta_3 \vdash ?\Gamma, !C}\,\mathbf{Cut}*$$
$$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A} \qquad \dfrac{\dfrac{\vdots \qquad\qquad \vdots}{\Theta_2, A, !C^n \vdash \Delta} \qquad \dfrac{\vdots}{!\Theta_3 \vdash ?\Gamma, C}\,\mathbf{!S}}{\Theta_2, !\Theta_3, A \vdash \Delta, ?\Gamma}\,\mathbf{Cut}*}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma}\,\mathbf{Cut}*$$

In the more complex case, when the cut-formulas descend from both hypotheses of

following reduction:

$$\dfrac{\dfrac{\vdots}{\Theta_1, !A^{n-1}, A \vdash \Sigma}\,{}_{!\mathbf{D}} \qquad \dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}}{\dfrac{\Theta_1, !A^n \vdash \Sigma \qquad\qquad !\Theta_2 \vdash ?\Gamma, !A}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,{}_{\mathbf{Cut!}}}\,{}^{!\mathbf{S}}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1, !A^{n-1}, A \vdash \Sigma} \qquad \dfrac{\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}}{!\Theta_2 \vdash ?\Gamma, !A}\,{}^{!\mathbf{S}}}{\Theta_1, !\Theta_2, A \vdash \Sigma, ?\Gamma}\,{}_{\mathbf{Cut!}} \qquad \dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}}{\dfrac{\Theta_1, !\Theta_2, !\Theta_2 \vdash \Sigma, ?\Gamma, ?\Gamma}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,{}^{?\mathbf{C}}}\,{}_{!\mathbf{C}}\,{}_{\mathbf{Cut}}$$

Note that the second case requires the duplication of the proof above the application of **!S**. Since A has fewer symbols than $!A$, the lower **Cut** in the second case is of degree smaller than d. By induction, we may assume that the upper application of **Cut!** is reducible in degree.

!S versus !S

$$\dfrac{\dfrac{\vdots}{!\Theta_1, !A \vdash ?\Sigma, B} \qquad \dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}}{\dfrac{!\Theta_1, !A \vdash ?\Sigma, !B}{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, !B}\,{}_{\mathbf{Cut}*}}\,{}^{!\mathbf{S}} \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{!\Theta_1, A \vdash ?\Sigma, B} \qquad \dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}}{\dfrac{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, B}{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, !B}\,{}^{!\mathbf{S}}}\,{}_{\mathbf{Cut}*}$$

Here we appeal to the induction hypothesis to produce a proof degree less than d of $!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, B$, and then construct the desired proof from that.

!D versus !S

As for the previous **!W** versus **!S** case, here we have two cases, depending on whether the **Cut**$*$ in question eliminates more than one occurrence of the cut-formula from the derelicted (**!D**) sequent. Again, informally, the two cases turn on the size of the package. If there is only one thing in the package, we simple make use of it, and throw away the wrapping. If there are more thing in the package, we take one out, and move the smaller package along its way.

In the first case, the cut eliminates the one occurrence of the cut-formula introduced by the **!D** rule, and thus the following reduction applies:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1, A \vdash \Sigma\end{array}}{\Theta_1, !A \vdash \Sigma}\,\mathbf{!D} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{!\Theta_2 \vdash ?\Gamma, !A}\,\mathbf{!S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}*$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Theta_1, A \vdash \Sigma\end{array} \qquad \begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}$$

However, in the second case, where the cut is actually a **Cut!** and eliminates more than one occurrence of the cut-formula from the derelicted sequent, we perform the

introduced by the **!W** rule, and thus this application of cut may be eliminated entirely:

$$\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma}\quad\dfrac{\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A^{\perp}}}{\Theta_1, !A \vdash \Sigma}\,{}^{!\mathbf{W}}\quad\dfrac{}{!\Theta_2 \vdash ?\Gamma, !A}\,{}^{!\mathbf{S}}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}* \quad\Longrightarrow\quad \dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma}\,{}^{!\mathbf{W}}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,{}^{?\mathbf{W}}$$

However, the second possibility, where the **Cut**$*$ is actually a **Cut**! and eliminates more than one occurrence of the cut-formula from the weakened sequent, we perform the following reduction:

$$\dfrac{\dfrac{\vdots}{\Theta_1(!A)^{n-1} \vdash \Sigma}\,{}^{!\mathbf{W}}}{\dfrac{\Theta_1(!A)^{n} \vdash \Sigma}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}!}\quad\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}\,{}^{!\mathbf{S}} \quad\Longrightarrow\quad \dfrac{\dfrac{\vdots}{\Theta_1, (!A)^{n-1} \vdash \Sigma}\quad\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}\,{}^{!\mathbf{S}}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}!$$

In the first possibility we have our result immediately, since the **Cut**$*$ has been eliminated. In the second possibility, we appeal to the induction hypothesis.

The ?**W** versus ?**S** case is similar.

!C versus !S

In this case we make critical use of the **Cut**! rule. Without this extra rule of inference this reduction is especially difficult to formulate correctly, and the induction required is complicated.

$$\dfrac{\dfrac{\vdots}{\Theta_1, !A, !A \vdash \Sigma}\,{}^{!\mathbf{C}}}{\dfrac{\Theta_1, !A \vdash \Sigma}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}*}\quad\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}\,{}^{!\mathbf{S}} \quad\Longrightarrow\quad \dfrac{\dfrac{\vdots}{\Theta_1, !A, !A \vdash \Sigma}\quad\dfrac{\vdots}{!\Theta_2 \vdash ?\Gamma, A}\,{}^{!\mathbf{S}}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,\mathbf{Cut}!$$

Here we know that the cut-formula begins with a !, and thus **Cut**! may apply to it. We thus produce a **Cut**! regardless of whether the original **Cut**$*$ was a **Cut** or a **Cut**!.

The ?**C** versus ?**S** case is similar.

& R versus & L

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta_1 \vdash \Sigma, A & \Theta_1 \vdash \Sigma, B\end{array}}{\Theta_1 \vdash \Sigma, (A \& B)}\,\&\mathbf{R} \qquad \dfrac{\begin{array}{c}\vdots \\ \Theta_2, A \vdash \Gamma\end{array}}{\Theta_2, (A \& B) \vdash \Gamma}\,\&L}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}\,\mathbf{Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta_1 \vdash \Sigma, A & \Theta_2, A \vdash \Gamma\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}\,\mathbf{Cut}$$

The symmetric case of & L is similar, as are the two cases of $\oplus$ R versus $\oplus$ L, all of which are omitted. We need not appeal to the induction hypothesis, and the cut-formula does not begin with ?, and thus we know that **Cut**? does not apply, and similarly for ! and the **Cut**! rule.

!W versus !S

For this and subsequent cases involving **!S** and **?S**, 'packaging' is a useful analogy. We build packages containing a number of contractions and a single **Cut**! when we reduce principal cases involving **!C** versus **!S**. We shrink the package in cases of **!W** versus **!S**, and we actually use the contents of the package as cases of **!D** versus **!S**. We let packages pass by each other at cases of **!S** versus **!S**, and at cases of **Cut**! versus **!S** and of $\otimes$ versus **!S** we break one package into two. This same intuition applies to the dual case involving ?.

For this case, **!W** versus **!S**, there are two possibilities, depending on whether the cut in question eliminates more than one occurrence of the cut-formula from the weakened sequent. Informally, the possibilities turn on whether there is only one thing in the package. If so, we don't need the package. If there are more things in the package, we shrink the package.

In the first possibility, the cut eliminates the one occurrence of the cut-formula

I versus any

If the last rule applied in either hypothesis is **I** (identity), then regardless of the rule applied in the other hypothesis we may remove the cut, and the application of identity:

$$\dfrac{\dfrac{}{p_i \vdash p_i}\mathbf{I} \qquad \begin{matrix}\vdots\\ \Theta, p_i \vdash \Gamma\end{matrix}}{\Theta, p_i \vdash \Gamma}\mathbf{Cut} \Longrightarrow \qquad \begin{matrix}\vdots\\ \Theta, p_i \vdash \Gamma\end{matrix}$$

The symmetric case (where p_i appears on the right in the non-axiom branch of the proof) is similar. Note that the identity axiom only applies to atomic propositions, and thus we know that **Cut**! and **Cut**? are inapplicable.

$\otimes$ R versus $\otimes$ L

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \otimes B), \Delta}\otimes\mathbf{R} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_3, A, B \vdash \Gamma\end{matrix}}{\Theta_3, (A \otimes B) \vdash \Gamma}\otimes L}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Gamma, \Delta}\mathbf{Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_3, A, B \vdash \Gamma\end{matrix}}{\Theta_2, \Theta_3, B \vdash \Gamma, \Delta}\mathbf{Cut}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Gamma, \Delta, \Sigma}\mathbf{Cut}$$

In this case, as in most of the principal formula cut-elimination steps, we need not appeal to the induction hypothesis of this lemma. We have eliminated the **Cut** of degree d, and replaced it with two applications of **Cut** of degree smaller than d.

The case of $\parr$ R versus $\parr$ L is similar to this one, and is omitted.

Section 2.3.2. Otherwise, we know the formula does not begin with ? on the right or ! on the left, and thus the lower **Cut**$*$ must actually be **Cut**.

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\Theta_1 \vdash \Sigma, A, C \quad \Theta_2, C \vdash \Gamma}\textbf{Cut}* \qquad \vdots}{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma \qquad\qquad \Theta_3 \vdash \Delta, A^\perp}\textbf{Cut}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots \quad \vdots}{\Theta_1 \vdash \Sigma, A, C \quad \Theta_3 \vdash \Delta, A^\perp}\textbf{Cut} \qquad \vdots}{\Theta_1, \Theta_3 \vdash \Sigma, \Delta, C \qquad\qquad \Theta_2, C \vdash \Gamma}\textbf{Cut}*}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma}$$

Here we know that the number of symbols in the formula A is d, and the number of symbols in the formula C is less than d. Thus by induction we know that we can construct a proof of degree less than d of $\vdash \Sigma, \Delta, C$, and from that we can construct our desired proof of $\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma$.

