Chapter 5

Linear Logic Programming

When we think of logic we generally first consider it as a discipline concerned with the study of propositions, truth, and inference. This may appear at first to be independent from any notion of computation. However, there are two immediate connections: *proofs as programs* and *proof search as computation*.

In constructive logic (and our approach has been constructive) we can view a proof as defining a construction (algorithm, program). For example, a proof of $A \supset B$ shows how to achieve goal B when given the resource A. Carrying out such a construction when we actually have obtained resource A the corresponds to computation. This notion of computation is most closely related to functional programming, but because of the state-aware nature of linear logic it also has some imperative flavor. We will discuss a computational interpretation of linear logic along these lines in Chapter **??**.

Another computational interpretation is closer to the way we have been using linear logic so far. Reconsider, for example, the encoding of Petri nets in linear logic. Each possible computation step of the Petri net is modeled by a corresponding inference step in linear logic. As a result, reachability in a Petri net corresponds to provability in its linear encoding. More importantly, each possible computation of a Petri net corresponds to a proof, and carrying out a computation corresponds to the construction of a proof. In other words, proof search in linear logic corresponds to computation.

This leads to the question if we can exploit this correspondence in order to design a *programming language* based on linear logic where computation is indeed proof search. The result of computation then is a particular proof, or possibly a collection or enumeration of proofs depending on the characteristics of the language. A program in this setting is simply a collection of propositions that, through their form, will lead the proof search engine down a particular path, thereby achieving a particular computation. In order to make this both feasible from the point of view of an implementation and predictable to the programmer, we need to full linear logic. We would like to emphasize that even on this fragment (called LHHF for Linear Hereditary Harrop Formulas), not every specification is executable, nor is it intended to be. We hope the development and the examples in this chapter will clarify this point.

5.1 Logic Programming as Goal-Directed Search

Our first approach to logic programming is via the notion of *goal-directed search*. It turns out that this view diverges from our earlier examples because it does not incorporate any concurrency. However, some of our earlier encodings can be rewritten to fit into the language given below.

Assume we are trying to prove $\Gamma; \Delta \Longrightarrow A$ where Γ are the unrestricted hypotheses (which correspond to the program), Δ are the linear hypotheses (which correspond to current state) and A which corresponds to the goal. The idea of goal-directed search is that we always first break down the structure of the goal A until it is atomic (P). At that point we focus on one of the hypotheses in Γ or Δ and apply consecutive left rules until the focus formula matches P and we can solve the subgoals generated during this process. This phase of the computation corresponds to a procedure call, where the generated subgoals correspond to the procedure body which is then executed in turn.

In order to have a satisfactory logical interpretation of the program (in addition to the computational one above), we would like this search procedure to be sound and non-deterministically complete. Soundness simply means that we only find valid proof, which is easy to accomplish since we only restrict the applications of ordinary sequent rules. Completeness means that for every derivation for the original judgment there is a sequence of choices according to the strategy above which finds this derivation. As we have seen in the development of focusing (Section 4.2), the goal-directed strategy above is generally *not* complete. However, it is complete if we restrict ourselves to right asynchronous connectives, because focusing is complete and all the connectives are orthogonal to each other.

This fragment (with some irrelevant, minor deviations) is called the system of linear hereditary Harrop formulas (LHHF).

$$A \quad ::= \quad P \mid A_1 \multimap A_2 \mid A_1 \& A_2 \mid \top \mid A_1 \supset A_2 \mid \forall x.\, A$$

We obtain the foundation for its operational semantics simply from the focusing system, restricted to the above connectives. Since they are all right asynchronous, we only need two of the four judgments. We call this a system of *uniform proofs* [MNPS91]. For the case of linear, this system has first been proposed by Hodas and Miller [HM94, Hod94], although similar ideas for classical linear logic had been developed independently by Andreoli [AP91, And92].

Goal-Directed Search. This corresponds to the *inversion* phase of the focusing system. Because all right asynchronous propositions are left synchronous, we do not need to collect them into an ordered context Ω but add them directly to the left synchronous hypotheses in Δ. We write

$$\Gamma; \Delta \Longrightarrow A \Uparrow$$

From the focusing system we obtain the following rules.

