Linear Logic

Patrick Lincoln
lincoln@cs.stanford.edu

Stanford University

1 Linear Logic

Linear logic was introduced by Girard in 1987 [11]. Since then many results have supported Girard's claims such as "Linear logic is a resource conscious logic". Increasingly, computer scientists have recognized linear logic as an expressive and powerful logic, with deep connections to concepts from computer science. The expressive power of linear logic is evidenced by some very natural encodings of computational models such as Petri nets, counter machines, Turing machines, and others.

This note presents an intuitive overview of linear logic, some recent theoretical results, and some interesting applications of linear logic to computer science. Other introductions to linear logic may be found in [12, 36].

2 Linear Logic vs Classical and Intuitionistic Logics

Linear logic differs from classical and intuitionistic logic in several fundamental ways. Classical logic may be viewed as if it deals with static propositions about the world where each proposition is either true or false. Because of the static nature of propositions in classical logic, one may "duplicate" propositions: P implies (P and P). Implicitly we learn that "one P is as good as two". Also, one may discard propositions: (P and Q) implies P. Here the proposition Q has been "thrown away". Both of these sentences are valid in classical logic for any P and Q.

In linear logic, these sentences are *not* valid. A linear logician might ask "Where did the second P come from?" and "Where did the Q go?". Of course these questions are nonsequiturs in the classical setting, since propositions are assumed to be static, unchanging facts about the world. On the other hand, the rules of linear logic imply that linear propositions stand for dynamic properties or finite resources.

For example, consider the propositions D, M, and C, conceived as resources:

$$\begin{aligned} D &\triangleq \text{"One Dollar"} \\ M &\triangleq \text{"A pack of Marlboros"} \\ C &\triangleq \text{"A pack of Camels"} \end{aligned}$$

Consider the following axiomatization of a vending machine:

$$D \text{ implies } M$$

$$D \text{ implies } C$$

Then in classical (or intuitionistic) logic, one is able to deduce

$$D \text{ implies } (M \text{ and } C)$$

Which may be read as "With one dollar, I may buy both a pack of Marlboros and a pack of Camels". Although this deduction is valid in classical logic, it is nonsense in the intended interpretation of propositions as resources: one cannot buy two packs of cigarettes with one dollar from the vending machine described. This paradox arises out of the confusion in classical (and intuitionistic) logic between two kinds of conjunction: one intuitively meaning "I have both" (which is written in linear logic as $\otimes$), and another meaning "I have a choice" (written & in linear logic).

Linear logic avoids such paradoxes by distinguishing two kinds of conjunction, two kinds of disjunction, and by introducing a modal storage operator that explicitly marks those propositions that can be arbitrarily reused.

Linear negation $A^{\perp}$ is involutive, that is, $(A^{\perp})^{\perp} = A$, but is yet constructive. This is one of the fascinating aspects of linear logic.

Linear logic "multiplicative" conjunction $A \otimes B$ stands for the proposition that one has both A and B at the same time. The linear logic "additive" conjunction $A\&B$ stands for "one's own choice" between A and B, but not both. Dually, there are two disjunctions. The multiplicative disjunction, written $A \wp B$ stands for the proposition "if not A, then B". Perhaps this disjunction can be more easily understood by considering linear implication $A \multimap B$, which is defined by $A^{\perp} \wp B$. The formula $A \multimap B$ can be thought of as "can B be derived using A *exactly once*?". The additive disjunction $A \oplus B$ stands for the possibility of either A or B, but you don't know which. That is, "someone else's choice."

For each of these connectives, there is a unit: 1 is the unit of $\otimes$, so $A \multimap (A \otimes 1)$ and $(A \otimes 1) \multimap A$. $\top$ is the unit of &, $\perp$ is the unit of $\wp$, and 0 is the unit of $\oplus$.

2.1 Exponentials

To complete the logic, there is a modal storage operator ! (of course) and its dual ? (why not). The formula $!A$ may be thought of as a printing press for A's, which can generate any number of A's. For example, the U.S. government can be thought to have $!Dollar$s, and doesn't need to balance its budget, while citizens do not have $!Dollar$s, and thus have to balance their budgets.

2.2 Example

To illustrate the use of linear connectives and modal operators, here is an example, inspired by Girard and Lafont [12]. Suppose for a fixed $5 price a restaurant will provide a hamburger, a Coke, as many french fries as you like, onion soup or salad (your choice), and pie or ice cream (some else's choice). One may encode this information in the linear logic formula beside the menu:

Fixed-Price Menu: \$5	$(D \otimes D \otimes D \otimes D \otimes D)$
Hamburger	$\multimap$
Coke	$[H \otimes C \otimes !F \otimes (O \,\&\, S) \otimes (P \oplus I)]$
All the french fries you can eat	
Onion Soup or Salad	
Pie or Ice Cream (depending on availability)	

3 Sequent Calculus Notation for Linear Logic

The entire set of Gentzen-style sequent rules for linear logic are given at the end of this note. As explained above, the rules define two conjunctions and two disjunctions, as well as modal and constant operators. One could add quantifiers to form first (or higher) order linear logic, but for this paper we will restrict attention to propositional linear logic.