2.3.2 Cut of principal formulas

If the proof of each hypothesis ends in a rule with the cut formula as its principal formula, then the two last rules above the cut must be one of these combinations: **I** versus any, $\otimes$ **R** versus $\otimes$ **L**, $\parr$ **R** versus $\parr$ **L**, & **R** versus & **L**, $\oplus$ **R** versus $\oplus$ **L**, ?**W** versus !**S**, ?**C** versus !**S**, ?**D** versus !**S**, !**S** versus !**S**, **Cut**$*$ versus !**S**, $\otimes$ versus !**S**, or $\perp$ versus 1. Since all formulas in the conclusion of !**S** are considered principal, the analysis of !**S** at this stage of the proof is rather complex.

In many of these cases, we know that the **Cut**! and **Cut**? rules are inapplicable, since the cut-formula has just been introduced, and it does not begin with a ! or ?. When we know this, we will disambiguate the reduction, and show the applications of **Cut**, **Cut**? and **Cut**! separately.

?C or !C

$$\dfrac{\dfrac{\vdots}{\dfrac{\Theta_1 \vdash \Sigma, A, ?B, ?B}{\Theta_1 \vdash \Sigma, A, ?B}\,?\mathbf{C}} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, ?B, ?B} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\dfrac{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B, ?B}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,?\mathbf{C}}\,\mathbf{Cut}*$$

?D or !D

$$\dfrac{\dfrac{\vdots}{\dfrac{\Theta_1 \vdash \Sigma, A, B}{\Theta_1 \vdash \Sigma, A, ?B}\,?\mathbf{D}} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, B} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\dfrac{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, B}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,?\mathbf{D}}\,\mathbf{Cut}*$$

$\perp$ R or 1 L

$$\dfrac{\dfrac{\vdots}{\dfrac{\Theta_1 \vdash \Sigma, A}{\Theta_1 \vdash \Sigma, A, \perp}\,\perp\mathbf{R}} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \perp}\,\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\dfrac{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \perp}\,\perp\mathbf{R}}\,\mathbf{Cut}*$$

$\top$ R or 0 L

$$\dfrac{\dfrac{}{\Theta_1 \vdash \Sigma, A, \top}\,\top\mathbf{R} \quad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^\perp}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \top}\,\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \top}\,\top\mathbf{R}$$

Cut

If the proof of one hypothesis ends in **Cut**$*$, then we know that it has degree less than d, by the hypothesis of this lemma. If the cut-formula of the lower degree d application of **Cut**$*$ begins with ? on the right or ! on the left, then it is considered principal (by definition) in the upper application of **Cut**$*$, and will be handled in

& R or $\oplus$ L

It is the elimination of this type of cut (among others) which may lead to an exponential blowup in the size of cut-free proofs.

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, C} \quad \dfrac{\vdots}{\Theta_1 \vdash \Sigma, B, C}}{\Theta_1 \vdash \Sigma, (A\&B), C}\,\&\mathbf{R} \qquad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A\&B), \Gamma}\,\mathbf{Cut}*$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A, C\end{matrix} \quad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma}\,\mathbf{Cut}* \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, B, C\end{matrix} \quad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, B, \Gamma}\,\mathbf{Cut}*}{\Theta_1, \Theta_2 \vdash \Sigma, (A\&B), \Gamma}\,\&\mathbf{R}$$

The increase in proof size comes from replicating the entire proof tree above $\Theta_2, C \vdash \Gamma$. Note that even though there are now two cuts instead of one, we may assume that both may be reduced in degree to less than d by induction on the size of the derivations. That is, there are fewer proof rules applied above each **Cut**$*$ than there were above the single application of **Cut**$*$ originally.

?W or !W

For this and the remaining cases, we omit discussion and simply indicate the reduction:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix}}{\Theta_1 \vdash \Sigma, A, ?B}\,?\mathbf{W} \quad \begin{matrix}\vdots\\ \Theta_2 \vdash \Gamma, A^{\perp}\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \quad \begin{matrix}\vdots\\ \Theta_2 \vdash \Gamma, A^{\perp}\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}\,\mathbf{Cut}*}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}\,?\mathbf{W}$$

$\parr$ R or $\otimes$ L

If the last rule applied in one hypothesis is $\parr$ R, and the cut-formula is not the main formula introduced by that application of $\parr$ R, then we may propagate the **Cut**$*$ upward, through the application of $\parr$ R:

$$\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, B, C}\,\dfrac{}{\Theta_1 \vdash \Sigma, (A \parr B), C}\parr\mathbf{R} \quad \begin{array}{c}\vdots\\ \Theta_2, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \parr B), \Gamma}\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\begin{array}{cc}\vdots & \vdots\\ \Theta_1 \vdash \Sigma, A, B, C & \Theta_2, C \vdash \Gamma\end{array}}{\dfrac{\Theta_1, \Theta_2 \vdash \Sigma, A, B, \Gamma}{\Theta_1, \Theta_2 \vdash \Sigma, (A \parr B), \Gamma}\parr\mathbf{R}}\mathbf{Cut}*$$

Again, the proof above the **Cut**$*$ is smaller after this transformation, and thus by induction we have our result.

$\oplus$ R or & L

Applications of **Cut**$*$ involving the two symmetric $\oplus$ R rules (where the cut-formula is not principle, that is, not introduced by this application of $\oplus$ R) may be eliminated in similar ways:

$$\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, C}\,\dfrac{}{\Theta_1 \vdash \Sigma, (A \oplus B), C}\oplus\mathbf{R} \quad \begin{array}{c}\vdots\\ \Theta_2, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \oplus B), \Gamma}\mathbf{Cut}* \quad \Longrightarrow \quad \dfrac{\begin{array}{cc}\vdots & \vdots\\ \Theta_1 \vdash \Sigma, A, C & \Theta_2, C \vdash \Gamma\end{array}}{\dfrac{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma}{\Theta_1, \Theta_2 \vdash \Sigma, (A \oplus B), \Gamma}\oplus\mathbf{R}}\mathbf{Cut}*$$

The second case of this rule is the same except the conclusion would contain the formula $(B \oplus A)$, instead of the formula $(A \oplus B)$ seen above.

with ?, then we may propagate the **Cut**$*$ upward, through the application of $\otimes$:

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta_1 \vdash \Sigma, A & \Theta_2 \vdash B, \Delta, C\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \otimes B), \Delta, C}\otimes\mathbf{R} \qquad \begin{array}{c}\vdots \\ \Theta_3, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (A \otimes B), \Delta, \Gamma}\mathbf{Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots \\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \dfrac{\begin{array}{cc}\vdots & \vdots \\ \Theta_2 \vdash B, \Delta, C & \Theta_3, C \vdash \Gamma\end{array}}{\Theta_2, \Theta_3 \vdash B, \Delta, \Gamma}\mathbf{Cut}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (A \otimes B), \Delta, \Gamma}\otimes\mathbf{R}$$

For the rules such as $\otimes$ R with two hypotheses, we give the reduction for the case where the non-principal cut-formula appears in the right hand hypothesis of the $\otimes$ rule, and appears in one specific position in that sequent. The symmetric case of the cut-formula appearing in the left hypothesis is very similar, and is always omitted. Since exchange is considered built-in to the system, sequents are considered multisets. Thus the exact position of formulas in sequents is unimportant. (Note that in noncommutative linear logic the relative position becomes vitally important.)

The proof ending in **Cut** after this transformation is smaller than the original proof, since the entire proof of $\Theta_1 \vdash \Sigma, A$, and the last application of $\otimes$ R are no longer above the **Cut**. Thus by induction on the size of proofs, we can construct the desired proof of degree less than d.

Note that the **Cut**! and **Cut**? rule only applies to formulas which begin with ! or ?, and thus this reduction, which is only used if the cut-formula does not begin with ? or !, applies only to **Cut** and not to **Cut**! or **Cut**?. Thus, we have disambiguated this case, and write only **Cut**. Tranformations are later presented in terms of **Cut**$*$, in order to cover all three possibilities simultaneously. The reductions given later (in Section 2.3.2) handle the case of **Cut**! and **Cut**?.

1. The cut-formula is *not* principal in one or both hypotheses.

2. The cut-formula *is* principal in both hypotheses.

In each case we will provide a reduction, which may eliminate the cut entirely, or replace it with one or two smaller cuts. Since this is a proof by induction on the size of a derivation, one may view this proof as a procedure which pushes applications of **Cut**$*$ of large degree up a derivation. Informally, this procedure pushes applications of **Cut**$*$ up through proof rules where the cut-formula is non-principal, until the critical point is reached where the cut-formula is principal in both hypotheses. In Girard's proof of cut-elimination for linear logic using proof nets, the non-principal cases are circumvented by following proof links. In both approaches, however, the principal cases require significant detailed analysis.

2.3.1 Cut of non-principal formulas

If the derivation of a hypothesis ends in a rule yielding a non-principal cut-formula, then the rule must be one of the following: $\otimes$ R, $\otimes$ L, $\parr$ R, $\parr$ L, $\oplus$ R, $\oplus$ L, & R, & L, ?**W**, ?**C**, ?**D**, !**W**, !**C**, !**D**, $\bot$ R, 1 L, $\top$ R, 0 L, or **Cut**$*$. The rules **I**, !**S**, ?**S**, $\bot$ L, and 1 R are absent from this list since those rules have no non-principal formulas in their conclusions. The later analysis of principal formula cuts considers these three cases. Also, most of the following cases come in two directly analogous cases, such as $\otimes$ R vs $\parr$ L. We will only present one of each such pair of cases.

$\otimes$ R, or $\parr$ L

If the last rule applied in one hypothesis is $\otimes$ **R**, the cut-formula is not the main formula introduced by that application of $\otimes$ **R**, and the cut-formula does not begin

Recall that the *principal formula* of an application of an inference rule is defined to be any formula which is introduced by that rule. For convenience, we extend the notion of principal formula in the following nonstandard ways. We will consider any formula beginning with ? appearing on the right side, and any formulas prefixed with ! on the left of the conclusion of the **!S**, $\otimes$ R, $\parr$ L, $\multimap$ L, or **Cut**$*$ rules to be principal. By this definition all formulas in the conclusion of **!S** are principal, and the only rule in which a formula beginning with ! may be principal on the right hand side is **!S**. This definition of principal formula simplifies the structure of the following proof somewhat.

Operationally, the cut-elimination procedure defined below first finds one of the "highest" cuts of maximal degree in the proof. That is, an application of **Cut**$*$ for which all applications of **Cut**$*$ in the derivation of either hypothesis is of smaller degree. Then a reduction is applied to that occurrence of **Cut**$*$, which simplifies or eliminates it, although it may replicate some other portions of the original proof. We iterate this procedure to remove all cuts of some degree, and then iterate the entire procedure to eliminate all cuts. In this way, any linear logic proof may be normalized into one without any uses of the **Cut**, **Cut**? or **Cut**! rules, at the possible expense of an (worse than) exponential blowup in the size of the resulting proof tree.