$$\frac{\Gamma; \Delta, A \Longrightarrow B \Uparrow}{\Gamma; \Delta \Longrightarrow A \multimap B \Uparrow} \multimap\mathrm{R} \qquad \frac{\Gamma; \Delta \Longrightarrow A \Uparrow \qquad \Gamma; \Delta \Longrightarrow B \Uparrow}{\Gamma; \Delta \Longrightarrow A \& B \Uparrow} \&\mathrm{R}$$

$$\frac{}{\Gamma; \Delta \Longrightarrow \top \Uparrow} \top\mathrm{R} \qquad \frac{\Gamma, A; \Delta \Longrightarrow B \Uparrow}{\Gamma; \Delta \Longrightarrow A \supset B \Uparrow} \supset\mathrm{R}$$

$$\frac{\Gamma; \Delta \Longrightarrow [a/x]A \Uparrow}{\Gamma; \Delta \Longrightarrow \forall x.\ A \Uparrow} \forall\mathrm{R}^a$$

Procedure Call. This corresponds to the *focusing* phase of the focusing system. In this case we can carry over the judgment directly

$$\Gamma; \Delta; A \Downarrow \Longrightarrow P$$

where A is the focus formula and P is always atomic. This judgment is also called *immediate entailment* because A must decompose to directly yield P as can be seen from the rules below.

This phase is triggered by a decision, if the goal formula is atomic.

$$\frac{\Gamma; \Delta; A \Downarrow \Longrightarrow P}{\Gamma; \Delta, A \Longrightarrow P \Uparrow} \mathsf{decideL} \qquad \frac{\Gamma, A; \Delta; A \Downarrow \Longrightarrow C}{\Gamma, A; \Delta \Longrightarrow P \Uparrow} \mathsf{decideL!}$$

During this phase, we simply carry out the left rules on the focus formula. Since we can never obtain a left asynchronous proposition (there are none among the linear hereditary Harrop formulas), we can only succeed if we obtain an atomic proposition equal to P.

$$\frac{}{\Gamma; \cdot; P \Downarrow \Longrightarrow P} \mathsf{init}$$

$$\frac{\Gamma; \Delta_1; B \Downarrow \Longrightarrow P \qquad \Gamma; \Delta_2 \Longrightarrow A \Uparrow}{\Gamma; \Delta_1, \Delta_2; A \multimap B \Downarrow \Longrightarrow P} \multimap\mathrm{L}$$

$$\frac{\Gamma; \Delta; A \Downarrow \Longrightarrow P}{\Gamma; \Delta; A \& B \Downarrow \Longrightarrow P} \&\mathrm{L}_1 \qquad \frac{\Gamma; \Delta; B \Downarrow \Longrightarrow P}{\Gamma; \Delta; A \& B \Downarrow \Longrightarrow P} \&\mathrm{L}_2$$

no left rule for $\top$ $\qquad \dfrac{\Gamma; \Delta; B \Downarrow \Longrightarrow P \qquad \Gamma; \cdot \Longrightarrow A \Uparrow}{\Gamma; \Delta; A \supset B \Downarrow \Longrightarrow P} \supset\mathrm{L}$

$$\frac{\Gamma; \Delta; [t/x]A \Downarrow \Longrightarrow P}{\Gamma; \Delta; \forall x.\ A \Downarrow \Longrightarrow P} \forall\mathrm{L}$$

Some of the premises of these rules refer back to goal-directed search. The collection of these premises for a particular focusing step (that is, procedure call) corresponds to the procedure body. Operationally, they will be solved only after we know if the init rule applies at the end of the sequence of focusing steps.

It is easy to see that uniform proofs are sound and complete with respect to the sequent calculus via the soundness and completeness of focusing.[1]

5.2 An Operational Semantics

The system of uniform proofs from the previous section is only the basis of an actual operational semantics for LHHF. There are still a number of choices left and we have to specify how they are resolved in order to know precisely how a query

$$\Gamma; \Delta \Longrightarrow A \Uparrow$$

executes. We organize this discussion into the forms of the non-deterministic choices that remain. We are not formal here, even though a formal description can certainly be given.[2]

Existential Non-Determinism. This arises in the choice of the term t in the $\forall$L rule during the focusing phase. This is resolved by substituting instead a logic variable X, where it is explicitly remember which parameters a the variable X may depend on. For initial sequents

$$\frac{}{\Gamma; \cdot; P \Downarrow \Longrightarrow P}\,\mathsf{init}$$

we instead allow the hypothesis P and goal P' and unify them instead of testing them for equality. Since we can always find a most general unifier, this involves no unnecessary overcommitment and we do not have to backtrack in order to retain completeness.

Conjunctive Non-Determinism. If several subgoals arise during focusing, or because we have a goal $A_1 \& A_2$, we have to solve all subgoals but the order presents a form of conjunctive non-determinism. We resolve this by always solving the premises in the uniform proof rules from left to right. This has the desirable effect that we only attempt to solve a subgoal once we have unified the atomic goal P with the proposition P' at the end of the focus formula.