The sequent calculus notation, due to Gentzen [10], uses roman letters for propositions, and greek letters for sequences of formulas. A *sequent* is composed of two sequences of formulas separated by a $\vdash$, or turnstile symbol. One may read the sequent $\Delta \vdash \Gamma$ as asserting that the multiplicative conjunction of the formulas in Δ together imply the multiplicative 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:

$$\frac{\text{Hypothesis1} \qquad \text{Hypothesis2}}{\text{Conclusion}}$$

4 Connections to Other Logics

The most interesting features of linear logic arise from the absence of the rules of contraction and weakening. In classical or intuitionistic logic, the following rules are allowed:

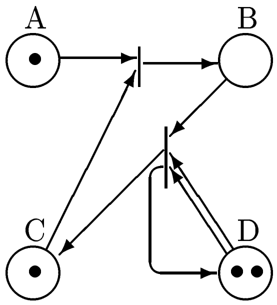
$$\textbf{Weakening Left} \quad \frac{\Delta \vdash \Sigma}{\Delta, A \vdash \Sigma} \qquad\qquad \frac{\Delta, A, A \vdash \Gamma}{\Delta, A \vdash \Gamma} \quad \textbf{Contraction Left}$$

Direct and Affine logic share with linear logic the elimination of the contraction rule [19]: *i.e.*propositions cannot be arbitrarily copied. However, both of these logics allow weakening. Relevance and Pertinent logic disallow weakening, but allow contraction: *i.e.*propositions cannot be arbitrarily discarded, but can be copied. Pertinent logic is decidable [32], but Relevance logic adds a distributivity axiom, which is absent from linear logic, which makes Relevance logic undecidable [37]. Linear logic disallows both weakening and contraction in general, although they are allowed for modal (! and ?) formulas.

Linear logic arose partly out of a study of intuitionistic implication. Girard found that the intuitionistic implication $A \Rightarrow B$ could be decomposed into two separate connectives: $!A \multimap B$. Girard showed that one could thus translate intuitionistic (and also classical) logic into linear logic directly, simply appending modals to certain subformulas and making the right choice as to which sort of conjunction and disjunction should be used. Here we see a first glimpse of the substance behind the slogan "Linear logic is a logic behind logics."

5 Connections to Computer Science

There has been much recent excitement about linear logic in the logic-based theoretical computer science community. Most of this excitement stems from the newfound ability to capture difficult "resource" problems logically. For example, linear logic provides a natural and simple encoding of Petri net reachability. In linear logic the formula $!((\mathbf{a} \otimes \mathbf{c}) \multimap \mathbf{b})$ may be used to encode a Petri net transition taking tokens from place **a** and **c** and adding a token to place **b**. Similarly, the formula $!((\mathbf{b} \otimes \mathbf{d} \otimes \mathbf{d}) \multimap (\mathbf{c} \otimes \mathbf{d}))$ may be seen as a transition taking one token from **b** and two tokens from **d**, and adding one token to **c**. These transitions are presented graphically below:



$$\begin{gathered} !((\mathbf{a} \otimes \mathbf{c}) \multimap \mathbf{b}), \\ !((\mathbf{b} \otimes \mathbf{d} \otimes \mathbf{d}) \multimap (\mathbf{c} \otimes \mathbf{d})), \\ \mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{d} \end{gathered}$$

Thus one can encode Petri net transitions as reusable linear implications. Tokens are represented as atomic propositions, and a reachability problem may be presented as a sequent:

$$!((\mathbf{a} \otimes \mathbf{c}) \multimap \mathbf{b}), !((\mathbf{b} \otimes \mathbf{d} \otimes \mathbf{d}) \multimap (\mathbf{c} \otimes \mathbf{d})), \mathbf{a}, \mathbf{c}, \mathbf{d}, \mathbf{d} \vdash \mathbf{c}, \mathbf{d}$$

This sequent is provable in linear logic if and only if there is a sequence of Petri net rule applications that transform the token set $\{a, c, d, d\}$ to $\{c, d\}$. This connection has been well-studied [5, 14, 30, 6, 9], and extended to cover other models of concurrency [22, 2, 35].

Linear logic has also been applied to several other areas of computer science. One key application of the resource-sensitive aspect of the logic was the development of a functional programming language implementation in which garbage collection was replaced by explicit duplication operations based on linear logic [21]. More recent work has attempted to find a linear logical basis for many optimizations in (lazy) functional programming language implementations by concentrating on linear logic as a type system [1, 15, 39, 40, 25, 8, 29, 41].