Technically, we begin with a lemma which constitutes the heart of the proof of cut-elimination. Although the proof of this lemma is rather lengthy, the reasoning is straightforward, and the remainder of the proof of cut-elimination is quite simple.

Lemma 2.3.2 (Reduce One Cut) *Given a proof of the sequent* $\Theta \vdash \Gamma$ *in linear logic which ends in an application of* **Cut**$*$ *of degree* $d > 0$*, and where the degree of the proofs of both hypothesis is less than* d*, we may construct a proof of* $\Theta \vdash \Gamma$ *in linear logic of degree less than* d*.*

Proof. By induction on the number of proof rules applied in the derivation of $\Theta \vdash \Gamma$.

Given a derivation which ends in a **Cut**$*$, we perform case analysis on the rules which were applied immediately above the **Cut**$*$. One of the following cases must apply to any such derivation:

$$\textbf{Cut?} \quad \frac{\Theta \vdash \Sigma, (?A)^n \qquad \Gamma, ?A \vdash \Delta}{\Theta, \Gamma \vdash \Sigma, \Delta} \quad n \geq 1$$

As mentioned in the last section, $(?A)^n$ denotes a multiset of formulas. For example, $(?A)^3 = ?A, ?A, ?A$. As stated in the side condition, the **Cut!** and **Cut?** rules are only applicable when n is at least 1.

We will use the symbol "**Cut**$*$" as a general term for the original **Cut** rule or the new **Cut!** and **Cut?** rules ambiguously. These new rules of inference are derivable; they may be simulated by several applications of contraction (**!C** or **?C**) and one application of the standard **Cut** rule. The original **Cut** rule coincides with **Cut!** (or **Cut?**) when $n = 1$. Adding these extra derived rules of inference simplifies the termination argument substantially by packaging together some number of contractions with the cut that eliminates the contracted formula. This package is only opened when the contracted formulas are actually used with the application of the **!D** or **?D** rules, thrown away by the **!W** or **?W** rules, or split into two packages by the $\otimes$ R, $\parr$ L, $\multimap$ L, **Cut**$*$ rules.

We will call a formula which appears in a hypothesis of an application of **Cut**$*$, but which does not occur in the conclusion a *cut-formula*. In the list of linear logic rules in Appendix A the cut-formula in the **Cut** rule is the formula named A, and in the **Cut!** and **Cut?** rules above, the cut-formulas are $!A$ and $?A$.

We also define the *degree* of a **Cut**$*$ to be the number of symbols in its cut-formula. For concreteness, we define here what is meant by number of symbols. We will consider each propositional symbol p_i to be a single symbol. We also consider the negation of each propositional symbol ${p_i}^\perp$ to be a single symbol. Finally, we count each connective and constant, $\otimes, \parr, \oplus, \&, ?, !, 1, \perp, 0, \top$, as a single symbol, but do not count parentheses. It is important to note that negation is defined, and therefore is not a connective. This method of accounting has the pleasant property that any linear logic formula A and its negation $A^\perp$ have exactly the same number of symbols. (One may prove this by induction on the structure of the formula A). Thus it does not matter which cut-formula we count when determining the degree of a cut. We also define the *degree* of a proof to be the maximum degree of any cut in the proof, or zero if there are no cuts.

2.3 Cut Elimination

The cut-elimination theorem, in general, states that whatever can be proven in the full version of a logic may also be proven without the use of the cut rule. This theorem is fundamental to linear logic, and was proven by Girard shortly after the introduction of the logic by presenting a cut-elimination procedure for proof nets [30]. Since Girard demonstrated the correspondence between proof nets and the sequent calculus presentation of linear logic, we could have relied on Girard's proof of cut-elimination. However, in later proofs, we make use of the syntax and exact form of a cut-elimination procedure for the sequent calculus formulation of linear logic. Girard's use of proof nets, and his reliance on the one-sided version of linear logic complicates the construction of our later theorems. Thus a full cut-elimination proof is given here.

The following demonstration of the cut-elimination theorem consists of a linear logic proof normalization procedure which slowly eliminates cuts from any linear logic proof. The procedure may greatly increase the size of the proof, although of course it will still be a proof of the same sequent. For technical reasons, we add derived rules of inference, **Cut!** and **Cut?**, which simplify the proof of termination. We then give a set of reductions which apply to proofs which end in **Cut**, **Cut!**, or **Cut?**, and using these we eliminate all uses of **Cut**, **Cut!**, and **Cut?** from a proof.

The proof structure is very close to the well known proofs of cut-elimination in classical logic [36], but is complicated by the extra information which must be preserved in a linear proof. The **Cut!** and **Cut?** rules defined below are reminiscent of Gentzen's MIX rule [29], and serve the same purpose, which is to package together certain inference rules. As in Gentzen's work, we add these extra rules, and then show that they (along with **Cut**) may be eliminated entirely from any proof. Thus we show that these new rules and **Cut** are redundant in linear logic.

Let us begin with some definitions. First, we define the following new rules of inference,

$$\textbf{Cut!} \quad \frac{\Theta \vdash \Sigma, !A \qquad \Gamma, (!A)^n \vdash \Delta}{\Theta, \Gamma \vdash \Sigma, \Delta} \quad n \geq 1$$

For a time it was conjectured that classical linear logic is conservative over ILL for formulas without the $\bot$ constant, but the following counterexample, due to Harold Schellinx, shows there to be other cases of non-conservativity as well: $((A \multimap B) \multimap 0) \multimap (A \otimes \top)$.

However, conservativity does hold for fragments of ILL which do not include either false constant ($\bot$ or 0), and conservativity also holds for fragments of ILL without $\multimap$.

Theorem 2.2.1 (Conservativity) CLL *is conservative over* ILL *for sequents not containing* $\bot$ *or* 0. CLL *is also conservative over* ILL *for sequents not containing* $\multimap$.

This theorem may be proven by induction on the assumed cut-free CLL proofs. Essentially, one can show that any cut-free CLL proof of a sequent meeting the above restrictions satisfies the intuitionistic restriction to at most one formula on the right throughout. The only interesting inductive steps are for the false constants and $\multimap$ L. In the first statement of the proof, if a proof ends in $\multimap$ L, then either the CLL application of $\multimap$ L matches the intuitionistic rule for $\multimap$ L, in which case we can immediately appeal to the induction hypothesis, or one of the hypotheses has an empty consequent, or right hand side of the $\vdash$. In the later case, one appeals to the property that for sequents satisfying the above syntactic constraints, no sequent $\Gamma \vdash$ is provable in CLL.

The bottom line here is that classical linear logic CLL is conservative over intuitionistic linear logic ILL only if one *completely* eliminates negation. Negation is first present in the rules for $^\bot$ which allow formulas to move from one side of a sequent to another. For example, the formula $(A^\bot \multimap B^\bot) \multimap (B \multimap A)$ is provable in CLL (utilizing $^\bot$) but not in ILL. Negation may also be implicitly recovered with the use of $\multimap$ and either 0 or $\bot$ (the two flavors of *false*). Conservativity holds if none of these kinds of negation arise, which may be ensured by eliminating $^\bot$ and $\multimap$, or $^\bot$ and 0 and $\bot$.

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Sigma \vdash A \qquad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$	**Cut**
E Left	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C}$		
$\otimes$**L**	$\dfrac{\Sigma, A, B \vdash C}{\Sigma, (A \otimes B) \vdash C}$	$\dfrac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}$	$\otimes$**R**
$\multimap$**L**	$\dfrac{\Sigma \vdash A \qquad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \multimap B) \vdash C}$	$\dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}$	$\multimap$**R**
$\oplus$**L**	$\dfrac{\Sigma, A \vdash C \qquad \Sigma, B \vdash C}{\Sigma, (A \oplus B) \vdash C}$	$\dfrac{\Sigma \vdash A \qquad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}$	**&R**
& L1	$\dfrac{\Sigma, A \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R1**
& L2	$\dfrac{\Sigma, B \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R2**
!WL	$\dfrac{\Sigma \vdash A}{\Sigma, !B \vdash A}$		
!DL	$\dfrac{\Sigma, A \vdash B}{\Sigma, !A \vdash B}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash ?A}$	**?DR**
!CL	$\dfrac{\Sigma, !A, !A \vdash B}{\Sigma, !A \vdash B}$		
?SL	$\dfrac{!\Sigma, A \vdash ?B}{!\Sigma, ?A \vdash ?B}$	$\dfrac{!\Sigma \vdash A}{!\Sigma \vdash !A}$	**!SR**
1**L**	$\dfrac{\Sigma \vdash A}{\Sigma, 1 \vdash A}$	$\dfrac{}{\vdash 1}$	1**R**
0**L**	$\dfrac{}{\Sigma, 0 \vdash A}$	$\dfrac{}{\Sigma \vdash \top}$	$\top$**R**

Figure 2.1: Sequent Calculus Rules for Intuitionistic Linear Logic (ILL)

in Figure 1.1, as *classical* linear logic, or CLL, to emphasize the distinction between intuitionistic and classical versions of linear logic.

The Gentzen-style inference rules for intuitionistic linear logic (ILL) given below in figure 2.1 are reproduced in Appendix B for convenience [4].

Note that the rule for $\perp$ R, among others, is severely restricted in the intuitionistic calculus.