Disjunctive Non-Determinism. Disjunctive non-determinism arises in the choice of the focus formula once the goal is atomic, and in the choice of the left rule if the focus formula is an alternative conjunction. This is resolved via depth-first search and backtracking. For the decision how to focus, we use the following order:

[1] *[add more formal statement]*
[2] *[several citations]*

1. First the linear or unrestricted hypotheses that were introduced during proof search, where we try the most recently made assumption first (right-to-left, in our notation).

2. Then we try the unrestricted assumptions that were fixed at the beginning of the search (the program), trying the propositions from first to last (left-to-right in our notation).

For alternative conjunctions as the focus formula, we first try the left conjunct and then the right conjunct.

Resource Non-Determinism. Resource non-determinism arises in the $\multimap L$ rule, where we have to split assumptions between the premises. Conceptually, this can be resolved by introducing Boolean constraints [HP97] and solving them eagerly. In order to gain a better intuition what this means operationally, equivalent systems that avoid explicit creation of constraints have been developed [CHP00]. We will give some intuition in Section 5.3 where we introduce the *input/output model* for resource management.

Unfortunately, the way we treate disjunctive non-determinism via depth-first search and backtracking means that there may be proofs we never find because the interpreter following our operational semantics does not terminate. Many attempts have been made to alleviate this difficulty, but none are entirely satisfactory. Depth-first search seems to be critical to obtain a simple and understandable operational semantics for programs that allows algorithms to be implemented efficiently in a logic programming language.

Even though the interpreter is incomplete in this sense, the non-deterministic completeness of the uniform proof system is still very important. This is because we would like to be able to interpret failure as unprovability. Since the uniform proof system is non-deterministically complete, we know that if the interpreter fails finitely and reports no proof can be found because all choices have been exhausted, then there cannot be a proof of the original goal.

To summarize, the interpreter may exhibit three behaviors.

1. Succeed with a proof. By soundness, the goal is provable. If we backtrack further, we may get other proofs.

2. Fail. By non-deterministic completeness, the goal is unprovable and hence not true in general.

3. Run. In this case we have no information yet. We cannot observe if the interpreter will run forever, so we have to let it run and hope for eventual termination, either with success or failure.

Note that these are exactly the same even if our interpreter were complete in the strong sense. The only difference would be that if there is a proof, the running interpreter would eventually succeed. It cannot always fail if there is no proof, because of the undecidability of this fragment (which is easy to verify, see Exerciseexc:lhhf-undec). One should note, however, that even simple decidable fragments may exhibit non-terminating behavior.

This observation reminds us that linear logic programming does not provide a general theorem prover. It is not possible to write an arbitrary specification (even in the LHHF fragment) and obtain a reasonable program. Instead, it is often possible to write programs at a very high level of abstraction, and sometimes even possible to few specification directly as programs, but just as often this is not the case. Some examples below should help to clarify this.

5.3 Deterministic Resource Management

In order to use linear hereditary Harrop formulas effectively as a logic programming language, we need to understand how resource non-determinism is resolved. This can be understood in three stages—we give here only the first and most important one.

The only rule where we have to consider how to split resources is the $\multimap$L rule.

$$\frac{\Gamma; \Delta_1; B \Downarrow \Longrightarrow P \qquad \Gamma; \Delta_2 \Longrightarrow A \Uparrow}{\Gamma; \Delta_1, \Delta_2; A \multimap B \Downarrow \Longrightarrow P} \multimap \mathrm{L}$$

Note that the left premise will be solved first and then the right premise. The way we resolve this choice is that we pass all the resources to the left premise and keep track which ones were actually needed in the proof of $B \Downarrow \Longrightarrow P$. The remaining ones are passed to the second premise once the first premise has succeeded. This strategy only make sense because we have already committed to solve conjunctive non-determinism by left-to-right subgoal selection.

This describes the *input/output model* of resource management. We reuse some of the notation introduced to describe Boolean constraints, by writing $u{:}A[1]$ for a linear hypothesis that is there, and $u{:}A[0]$ for a linear hypothesis that has been consumed somewhere. No other Boolean expression arise here. The main judgments are now

$$\begin{array}{c} \Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \Uparrow \\ \Gamma; \Delta_I \backslash \Delta_O; A \Downarrow \Longrightarrow P \end{array}$$

where Δ_I stands for the input resources that may be consumed and Δ_O stands for the output resources that were not consumed. Note that the judgment is hypothetical in Δ_I but not in Δ_O. First, the goal-directed phase of search.