Other applications include analyzing the control structure of logic programs [7], generalized logic programming [4, 16], and natural language processing [23]. A natural characterization of polynomial time computations can be given in a bounded version of linear logic [13] obtained by limiting reuse to specified bounds, *i.e.*, by bounding the number of references to each datum in memory.

We now turn our attention to some questions of a more theoretical nature.

6 Complexity Results for Linear Logic

Although propositional linear logic was known to be very expressive, for some time it was thought to be decidable. However, propositional linear logic was recently shown to be undecidable [26]. Several other complexity results are given in [26], including the PSPACE-completeness of propositional linear logic without ! or ?.

A key open complexity problem is the decision problem for the $\otimes, \wp, !, ?$ fragment of linear logic. An equivalent fragment is $\otimes, \multimap, !, ?$, which suffices for the encoding of Petri net reachability questions, as shown above, and thus is at least EXPSPACE-hard [31]. It is currently unknown if this multiplicative-exponential fragment of linear logic is decidable or not.

6.1 Undecidability of Propositional Linear Logic

The full proof of undecidability is presented in [27], and is sketched below.

The proof of the undecidability of full linear logic proceeds by reduction of a form of alternating counter machine to propositional linear logic. An and-branching two-counter machine (ACM) is a nondeterministic machine with a finite set of states. A configuration is a triple $\langle Q_i, A, B\rangle$, where Q_i is a state, and A and B are natural numbers, the values of two counters. An ACM has a finite set of instructions of five kinds: **Increment-A**, **Increment-B**, **Decrement-A**, **Decrement-B**, and **Fork**. The **Increment** and **Decrement** instructions operate as they do in standard counter machines [33]. The **Fork** instruction causes a machine to split into two independent machines: from state $\langle Q_i, A, B\rangle$ a machine taking the transition $Q_i\mathbf{Fork}Q_j, Q_k$ results in two machines, $\langle Q_j, A, B\rangle$ and $\langle Q_k, A, B\rangle$. Thus an instantaneous description is **set of** machine configurations, which is accepting only if all machine configurations are in the final state, and all counters are zero. ACM's have an undecidable halting problem. One may notice that ACM's are essentially alternating Petri nets. It is convenient to use ACM's as opposed to standard counter machines to show undecidability, since zero-test has no natural counterpart in linear logic, but there is a natural counterpart of **Fork**: the additive conjunction &. The remaining ACM instructions may be encoded using techniques very similar to the Petri net rechability encoding described earlier.

Linear logic has a great control over resources, through the elimination of weakening and contraction, and the explicit addition of a resuable (modal) operator. Although the logic does not have quantifiers, the combination of these features yields a great deal of expressive power.

6.2 Complexity of Fragments of Linear Logic

6.2.1 PSPACE-completeness of Linear Logic Without $!, ?$

The multiplicative-additive fragment of linear logic (MALL) excludes the reusable modals $!, ?$. Thus, every formula is "used" at most once in any branch of any cut-free MALL proof. Also, in every non-cut MALL rule, each hypothesis sequent has a smaller number of symbols than the conclusion sequent. This provides an immediate linear bound on the depth of cut-free MALL proofs. Since MALL enjoys a cut-elimination property, there is a nondeterministic PSPACE algorithm to decide MALL sequents.

To show that MALL is PSPACE-Hard, one can encode classical quantified boolean formulas (QBF). For simplicity one may assume that a QBF is presented in prenex form. The quantifier-free formula may be encoded using truth tables, but the quantifiers present some difficulty. One may encode quantifiers using the additives: $\forall x$ as $(x \& x^{\perp})$, and $\exists x$ as $(x \oplus x^{\perp})$. This encoding has incorrect behavior in that it does not respect quantifier order, but using multiplicative connectives one can enforce an ordering upon the encoding of quantifiers to achieve soundness and completeness. The full proof of PSPACE-completeness is presented in [27].

6.2.2 NP-completeness of Multiplicative Linear Logic

The multiplicative fragment of linear logic contains only the connectives $\otimes$ and $\multimap$ (or equivalently $\otimes$ and $\wp$), a set of propositions, and the constants 1 and $\perp$. The decision problem for this fragment is in NP, since an entire cut-free multiplicative proof may be guessed and checked in polynomial time. The decision problem is NP-hard by reduction from 3-Partition, a problem which requires a perfect partitioning of groups of objects in much the same way that linear logic requires a complete accounting of propositions [17, 18]. Somewhat surprisingly, there is an alternate encoding of 3-Partition in multiplicative linear logic that does not use any propositions, that is, using only the constants 1 and $\perp$ and the connectives $\otimes$ and $\multimap$. Thus this multiplicative constant-only fragment of linear logic is also NP-complete [28]. The full proof of NP-completeness is presented in [17].