A somewhat unexpected result is that CLL is not conservative over ILL, even for conclusion sequents which satisfy the intuitionistic restriction of one formula on the right hand side. For example, consider the formula $(((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))$. In considering a cut-free proof of this formula, we have no choice but to construct the following partial proof:

$$\dfrac{\begin{array}{c}\vdots\\ ((A \multimap (B \multimap \perp)) \multimap \perp) \vdash A \otimes B\end{array}}{\vdash (((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))}\multimap R$$

In CLL, one can complete the proof as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}I \qquad \dfrac{}{B \vdash B}I}{A, B \vdash A \otimes B}\otimes R}{A, B \vdash \perp, A \otimes B}\perp R}{A \vdash (B \multimap \perp), A \otimes B}\multimap R}{\vdash (A \multimap (B \multimap \perp)), A \otimes B}\multimap R \qquad \dfrac{}{\perp \vdash}}{((A \multimap (B \multimap \perp)) \multimap \perp) \vdash A \otimes B}\multimap L}{\vdash (((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))}\multimap R$$

However, in ILL the critical $\multimap$ L inference is not available, and it can be shown that this sequent is not provable in ILL. The same problem arises with the simpler example $(((A \multimap \perp) \multimap \perp) \multimap A)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}I}{A \vdash \perp, A}\perp R}{\vdash (A \multimap \perp), A}\multimap R \qquad \dfrac{}{\perp \vdash}}{((A \multimap \perp) \multimap \perp) \vdash A}\multimap L}{\vdash (((A \multimap \perp) \multimap \perp) \multimap A)}\multimap R$$

to be constructed out of *multisets* of formulas on each side of the $\vdash$. This alteration to the sequent calculus is similar to the standard use of *sets* on both sides of a $\vdash$ in classical and intuitionistic logic, but in linear logic the multiplicity of formulas is of crucial importance, while it is completely unimportant in classical logic. Assuming that sequents are built from multisets rather than sequences of formulas, one simply ignores all applications of the exchange rule. For the majority of this thesis we will make this assumption, although when noncommutative linear logic is considered a great deal more care will be taken with such matters.

As is relatively standard, C^n will be used to indicate a sequence of n C's, separated by commas, as follows:

$$C^n \triangleq \overbrace{C, C, \cdots, C}^{n}$$

Since $p^\perp$ is an atomic symbol, the notation $p^{\perp\,3}$ will be used for $(p^\perp)^3$, which is simply $p^\perp, p^\perp, p^\perp$.

We define a *positive* literal to be one of the given propositional symbols p_i or $p_i^\perp$ which occurs with positive polarity. A *negative* literal is one of these symbols which occurs with negative polarity.

The class of *subformulas* of a given formula or sequent is defined by the following: A is a subformula of A. If A is a subformula of B, then A is also a subformula of the following formulas: $?B$, $!B$, $B \otimes C$, $C \otimes B$, $B \parr C$, $C \parr B$, $B \multimap C$, and $C \multimap B$. Also, $A\{t/x\}$ for any t is a subformula of $\forall x A$ and $\exists x A$. If A is a subformula of B, then A is also a subformula of the sequents $\Sigma \vdash \Gamma_1, B, \Gamma_2$ and $\Sigma_1, B, \Sigma_2 \vdash \Gamma$.

2.2 Intuitionistic Linear Logic

The intuitionistic version of linear logic (ILL) is generated by restricting sequents to include at most formula on the right hand side of the $\vdash$, and removing the rules for $^\perp$ which allow formulas to move from one side of a sequent to another. Standard intuitionistic logic may also be developed from classical logic in a similar way. In linear logic, however, there are some interesting twists which must be considered. To be clear, for the remainder of this section we will refer to linear logic, as defined above

Chapter 2

Linear Proof Theory

Below is a brief presentation of the more foundational concepts of linear logic and proof theory necessary for the remainder of this thesis. The reader wishing more detailed discussion of this introductory material is referred to [29, 51, 30].

2.1 Basic Properties of Linear Logic

As mentioned earlier, because linear logic has an involutive negation, $(A^{\perp})^{\perp} = A$, one may formulate linear logic as a one-sided system. Informally, one may consider the $^{\perp}$R and $^{\perp}$L rules to be "built in" to the system as if these were structural rules. In this case, many rules are identified: $\otimes$R and $\parr$L become identical rules. One translates the sequent $\Sigma \vdash \Gamma$ into an equivalent sequent $\vdash (\Sigma)^{\perp}, \Gamma$, where the negation of a sequence of formulas is the sequence of their negations (in reverse order, if order is important).

One may then work in a one-sided system where all formulas appear on the right of the $\vdash$, and there are no sequent rules for connectives on the left of the $\vdash$. Some of the proofs by case analysis of the sequent calculus rules are greatly simplified by this, although there is some price for this convenience in the difficulty of comprehending complicated sequent proofs in the one sided system. This thesis will use both one sided and two sided sequents, as is convenient for the topic under consideration.

Also, because linear logic contains the exchange rule, one may consider sequents

to the development of this vision.

Two results of this thesis contribute to this propositions-as-types approach to studying linear logic. First, there is a language based on linear logic which enforces a certain economy of variable uses: each variable must appear exactly twice — once in definition, and once in use. Second, there is a two-space memory model which allows a compiler to generate efficient code in some cases for the linear language. One space contains linear objects, with exactly one pointer to them, and the other space contains all other objects. There are relatively straightforward modifications to the Tim, or three instruction machine which allow it to implement the linear language with this two-space memory model. Together these advances in the development of the propositions-as-types computational interpretation of linear logic add more evidence that certain resource issues in programming languages can be understood from a linear perspective. However, these results are merely stepping stones to a larger understanding of the logical basis of practically important implementation techniques.

system from a sequent system are due to Prawitz [82].

Showing that the two systems are equivalent, we immediately get a cross-fertilization of theorems between the two systems. Perhaps the most important such theorem is type soundness, or the subject-reduction theorem. This theorem states that if a term has a derivable typing, then the result of reducing the term still has the same type. If this theorem failed, there would be little reason to call a logical framework a "type system", since terms might have a certain type, but after reduction the terms would not have this type. Another important theorem proved in this thesis is the existence of a (unique) most general type for any linear term.

Looking at the systems informally, they are different proof systems for the same logic. The sequent system is conservative with respect to types: any type appearing anywhere in a cut-free proof appears as a subformula of the type of the conclusion. On the other hand, the natural deduction system is conservative with respect to terms: any term appearing anywhere in a cut-free proof appears as a subformula of the term in the conclusion. Both of these properties follow immediately from the cut-elimination theorems.

Implementation

The vision here is of a language of the style of ML [74], but with a more detailed type system. Most current compilers for ML perform type checking, and then discard all the type information before actually compiling the program. Type checking allows the ML compiler to dispense with the run-time checks common to other functional language implementations, such as verifying that the arguments to a call to the function + are in fact numbers.

The guiding vision of this research program has been the development of a type system based on linear logic which would lend practical, useful advice to an ML-style compiler. The main idea of studying such an application of linear logic dates back to the introduction of linear logic and very early work [30, 53]. Others have also studied the further application of linear type information in the study of garbage collection and array update-in-place [21, 95]. In our investigations, we were led into considering decision problems and their complexity, but eventually these investigations led back

style, by assigning program constructors to inference rules. Thus an ML^{--} program can be seen as a notation for a linear logic proof, and the type of the program can be seen as the most general conclusion which can be inferred from the proof. Alternatively, one may view linear logic as a type system for ML^{--}, where linear formulas correspond to types.

Sequent versus Natural Deduction

The Curry-Howard correspondence holds between the lambda calculus and a natural deduction system for intuitionistic logic. However, the closest analog of natural deduction for linear logic are proof nets. There have been some attempts to use proof nets as a basis for forming proof terms, but proof nets suffer from some difficulties in representing full propositional linear logic including additives, constants, and modals. Attempts to use the "raw" sequent calculus for intuitionistic linear logic, for example by Abramsky [1], have shown some promise.

In this thesis, a new track is followed which shares properties of both of the above approaches: a sequent calculus presentation of a natural deduction system for intuitionistic linear logic is used as the basis for the proof terms. The presentation of natural deduction for intuitionistic logic in the sequent calculus dates back to some of Gentzen's original work on the sequent calculus. The proof system here is a result of straightforward application of those same ideas to intuitionistic linear logic.

The main idea is that a natural deduction-style introduction rule is represented by sequent calculus rule which introduces the formula on the right of the turnstile. All of the "right" rules of the sequent calculus are preserved unchanged in the natural deduction style sequent system. The "left" rules, however, are changed dramatically, becoming "elimination" rules in the modified system. These elimination rules are effectively simulatable in the sequent calculus with one application of the left rule, and one application of cut on the introduced formula. The "left" rules of the sequent calculus can be derived in the natural deduction system with one application of the elimination rule, one application of identity, and one application of cut. Therefore the same sequents (terms with typings) are provable in both systems, perhaps up to some applications of cut. Many of these ideas for creating a natural deduction style

power of linear logic. We are able to encode such thorny concepts as mutual exclusion very naturally in linear logic. This gives the strong impression that linear logic is certainly more basic and more expressive than intuitionistic logic.

The results of the previous sections immediately bring several "encoding" questions to mind: is it possible to encode other more standard logics in linear logic? In Girard's first paper on the subject, he gives encodings of intuitionistic and classical logic into full linear logic (using the modal operator !). However, full propositional linear logic is undecidable, while propositional intuitionistic logic is PSPACE-complete, as shown by Statman [86]. In fact, Statman shows that even the implicational fragment of intuitionistic logic is PSPACE-complete. Recalling the early results we see that the fragment of linear logic without the modals is exactly PSPACE-complete. This raises the question: is there a "logical" embedding of propositional intuitionistic logic into linear logic that does not make use of the modals?

The translation affirmatively answering this question given in Chapter 6 is an "asymmetrical interpretation." That is, occurrences of formulas of positive polarity are translated differently from negative occurrences [36]. The translation goes through an intermediate logic that is very similar to one employed by Hudelmaier [44] in the study of cut elimination in intuitionistic logic.

The translation itself makes use of various proof theoretic tricks which were developed in the previous sections. Essentially, intuitionistic implication is encoded as two connectives: linear implication and additive conjunction. An encoding based only on this is straightforward but unsound: a linear translation of an intuitionistic formula may be erroneously provable by violating the atomicity of (translated) intuitionistic implications. Such a proof would not be possible in intuitionistic logic, where effectively each pair must be acted on as one. A "lock and key" mechanism is therefore used to ensure mutual exclusion among translations of intuitionistic implications.

1.4.4 Linear ML^{--}

Chapter 7 concerns a functional language ML^{--} that is related to languages investigated by Lafont [53], Abramsky [1], Wadler [91, 93], Chirimar, Gunter, and Riecke [21], and others. The basic idea is to view linear logic in a Curry-Howard

that every formula that occurs in the conclusion is analyzed exactly once in the entirety of any cut-free proof. This gives an immediate linear bound on the number of inferences in any cut-free proof, showing that this logic is in NP, as one may simply guess and check an entire proof in linear space and time. Of course, this brings up the question of whether there is a polynomial decision algorithm for the multiplicatives, or if they are NP-hard. Previously, multiplicative affine (or direct) logic was shown to be NP-complete [59], but this proof relied on the use of weakening, a rule present in affine logic, but not in linear logic. Max Kanovich recently settled this problem by showing that the multiplicatives are NP-hard [47]. Chapter 5 presents an alternate proof of the NP-completeness. This proof is presented in order to facilitate the proof of the stronger result that even without propositions the multiplicatives are NP-complete. That is, constant-only multiplicative linear logic where formulas are built from only (multiplicative) and, or, true, and false ($\otimes$, $\parr$, 1, $\bot$) is NP-complete.

The key to the proof of NP-hardness of the multiplicatives is the existence of NP-complete problems that have the property that each entity in the problem must be used. In many problems such as SAT, if one proposition in a clause is true, the rest of the propositions do not matter, and may be either true or false. This "sloppy" behavior is very difficult to encode using only the multiplicatives. However, for certain NP-complete problems, such as 3-PARTITION, the combinatorial explosion arises from a partitioning of elements that must all be used in the end. Because of this property, 3-PARTITION can easily be encoded using just the multiplicative constants, thus demonstrating the NP-hardness of the linear (multiplicative) circuit evaluation problem.

1.4.3 Linearizing Intuitionistic Implication

There are two motivational points to be made about Chapter 6. First, Girard's original study of linear logic began with the decomposition of intuitionistic implication into a modal reuse operator and a linear implication. We analyze intuitionistic implication along a different cleavage plane, where it is broken down into its additive and multiplicative characters. Second, we provide another example of the expressive

some limited form of exchange the above encoding fails. Since there are many reasonable assumptions one could make about the exact form of noncommutative linear logic, we assume commuting exponentials, as have previous authors, including [97].

Again in the noncommutative case, the first order version is conservative over the propositional fragment, and thus first order noncommutative linear logic is also undecidable.

PSPACE-Completeness of Propositional Linear Logic Without Modals

Without the modal operators ? and ! there is an immediate proof search procedure for linear logic that is complete and always terminates. Therefore, linear logic without the modals is decidable. As mentioned earlier, this logic has the property that every formula which occurs in the conclusion is analyzed at most once along any branch of any cut-free proof. This gives an immediate linear bound on the depth of any cut-free proof. Thus an obvious decision procedure takes only polynomial space. This raises the question of whether the decision problem is PSPACE-hard. We show that one may encode quantified boolean formulas in propositional nonmodal linear logic, and one can also encode (normal) intuitionistic implication in nonmodal linear logic. Both of these logics are known to be PSPACE-hard. Thus the decision problem for propositional linear logic without modals is PSPACE-hard, and therefore PSPACE-complete.

As we will see, the encoding of these PSPACE-hard problems is not straightforward. In the naive encoding of quantified boolean, discussed in detail later, formulas encoding $\forall X, \exists Y, F$ and $\exists Y, \forall X, F$ would be the same. This lack of attention to quantifier order can be corrected with the use of "lock and keys", a kind of multiplicative guard used to enforce an ordering on quantifiers in the encoding of quantified Boolean formulas, and also used to provide a kind of mutual exclusion in the encoding of intuitionistic implication.

NP-Completeness of Multiplicative Fragment

Without the modals and without additive operators, linear logic is reduced to its core fragment, the so-called multiplicatives. Multiplicative linear logic has the property

the alternation can be encoded using the additives. However, this approach will not be explored in this thesis. Previously, Urquhart has demonstrated the undecidability of relevant implication [88]. However, that result does not bear on linear logic due to the requirement of a distributivity axiom that is not present in linear logic [89].

Non-Commutative Undecidability

The non-commutative variant of propositional linear logic is undecidable, even *without* the additive connectives that are used in the commutative case to encode zero-test. The intuition behind this result is that one *can* test natively in non-commutative logic for zero occurrences of a certain proposition *in a certain position* without resorting to an and-branching (or alternating) encoding. The proof of undecidability is by reduction from the (undecidable) word problem for non-commutative semigroups.

Although the proof of undecidability of non-commutative propositional linear logic stated later in this thesis is formulated in terms of Semi-Thue systems, it is perhaps easier to see this result informally in terms of standard Turing machines. One can encode a Turing machine tape as a sequence of propositions, one proposition for each tape symbol. The current position of the read/write head of the Turing machine, and its current state can be encoded as a single occurrence of a proposition at the appropriate location in the sequence of propositions corresponding to the Turing tape.

For example, $B, A, Q_i, B \vdash$ is a sequent which might encode a Turing machine with a tape with three non-blank symbols, currently in state Q_i, where the read/write head is currently over the symbol A. Transitions of the machine (the Turing machine program) can be encoded either as nonlogical theory axioms, or as modal implications. The formula $!((A \otimes Q_i) \multimap (Q_j \otimes B))$ may be read as: in state Q_i, if the head is over the symbol A, then erase the A, write a B, move the head to the right, and change to state Q_j.

It should be noted that in order for these results to hold, one must assume that either the modals of linear logic (for which contraction and weakening are applicable) commute with respect to any other formula, or that there is a new modal representing commutativity. In either case, there is an encoding of Turing machines, but without

intuitionistic logic into full first order linear logic [30]. However, most of the results described in this thesis involve the propositional fragment of linear logic. The first new result involves full propositional linear logic.

Undecidability

Since most propositional logics are decidable, it is reasonable, a priori, to expect that propositional linear logic would be decidable also. Further evidence for decidability is the cut-elimination theorem for linear logic. However, we prove that propositional linear logic is undecidable. That is, the set of valid (provable) sequents in propositional linear logic is not recursive.

The most obvious line of proof is to encode counter machines in the manner of [69]. However, the obvious encoding runs into a difficulty. While it is easy to encode the increment and decrement instructions in linear logic using the multiplicatives, the critical zero-test instruction of counter machines has no natural analog in linear logic. That is, there is no connective that would allow a proof to proceed when there are no occurrences of a certain proposition (i.e., zero-test succeeds), but perhaps any number of other propositions. However, linear logic does have the expressive power to encode such a zero-test by other means.

The linear logic feature used to encode zero test is the additive conjunction. This connective allows one to encode a certain type of and-branching (in the terminology of alternation [19]). Viewing the proof computationally, two branches are fired off, each of which must be provable in the *same context*. This sharing of contexts provided by the additive conjunction enables one to encode zero-test by enforcing that one branch verifies that the counter is zero and then terminates, while the other branch continues assuming that the counter was in fact zero. If the counter in question is not zero, the first "zero-checking" branch will not terminate successfully, preventing successful termination of the entire computation. This kind of "zero-checking" can be accomplished natively in the logic.

Another approach to describing this result is that the reachability problem for alternating (in the sense of [20]) Petri nets is undecidable. One can directly encode normal Petri net transitions using the multiplicatives and exponentials [8, 67], and

1.4 Overview of Thesis Results

This overview lists the most important results of this thesis and gives a glimpse of the proofs of the key theorems.

1.4.1 Basic Theorems

The most basic theorem of proof theory is the cut-elimination theorem. Most systems presented here enjoy a cut-elimination theorem, which simply states that any formula provable in a logic is also probable in that logic without the cut rule. Most proofs of cut-elimination in this thesis follow Gentzen's proof of the "Hauptsatz" [29]. The main idea is to first add a derived rule of inference that encapsulates the application of several inference rules. Then one eliminates all applications of cut and the derived rule together. The derived rules act as a bookkeeping trick that simplifies the proof of termination of cut-elimination. This theorem was proved for linear logic by Girard in [30] using an elegant proof notation called proof-nets, which are not used in this thesis. In Chapter 2 a full proof of cut-elimination in the sequent calculus is given in order to facilitate later proofs that depend on this form of proof of cut-elimination. Some formal systems in this thesis do not have a cut elimination theorem, for example those systems with nonlogical theories. Nevertheless, in these cases one may still find precise normal forms for proofs, even though all uses of cut cannot be eliminated.

Several results follow immediately from the cut elimination theorem. For instance, the subformula property states that any formula appearing anywhere in a cut-free proof of a sequent also appears in the conclusion sequent. Also, consistency of a logic follows from cut-elimination and the subformula property. Linear logic has the subformula property and is consistent.

1.4.2 Complexity Results

The complexity of the decision problem for various fragments of linear logic are summarized here. The full logic with first order quantifiers was known from the outset to be undecidable, as an immediate corollary of Girard's embedding of first order

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2}$	**Cut**
E Left	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Sigma}{\Gamma_1, B, A, \Gamma_2 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma_1, A, B, \Sigma_2}{\Gamma \vdash \Sigma_1, B, A, \Sigma_2}$	**E Right**
$\otimes$ **Left**	$\dfrac{\Gamma, A, B \vdash \Sigma}{\Gamma, (A \otimes B) \vdash \Sigma}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2 \vdash B, \Sigma_2}{\Gamma_1, \Gamma_2 \vdash (A \otimes B), \Sigma_1, \Sigma_2}$	$\otimes$ **Right**
$\parr$ **Left**	$\dfrac{\Gamma_1, A \vdash \Sigma_1 \quad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \parr B) \vdash \Sigma_1, \Sigma_2}$	$\dfrac{\Gamma \vdash A, B, \Sigma}{\Gamma \vdash (A \parr B), \Sigma}$	$\parr$ **Right**
$\oplus$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	$\&$ **Right**
$\&$ **Left1**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right1**
$\&$ **Left2**	$\dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right2**
! W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	**! C**
! D	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma}$	**! S**
? W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{\Gamma \vdash ?A, ?A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	**? C**
? D	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{!\Gamma, A \vdash ?\Sigma}{!\Gamma, ?A \vdash ?\Sigma}$	**? S**
$^\perp$ **Left**	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma, A^\perp \vdash \Sigma}$	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma \vdash A^\perp, \Sigma}$	$^\perp$ **Right**
0 **Left**	$\Gamma, 0 \vdash \Sigma$	$\Gamma \vdash \top, \Sigma$	$\top$ **Right**
1 **Left**	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, 1 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash \perp, \Sigma}$	$\perp$ **Right**
$\perp$ **Left**	$\perp \vdash$	$\vdash 1$	1 **Right**

Figure 1.1: Sequent Calculus Rules for Linear Logic

1.3.9 The Sequent Calculus of Linear Logic

The sequent rules for linear logic assume that the commas of the sequent are multiplicative. That is, a sequent $\Sigma \vdash \Gamma$ asserts that the multiplicative conjunction of the formulas in Σ together imply the multiplicative disjunction of the formulas in Δ.