6.3 Summary of Linear Logic Complexity

The following table summarizes some of the results thus far achieved in the study of the complexity of the decision problems for fragments of linear logic. The first two fragments listed are those discussed in some detail above, the full propositional logic, and the propositional fragment without modals $!, ?$ (MALL). The decidability of the next problem is in some question. An encoding of Petri net reachability problems in this fragment has been studied in [5], but although EXPSPACE-hard [31], Petri net reachability is known to be decidable [20]. It is not known how much more expressive this fragment of linear logic might be. The fourth fragment containing only the multiplicative connectives is NP-Complete [17, 18].

Connectives In Fragment	Linear Logic Complexity
$\otimes \wp$ $\& \oplus$ $!?$	Undecidable [26]
$\otimes \wp$ $\& \oplus$	PSPACE-Complete [26]
$\otimes \wp$ $!?$	unknown
$\otimes \wp$	NP-Complete [17]

In summary, linear logic is an expressive logic with an intrinsic accounting of resources. Although a non-classical logic, linear logic has the pleasant features of cut-elimination and involutive negation. In practical use, much mileage can be gained from the resource-sensitivity of linear logic to encode difficult problems in even the propositional fragment of linear logic. Current work is progressing to exploit the unique features of linear logic for use as a type system to study computational complexity [13] and compiler optimization techniques [40, 8, 29, 41, 34, 25], as well as uses in logic programming [16, 3, 4], natural language processing [24, 38], and concurrency [5, 30, 35]. These recent contributions are developing linear logic from a theoretical curiosity into a tool that already has practical use within mainstream computer science.

References

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

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

[4] 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.

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

[6] 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.

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

[8] J. Chirimar, C. Gunter, and J. Riecke. Linear ML. In *Lisp and Functional Programming*, 1992. To Appear.

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

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

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

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

[13] 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. Also To Appear in TCS.

[14] 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.

[15] 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.

[16] 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.*

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

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

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

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

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

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

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

[24] J. Lambek. From categorial grammar to bilinear logic. Draft., 1991.

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

[26] 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.

[27] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. Technical Report SRI-CSL-90-08, CSL, SRI International, 1990. To appear in Annals of Pure and Applied Logic.

[28] P. Lincoln and T. Winkler. Constant multiplicative linear logic is NP-complete. Draft, 1992.

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

[30] 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.

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

[32] R.K. Meyer. *Topics in Modal and Many-Valued Logic.* PhD thesis, University of Pittsburgh, 1966.

[33] 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.

[34] P. O'Hearn. Linear logic and interference control. In *4th Conf. on Category Thoery and CS*, 1991.

[35] V.R. Pratt. Event spaces and their linear logic. In *Proc. Second International Conference on Algebraic Methodology and Software Technology.* Springer-Verlag, 1992.

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

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

[38] J. VanBentham. *Language in Action.* North-Holland, 1991.

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

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

[41] D. Wakeling and C. Runciman. Linearity and laziness. In *Proc. 5-th ACM Conference on Functional Programming Languages and Computer Architecture*. Springer-Verlag, 1991.

Definition of Linear Negation

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

Sequent Calculus Rules for Linear Logic

Identity	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \qquad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2}$	**Cut**
Exch. Left	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Sigma}{\Gamma_1, B, A, \Gamma_2 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma_1, A, B, \Sigma_2}{\Gamma \vdash \Sigma_1, B, A, \Sigma_2}$	**Exch. Right**
$\otimes$ **Left**	$\dfrac{\Gamma, A, B \vdash \Sigma}{\Gamma, (A \otimes B) \vdash \Sigma}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \qquad \Gamma_2 \vdash B, \Sigma_2}{\Gamma_1, \Gamma_2 \vdash (A \otimes B), \Sigma_1, \Sigma_2}$	$\otimes$ **Right**
$\multimap$ **Left**	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \qquad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \multimap B) \vdash \Sigma_1, \Sigma_2}$	$\dfrac{\Gamma, A \vdash B, \Sigma}{\Gamma \vdash (A \multimap B), \Sigma}$	$\multimap$ **Right**
$\wp$ **Left**	$\dfrac{\Gamma_1, A \vdash \Sigma_1 \qquad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \wp B) \vdash \Sigma_1, \Sigma_2}$	$\dfrac{\Gamma \vdash A, B, \Sigma}{\Gamma \vdash (A \wp B), \Sigma}$	$\wp$ **Right**
$\&$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma} \qquad \dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \qquad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	$\&$ **Right**
$\oplus$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma \qquad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma} \qquad \dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right**
! 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**
$\perp$ **Left**	$\perp \vdash$	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash \perp, \Sigma}$	$\perp$ **Right**
1 **Left**	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, 1 \vdash \Sigma}$	$\vdash 1$	1 **Right**