The rules given in Figure 1.1 originally appeared in [30]. However, the rules presented here are for the two-sided sequent system, with formulas appearing on both sides of the $\vdash$. It has become somewhat standard now to present linear logic in a one-sided sequent system, by negating every formula which would have appeared on the left of the $\vdash$, and moving them to the right. One sided systems have the advantage of having half the proof rules as two sided systems for the same logic, but suffer from the disadvantage that sequents are harder for some to read.

The two-sided Gentzen-style inference rules for linear logic given in Figure 1.1 are reproduced in Appendix A for convenience.

It simplifies presentation to consider negation as defined, rather than being a connective. Here the symbol $\triangleq$ is used to denote "is defined as".

$$
\begin{array}{rcl@{\qquad}rcl}
(P_i)^\perp & \triangleq & P_i^\perp & (P_i^\perp)^\perp & \triangleq & P_i \\
(A \otimes B)^\perp & \triangleq & A \multimap B^\perp & (A \multimap B)^\perp & \triangleq & A \otimes B^\perp \\
(A \otimes B)^\perp & \triangleq & A^\perp \parr B^\perp & (A \parr B)^\perp & \triangleq & A^\perp \otimes B^\perp \\
(A \oplus B)^\perp & \triangleq & A^\perp \,\&\, B^\perp & (A \,\&\, B)^\perp & \triangleq & A^\perp \oplus B^\perp \\
(!A)^\perp & \triangleq & ?A^\perp & (?A)^\perp & \triangleq & !A^\perp \\
(1)^\perp & \triangleq & \bot & (\bot)^\perp & \triangleq & 1 \\
(0)^\perp & \triangleq & \top & (\top)^\perp & \triangleq & 0 \\
(\forall x.A)^\perp & \triangleq & \exists x.(A)^\perp & (\exists x.A)^\perp & \triangleq & \forall x.(A)^\perp
\end{array}
$$

We also define the linear implication connective $\multimap$ with the equation

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

and thus we omit explicit rules for the $\multimap$ connective, although many find the $\multimap$ connective much easier to comprehend than $\parr$.

Here are the precise sequent rules for linear !:

$$\textbf{! W}\ \frac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma} \qquad \frac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}\ \textbf{! C}$$

$$\textbf{! D}\ \frac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma} \qquad \frac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma}\ \textbf{! S}$$

The **!W** rule is the weakening rule, here restricted to formulas on the left of the turnstile that are prefixed with !. The **!C** rule is the contraction rule, similarly restricted. The **!D** rule connects the modal formulas to the rest of the logic. That is, one may consider ! to be a "wrapper", that allows arbitrary duplication and discarding, but which must be explicitly removed before the contents can be used. Finally, the **!S** rule allows the generation of ! formulas on the right hand side of a sequent. Intuitively, if one can prove A using resources which are all !'d, then one can produce any number of As (that is, $!A$). One may view ! as a particular kind of necessitation operator, strengthening the force of an assertion, since **!D** and **!S** are the standard proof rules for necessitation. There is also a set of rules for the ? operator, essentially the same as the rules for ! mirrored onto the opposite side of the $\vdash$.

1.3.8 Linear Logic Quantifier Rules

Predicate Linear logic has the standard quantifier rules:

$$\exists\ \textbf{Right}\ \frac{\Gamma \vdash A\{t/x\}, \Sigma}{\Gamma \vdash \exists x.A, \Sigma} \qquad \frac{\Gamma \vdash A\{y/x\}, \Sigma}{\Gamma \vdash \forall x.A, \Sigma}\ \forall\ \textbf{Right}$$

In the $\exists$**R** rule, substituting an arbitrary term t for the free occurrences of x in A is denoted $A\{t/x\}$, where the bound variables in A are renamed to avoid any clashes. The $\forall$ **Right** rule is only applicable if the variable y is not free in Γ, Σ, and any nonlogical theory axioms. As for the other connectives of linear logic, there are right and left versions of the $\exists$ and $\forall$ rules. With these rules one may consider first order and higher order systems of linear logic. However, we will focus almost exclusively on propositional linear logic, where quantifiers never appear.

1.3.6 The Units of Linear Logic

In linear logic, the four connectives have separate units (identity elements). The unit of $\otimes$ is called 1 (one), the unit of $\&$ is $\top$ (top), the unit of $\parr$ is $\bot$ (bottom), and the unit of $\oplus$ is 0 (zero). The sequent rules for these constants are given below. Note that there is no rule for 0 on the right hand side of the turnstile, nor for $\top$ on the left.

$$\frac{}{\Sigma \vdash \top, \Delta} \qquad \frac{}{\vdash 1} \qquad \frac{}{\bot \vdash}$$

$$\frac{}{\Sigma, 0 \vdash \Delta} \qquad \frac{\Sigma \vdash \Delta}{\Sigma, 1 \vdash \Delta} \qquad \frac{\Sigma \vdash \Delta}{\Sigma \vdash \bot, \Delta}$$

The $\top$ rule may be thought of as stating that $\top$ is "really true", no matter what the context. On the other hand, 1 is true, but only in the complete absence of any other formulas. $\bot$ is a kind of false that may be disregarded, and 0 is a kind of false that must be accounted for.

1.3.7 Resource Control Aspects of Linear Logic

An important property of linear logic with $\otimes$ $\parr$, $\&$, $\oplus$, 1, $\bot$, $\top$, 0 is that every connective of the conclusion of a proof is analyzed at most once in any branch of the proof. This can be seen as an advantage, since it affords a great deal of control over the process of proof search, and yields an immediate polynomial space decision procedure. This can also be seen as a disadvantage, because one cannot symmetrically encode intuitionistic logic since intuitionistic logic allows arbitrary uses of certain formulas. Therefore a pair of modal operators ! and ? are included in linear logic to retrieve the kind of expressive power introduced by arbitrary reuse.

The formula $!A$ represents unlimited use of the formula A. Thus in the sequent $!A, \Sigma \vdash \Delta$, the connectives of A may be analyzed any number of times, including zero. In logical terms, weakening and contraction apply to ! formulas on the left of the $\vdash$.

1.3.4 Conjunction and Disjunction

In sequent calculus with the structural rules of weakening, contraction, and exchange on both sides of $\vdash$, the following two rules for conjunction are equivalent. That is, the rule for $\wedge_1$ may be derived from the rule for $\wedge_2$, and vice versa.

$$\frac{\Sigma \vdash A, \Delta \qquad \Sigma \vdash B, \Delta}{\Sigma \vdash (A \wedge_1 B), \Delta} \qquad \frac{\Sigma \vdash A, \Delta \qquad \Theta \vdash B, \Gamma}{\Sigma, \Theta \vdash (A \wedge_2 B), \Delta, \Gamma}$$

However, if one removes the structural rules of contraction and weakening, as in linear logic, the rules for $\wedge_1$ and $\wedge_2$ are *not* equivalent. The form of rules where the context (Σ and Δ) is used in both hypotheses ($\wedge_1$) is hereafter referred to as an "additive" rule, and the additive conjunction will be written $\&$. The second kind of rule, where the context is divided among the hypotheses, is referred to as "multiplicative", and will be written $\otimes$. The precise proof rules for these connectives are repeated in Figure 1.1 and Appendix A.

For similar reasons, linear logic distinguishes between two kinds of disjunctions, an additive one $\oplus$ (plus) and a multiplicative one (par):

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

1.3.5 The Cut Rule

The cut rule provides a kind of modes ponens for a logic:

$$\frac{\Gamma_1 \vdash A, \Sigma_1 \qquad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2} \ \textbf{Cut}$$

This rule may be read as "if A or Σ_1 can be derived from Γ_1, and Γ_2 and A together derive Σ_2, then Γ_1 and Γ_2 together derive Σ_1 or Σ_2". Of course, we intend multiplicative conjunction and disjunction in the above description. This rule perhaps is easiest to understand if Σ_1 is empty. In that case, this rule essentially states that Γ_1 can be plugged in for A in the derivation of $\Gamma_2, A \vdash \Sigma_2$.

hand side of a sequent. Linear logic completely forbids the application of weakening on either side of $\vdash$.

Classical and intuitionistic logics also allow the following rule of inference called *contraction*:

$$\frac{\Sigma, A, A \vdash \Delta}{\Sigma, A \vdash \Delta}$$

Intuitively, if one can prove a formula from two assumptions of a formula A, then one assumption of formula A suffices. This is a mild strengthening of the conclusion. Algebraically, the rule asserts that the comma (conjunction) forming the left of a sequent is idempotent.

Classical logic allows contraction to be applied on both sides of a sequent (i.e., both sides of $\vdash$). Intuitionistic logic restricts contraction to apply only on the left hand side of the $\vdash$. Linear logic completely forbids the application of contraction on either side of $\vdash$.

Classical, intuitionistic, and linear logic all allow the following structural rule called *exchange*:

$$\frac{\Sigma, A, B, \Gamma \vdash \Delta}{\Sigma, B, A, \Gamma \vdash \Delta}$$

Intuitively, the rule asserts that the order of formulas is unimportant, or in other words, the comma (conjunction) is commutative. In Chapter 4 we consider the non-commutative variant of linear logic, with the structural rule of exchange omitted from the logic.

As a convenience, one may simplify presentations of the sequent calculus for classical or intuitionistic logic by treating sequences of formulas Σ as *sets*. A derivation in this system would no longer require that the conclusion of each rule exactly match the hypothesis of the next rule, but instead one simply requires a match up to applications of the structural rules. One may view this as simply ignoring the applications of structural rules, leaving their application implicit. The analogous presentation of linear logic treats sequences of formulas as multisets.

The *principal formula* of an occurrence of an inference rule is the formula that appears in the conclusion but does not appear in any hypothesis. In the case above, the formula $B \wedge A$ is the principal formula of the $\wedge$ rule. We will also say that the formula $B \wedge A$ is *analyzed*, or *broken* in the last rule applied in the above proof. This terminology stems from usage in proof search, where one begins with a conclusion, or root sequent, and searches for proofs. In incrementally building that deduction, the proof search procedure is said to analyze a formula.

The occurrence of a rule in a proof is said to be an *inference*. When we speak of a formula in a sequent, we are really referring to an occurrence of the formula.

1.3.3 Structural Rules

The discussion in Section 1.3.2 applies to the sequent calculi for classical or intuitionistic logic as Gentzen originally presented them [29]. For the remainder of the thesis we will focus on linear logic.

The rules of inference for linear logic differ from classical logic and intuitionistic logic in many ways, but the most dramatic difference can be explained by the rules of classical logic that are "missing" from linear logic: weakening and contraction.

Classical and intuitionistic logics allow the following rule of inference called *weakening* or *thinning*:

$$\frac{\Sigma \vdash \Delta}{\Sigma, A \vdash \Delta}$$

Intuitively, if Σ is the single formula X, and Δ is the single formula Y, and one has a proof of X implies Y, then one can assert that (X and A) implies Y. The effect of this rule is to weaken the conclusion, but if the original sequent was true, then the new one surely is also.

Classical logic allows weakening to be applied on both sides of a sequent (i.e., both sides of $\vdash$). This is sound because the right hand side of a sequent is interpreted as the disjunction of formulas, and thus adding formulas to the conclusion also weakens the result. Intuitionistic logic restricts weakening to apply only on the left hand side of the $\vdash$, since intuitionistic logic forbids multiple formulas from appearing on the right

Some sequent calculus rules have one hypothesis, and some have no hypotheses. For example, the rule below for identity has no hypotheses:

$$\frac{}{A \vdash A}$$

Given a set of proof rules, a *deduction* is a structure where the conclusion of one proof rule exactly matches the hypothesis of the next rule. Because each rule has exactly one conclusion, all deductions are trees, with the root (conclusion) at the bottom, and the leaves (hypotheses) at the top. Each branch of a deduction is a sequence of applications of proof rules, some, such as $\wedge$, represent branching points in the deduction tree, and some, such as identity, terminate a branch. A *proof* in the sequent calculus is a deduction with no assumptions, *i.e.*, every leaf is either identity or a logical axiom. In other words, each branch terminates with an application of a proof rule with no hypotheses.

A useful extension to the sequent calculus is the introduction of theories. A *theory* is a set of sequents, each element of which is called a (non-logical) axiom. A proof in a theory is a deduction where every branch is terminated by a proof rule with no hypotheses, a logical axiom, or by a sequent that is in the theory.

Sequent proofs of formulas will be displayed with the name of the inference rule near the right hand side of the line delineating the inference. This is an aid to the reader of complicated sequent proofs, although this is not a standard proof notation. For example, here is a sequent calculus proof with two applications of the identity rule, and one application of the conjunction rule:

$$\dfrac{\dfrac{}{A \vdash A}\,I \qquad \dfrac{}{A \vdash A}\,I}{A \vdash A \wedge A}\,\wedge$$

The next is a proof in the theory with the single axiom $A \vdash B$. Note that we have written a line over the use of the axiom as if there were an inference rule for each axiom in a theory, and labeled it **T**.

$$\dfrac{\dfrac{}{A \vdash B}\,T \qquad \dfrac{}{A \vdash A}\,I}{A \vdash B \wedge A}\,\wedge$$

The following notational conventions will be used:

p_i	Positive propositional literal
$p_i^{\perp}$	Negative propositional literal
A, B, C	Arbitrary formulas
$\Sigma, \Gamma, \Delta, \Theta$	Arbitrary sequences of formulas
$\otimes$	Tensor, the multiplicative conjunction
1	One, the unit of tensor
$\parr$	Par, the multiplicative disjunction
$\perp$	Bottom, the unit of par
$\&$	With, the additive conjunction
$\top$	Top, the unit of with
$\oplus$	Plus, the additive disjunction
0	Zero, the unit of plus
$\multimap$	Linear implication, definable from par and negation

1.3.2 Sequent Calculus

The sequent calculus, devised by Gentzen [29], will be used throughout this thesis.

A *sequent* is composed of two sequences of formulas separated by the turnstile symbol, $\vdash$. One may read the sequent $\Delta \vdash \Gamma$ as asserting that the conjunction of the formulas in Δ imply the disjunction of the formulas in Γ.

A *sequent calculus proof rule* consists of a set of hypothesis sequents, displayed above a horizontal line, and a single conclusion sequent, displayed below the line, as below:

$$\frac{\text{Hypothesis1} \qquad \cdots \qquad \text{HypothesisK}}{\text{Conclusion}}$$

For example, this is the standard rule for conjunction:

$$\frac{\Sigma \vdash A, \Delta \qquad \Sigma \vdash B, \Delta}{\Sigma \vdash A \wedge B, \Delta}$$

The two hypotheses are $\Sigma \vdash A, \Delta$ and $\Sigma \vdash B, \Delta$, and the single conclusion is $\Sigma \vdash A \wedge B, \Delta$. As is standard, rules are implicitly universally quantified schemas.

procedure terminates) immediately yields a strong normalization property for simply-typed lambda terms.

This thesis analyzes an analogous correspondence between a functional language and linear logic which has been studied by Girard, Lafont, Abramsky, and others [53, 1, 91, 93, 95, 21, 16]. Interesting related work along a similar vein (the geometry of interaction) includes [32, 31, 33, 54, 3, 38, 22]. The main novel points developed in this thesis are the proofs of the subject-reduction and most general type theorems for a linear functional language. Also, from an implementation standpoint, a two-space memory model is developed which allows a compiler to generate more efficient code by taking into account the linear type of certain terms. This memory model has been used in an implementation of a linear declarative language based on the Three Instruction Machine (TIM) [27].

1.3 Formal Overview of Linear Logic

Below is a brief presentation of the notation used in this thesis (Girard's), and some of the more foundational concepts of logic and proof theory necessary for the remainder of this thesis. This thesis is self-contained, but interested readers are referred to [29, 51, 30, 36] for some of the proofs referred to in the text. Introductions to linear logic in general are given in [84, 57, 34, 87].

1.3.1 Notation and Terminology

Girard's notation for the logical connectives of linear logic ($\otimes$ $\parr$, $\&$, $\oplus$, 1, $\bot$, $\top$, 0) will be used throughout the thesis [1].

[1] Recent discussions of some alternative notations have occurred on an electronic mailing list maintained by the author. The mailing list is called "linear@cs.stanford.edu". To subscribe to this list send electronic mail to "linear-request@cs.stanford.edu".

The "Girard Correspondence" is closely related to the logic programming approach. It establishes the connection between formulas and states of computation, and between proofs and computations. Girard's correspondence differs from the usual logic programming approach in that it identifies *proofs* with computations, whereas logic programming identifies *proof search* with computation. In this thesis we refine this correspondence through separating the program from the state of the machine by working in nonlogical theories that are derived from programs. The version of the Girard correspondence developed in this thesis maintains a clear distinction between program and state: a program is a non-logical theory, a machine state is a sequent, and a proof is a (successful) computation.

This Girard correspondence is used to prove several new complexity results for the decision problems for several fragments of linear logic. Natural fragments of linear logic vary widely in complexity. One fragment has an NP-complete decision problem, one has a PSPACE-complete decision problem, and full propositional logic is undecidable. Using the Girard correspondence, many models of computation can be captured logically. For example, in noncommutative linear logic one can encode several variants of Turing machines (multi-tape, multi-head, multiple points of control). A second example, due to previous researchers, is that Petri net computations can be captured quite naturally in the tensor fragment of linear logic with tensor theories [8, 67]. Also, many logical formalisms used in knowledge representation and linguistics are characterized by fragments of linear logic.

1.2.2 Linearizing Curry-Howard

Another very useful correspondence, known as the "propositions-as-types" or Curry-Howard correspondence, connects natural deduction systems for intuitionistic logic and typed lambda calculus. This connection is quite deep, as cut elimination in intuitionistic logic corresponds to reduction in the lambda calculus. Thus one may view typed lambda terms as notations for proofs in intuitionistic logic. One may also view the soundness of cut-elimination for intuitionistic logic as providing a proof that reduction preserves types in the lambda calculus. Further, the strong normalization property of natural deduction systems for intuitionistic logic (*i.e.* the cut-elimination

1.2 Overview of Proof-Theoretic Results

This investigation into linear logic first builds a set of proof theoretic tools for linear logic, including cut-elimination, non-logical theories, and permutabilities of inferences. None of these tools are particularly novel to this thesis, but the presentation of them in the sequent calculus and collection of them together aids further developments a great deal. In particular, Girard proved cut-elimination using proof nets in [30], while cut-elimination is proved here using the sequent calculus in order to facilitate later proofs which make use of the syntax and exact form of this proof. Non-logical theories for linear logic have been investigated in [39, 67] and are extended here to include a wider range of axioms. Andreoli, in the context of proof search for linear logic as a programming language [6], and Bellin, in an unpublished manuscript [14], have also studied some permutabilities and impermutabilities in the sequent calculus for linear logic, and the proofs of soundness and completeness of proof nets for fragments of linear logic also (implicitly) give permutability theorems in the sequent calculus [30]. A complete list of permutabilities is given in this thesis.

On this expanded proof theoretic foundation, this thesis builds in two directions corresponding to two modes of interpreting logics computationally. One branch of development investigates the "Girard correspondence" which connects sequents with states, and proofs with computations. This correspondence is closely related to the logic programming approach. The second branch of development explores a Curry-Howard-style correspondence between proofs and programs, and proof-normalization with execution. These two modes are summarized briefly in the next subsections.

1.2.1 Girard Correspondence

The "logic programming" approach to the computational interpretation of a logic, as exemplified by the programming language Prolog, is a correspondence between conclusions of formal proofs and programs, and also between the process of searching for a proof and the execution of the logic program. Andreoli and Pareschi, among others, have investigated this paradigm as it is applied to linear logic, and have developed a very useful programming style and implementation [7].

revisits the Curry-Howard correspondence between proofs and programs, originally observed for intuitionistic logic. Linear logic adds a greater degree of control over the structure of programs to the Curry-Howard correspondence. The main conclusion drawn from the results of this thesis is that linear logic is a *computational* logic behind logics. That is, linear logic is not about "Truth"; it is about computation.

1.1 Brief History of Linear Logic

Linear logic, although only recently formulated [30], is closely related to much older logics. Lambek developed a non-commutative logic intended for the analysis of natural language sentence structure a few decades earlier than Girard formulated linear logic [55, 56]. Relevance logic and Direct logic, which are also much older than linear logic, had already been well studied when linear logic appeared on the scene [25, 88, 50]. In an oversimplified view, linear logic sits below relevance and direct logic, and above the Lambek calculus, according to the inclusion or exclusion of certain "structural rules" called weakening, contraction, and exchange, discussed later in this thesis [15, 78].

Linear logic arose from the semantic study of intuitionistic implication. In fact, Girard gave two separate semantics in his article introducing linear logic [30]. The exploration of alternate semantic bases for linear logic continues today [81, 66, 85, 24, 10, 18]. Linear logic proof theory was also begun by Girard with the introduction of a sequent calculus for linear logic, and an alternate notation for linear logic proofs called proof nets [30]. Study of proof net related issues is also still active [32, 31, 33, 54, 3, 38, 22].

This thesis addresses several issues in the proof theory of linear logic, but is not concerned with semantics, nor does it utilize proof nets.

Chapter 1

Introduction

Linear logic was introduced by Girard in 1987 [30] as a "logic behind logics", and as a "resource conscious logic". In the framework of linear logic, this thesis addresses both complexity and programming language issues. The main theoretical concern is to strengthen the conceptual underpinnings necessary to apply proof theory to reason about computations. We demonstrate the undecidability of propositional linear logic, prove that noncommutative non-additive propositional linear logic is also undecidable, and give tight complexity results for other natural fragments of linear logic. Then, using these results, we explore an application of proof theory to computation, describing a functional language ML^{--} and its (compiled) implementation. The linear analysis of this programming language yields compile-time type information about resource manipulation, which may be useful in the control of some aspects of program execution such as storage allocation, garbage collection, and array update in place.

The principal contribution of this thesis is the investigation of two computational interpretations of linear logic. The first set of results demonstrates the power of a correspondence advocated by Girard between proofs and computations. This correspondence links formulas to states of machines, and connects inference steps in a proof to transitions in a computation. This "Girard" correspondence allows the use of proof theory to reason about computations and their properties such as correctness and termination. Moreover, it is the natural way to study the computational complexity of decision problems in various logics. The second set of results in this thesis

6.18 Case **Left** $\supset$ **2**. 160
6.19 Definition of Ξ 165

7.1 Grammar of the Linear Lambda Calculus 176
7.2 Prolog Implementation of the $\otimes$, $\multimap$ fragment of NAT2 203
7.3 Prolog Implementation of the ! fragment of NAT2 204

List of Figures

1.1 Sequent Calculus Rules for Linear Logic 14

2.1 Sequent Calculus Rules for Intuitionistic Linear Logic (ILL) 27

3.1 Zero-test proof 77
3.2 Proof corresponding to computation 77

6.1 Proof of $\Sigma \vdash r$ in IIL where Σ is $l \supset r, (p \supset q) \supset l, (q \supset r) \supset q$ 143
6.2 Modified Proof 144
6.3 "Linearized" Proof in IIL* where A is $l \supset r$, B is $r \supset q$, C is $(q \supset r) \supset q$, D is $q \supset l$. 144
6.4 Toward IMALL translation of example. 145
6.5 Rules for IIL 146
6.6 Rules for IIL* 150
6.7 Permuting backward inferences 151
6.8 Definition of *weight* 152
6.9 Example calculation of weight 153
6.10 Rules for IMALL 154
6.11 Definition of translation 155
6.12 IIL* and IMALL proofs of example. 156
6.13 Case 1 of Lemma 6.4.69 157
6.14 Case 2 of Lemma 6.4.69 157
6.15 Proof of Lemma 6.4.70 157
6.16 Case **Right** $\supset$. 158
6.17 Case **Left** $\supset$ **1**. 159

List of Tables

6.6.1 Depth Reduction in IIL 165
6.7 Discussion 167
6.8 Summary of Chapter 168

7 Linear ML^{--} **169**
7.1 Introduction 169
7.2 Why explicit storage operations? 172
7.3 The Linear Calculus 175
7.3.1 Linear terms and reduction 176
7.3.2 Typing preliminaries 177
7.3.3 The typing rules 178
7.3.4 NAT Subst Elimination 181
7.3.5 Equivalence of SEQ and NAT 198
7.4 Most General Linear Type 199
7.4.1 Most General Types in NAT and SEQ 199
7.4.2 Most General Types in NAT2 201
7.5 Type Soundness 202
7.6 Implementation of LC 206
7.7 Summary of Chapter 209

8 Conclusion **210**

A Linear Logic Sequent Calculus Rules **213**

B Prop. Intuitionistic Linear Logic **215**

C Linear Calculus **217**

D SEQ Proof Rules **219**

E NAT Proof Rules **222**

Bibliography **225**

4.5.2 Rotate Rule versus Embedding 94
4.5.3 CL without ?**E** 95
4.5.4 Alternate $\otimes$ 96
4.5.5 Mix and Match 97
4.6 Degenerate Noncommutative Linear Logics 97
4.6.1 Intermingling $\otimes$ 97
4.6.2 Intermingling **Cut** 98
4.7 Summary of Chapter 99

5 Decidable Fragments of Linear Logic **100**
5.1 MALL is PSPACE-complete 100
5.1.1 Membership in PSPACE 101
5.1.2 Informal Outline of PSPACE-hardness of MALL 102
5.1.3 Encoding Boolean Evaluation 104
5.1.4 Encoding Boolean Quantification 108
5.1.5 Formal Definition of the Encoding 110
5.1.6 Proof of PSPACE-hardness of MALL 113
5.1.7 IMALL is PSPACE-Complete 123
5.2 Multiplicative Linear Logic is NP-Complete 124
5.2.1 IMLL and CMLL Are In NP 124
5.2.2 IMLL is NP-Hard 125
5.2.3 Constant-Only Case 130
5.2.4 Constant-Only Multiplicative Linear Logic is NP-Complete . . 137
5.3 Summary of Chapter 137

6 Linearizing Intuitionistic Implication **139**
6.1 Overview 141
6.2 Properties of IIL 146
6.3 IIL and IIL* 149
6.4 IIL* to IMALL 154
6.5 Completeness of Translation 158
6.6 Efficiency of Transformation 164

1.4.4 Linear ML^{--} 20

2 Linear Proof Theory **24**

2.1 Basic Properties of Linear Logic 24
2.2 Intuitionistic Linear Logic 25
2.3 Cut Elimination 29
2.3.1 Cut of non-principal formulas 32
2.3.2 Cut of principal formulas 37
2.4 Subformula Property 47
2.5 Polarity 48
2.6 Permutabilities of Linear Logic 49
2.7 Linear Logic Augmented With Theories 52
2.7.1 Encoding MALL Theories 52
2.7.2 Cut Standardization 55
2.8 Summary of Chapter 59

3 Propositional Linear Logic is Undecidable **60**

3.1 And-Branching Two Counter Machines Without Zero-Test 61
3.1.1 Two Counter Machines 63
3.2 From Machines to Logic 67
3.3 Example Computation 74
3.4 Summary of Chapter 78

4 Noncommutative Propositional Linear Logic **79**

4.1 FICL Rules 81
4.2 FICL is Undecidable 83
4.3 FICL Theories 84
4.3.1 Theories into FICL 85
4.3.2 Semi-Thue Systems 88
4.4 From Semi-Thue Systems to Noncommutative Linear Logic 89
4.5 Other Noncommutative Logics 92
4.5.1 One-sided Noncommutative Linear Logic: CL 92

Contents

Abstract iv

Acknowledgements vi

1 Introduction 1

1.1 Brief History of Linear Logic 2

1.2 Overview of Proof-Theoretic Results 3

1.2.1 Girard Correspondence 3

1.2.2 Linearizing Curry-Howard 4

1.3 Formal Overview of Linear Logic 5

1.3.1 Notation and Terminology 5

1.3.2 Sequent Calculus 6

1.3.3 Structural Rules 8

1.3.4 Conjunction and Disjunction 10

1.3.5 The Cut Rule 10

1.3.6 The Units of Linear Logic 11

1.3.7 Resource Control Aspects of Linear Logic 11

1.3.8 Linear Logic Quantifier Rules 12

1.3.9 The Sequent Calculus of Linear Logic 13

1.4 Overview of Thesis Results 15

1.4.1 Basic Theorems 15

1.4.2 Complexity Results 15

1.4.3 Linearizing Intuitionistic Implication 19

Chapters 3 and 4 and section 5.1 are based on joint work with John Mitchell, Andre Scedrov, and Natarajan Shankar [60]. Section 5.2 is based on joint work with Tim Winkler [63]. Chapter 6 is based on joint work with Andre Scedrov and Natarajan Shankar [62]. Chapter 7 is based on joint work with John Mitchell [58].

This work was supported by an AT&T Bell Laboratories doctoral scholarship and by SRI International.

I would like to give thanks to Andre Scedrov, Natarajan Shankar, Tim Winkler, and especially my advisor John Mitchell for our fruitful collaborations. I would also like to thank Vaughan Pratt, Grigori Mints, Steven Phillips, Jose Meseguer, Narciso Marti-Oliet, Sam Owre, and Vijay Saraswat for stimulating discussions on related subjects. Earlier guidance given by Hassan Aït-Kaci, Rishiyur Nikhil, Woody Bledsoe, and Bob Boyer was instrumental in the direction of my research. Finally, I would like to thank Raymonde Guindon and the rest of my growing family for their abundant support.

Acknowledgements

COMPUTATIONAL ASPECTS

OF LINEAR LOGIC

by

Patrick Denis Lincoln, Ph.D.

Stanford University, 1992

Linear logic was introduced by Girard in 1987 [30] both as a "logic behind logics", and as a "resource conscious logic". This thesis investigates computational aspects of linear logic. The main results of this work support the proposition that linear logic is a *computational* logic behind logics. This thesis augments the proof theoretic framework of linear logic by providing theorems such as permutability, impermutability, and cut-normalization with non-logical theories. On this expanded proof theoretic base, many complexity results are proved using a correspondence between proofs and computations. Among these results are the undecidability of propositional linear logic, the PSPACE-completeness of MALL, and the NP-completeness of the constant-only multiplicative fragment of linear logic. Another application of proof theory to computation is explored for a functional language ML^{--} and its (compiled) implementation. The proposed linear type system for ML^{--} yields compile-time type information about resource manipulation that may be useful in the control of some aspects of program execution such as storage allocation, garbage collection, and array update in place. Most general type and subject reduction theorems are proved, and a compiled implementation based on the Three Instruction Machine is described. Together, these results point out that linear logic is not about "Truth"; it is about computation.

Abstract

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John Mitchell
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Grigori Mints
(Philosophy)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Vaughan Pratt

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Natarajan Shankar
(SRI International)

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

© Copyright 1992 by Patrick D. Lincoln
All Rights Reserved

COMPUTATIONAL ASPECTS OF LINEAR LOGIC

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Patrick D. Lincoln
August 1992