$$\frac{\Gamma; \Delta_I, u{:}A[1] \backslash \Delta_O, u{:}A[0] \Longrightarrow B \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \multimap B \Uparrow} \multimap\mathrm{R}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \Uparrow \qquad \Gamma; \Delta_I \backslash \Delta_O \Longrightarrow B \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \& B \Uparrow} \&\mathrm{R}$$

$$\frac{\Delta_I \supseteq \Delta_O}{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow \top \Uparrow} \top\mathrm{R} \qquad \frac{\Gamma, A; \Delta_I \backslash \Delta_O \Longrightarrow B \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \supset B \Uparrow} \supset\mathrm{R}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow [a/x]A \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow \forall x.\ A \Uparrow} \forall\mathrm{R}^a$$

The right rule for linear implication requires that the linear assumption be consumed ($A[0]$). For $\top$ we have to allow an arbitrary subset of the input hypotheses to be consumed. This relation is defined by

$$\frac{}{\cdot \supseteq \cdot} \qquad \frac{\Delta_I \supseteq \Delta_O}{\Delta_I, u{:}A[0] \supseteq \Delta_O, u{:}A[0]}$$

$$\frac{\Delta_I \supseteq \Delta_O}{\Delta_I, u{:}A[1] \supseteq \Delta_O, u{:}A[1]} \qquad \frac{\Delta_I \supseteq \Delta_O}{\Delta_I, u{:}A[1] \supseteq \Delta_O, u{:}A[0]}$$

The non-determinism that arises in this rule has to be eliminated in a second step (see [CHP00]).[3]

Second, the transition between the phases. Note that linear hypotheses are consumed here, and not at the rules for initial sequents.

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; A \Downarrow \Longrightarrow P}{\Gamma; \Delta_I, u{:}A[1] \backslash \Delta_O, u{:}A[0] \Longrightarrow P \Uparrow} \mathsf{decideL} \qquad \frac{\Gamma, A; \Delta_I \backslash \Delta_O; A \Downarrow \Longrightarrow C}{\Gamma, A; \Delta_I \backslash \Delta_O \Longrightarrow P \Uparrow} \mathsf{decideL!}$$

Third, the focusing phase. The critical rule is the one for $\multimap$L where resources are passed through from left to right with Δ_M representing the hypotheses that have not been consumed in the proof of the first premise. Also note that for initial sequent we simply pass through all linear hypothesis rather than checking if they are empty. This is because we are no longer required, locally, that all linear assumptions are used, since some pending subgoal may consume it later.

[3] [*maybe reformulate and add here*]

$$\frac{}{\Gamma; \Delta_I \backslash \Delta_I; P \Downarrow \Longrightarrow P}\ \mathsf{init}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; B \Downarrow \Longrightarrow P \qquad \Gamma; \Delta_M \backslash \Delta_O \Longrightarrow A \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O; A \multimap B \Downarrow \Longrightarrow P} \multimap \mathrm{L}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; A \Downarrow \Longrightarrow P}{\Gamma; \Delta_I \backslash \Delta_O; A \& B \Downarrow \Longrightarrow P} \&\mathrm{L}_1 \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; B \Downarrow \Longrightarrow P}{\Gamma; \Delta_I \backslash \Delta_O; A \& B \Downarrow \Longrightarrow P} \&\mathrm{L}_2$$

$$\text{no left rule for } \top \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; B \Downarrow \Longrightarrow P \qquad \Gamma; \cdot \Longrightarrow A \Uparrow}{\Gamma; \Delta_I \backslash \Delta_O; A \supset B \Downarrow \Longrightarrow P} \supset\mathrm{L}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; [t/x]A \Downarrow \Longrightarrow P}{\Gamma; \Delta_I \backslash \Delta_O; \forall x.\, A \Downarrow \Longrightarrow P} \forall\mathrm{L}$$

In order to execute a query $\Gamma; \Delta \Longrightarrow A$ we instead execute

$$\Gamma; \Delta[1] \backslash \Delta[0] \Longrightarrow A \Uparrow$$

where $(u_1{:}A_1, \ldots, u_n{:}A_n)[b]$ is shorthand for $u_1{:}A_1[b], \ldots, u_n{:}A_n[b]$. This guarantees that all linear hypotheses have indeed be consumed.

In the statement of soundness and completeness, however, we need to be more general to account for intermediate states. The idea of soundness is that if $\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \Uparrow$ then if we delete all hypotheses from Δ_I that are still in Δ_O, we should have a uniform proof from the resulting hypotheses. We therefore define subtraction, $\Delta_I - \Delta_O = \Delta$, where Δ_I and Δ_O have Boolean annotations and Δ does not.

$$\frac{}{\cdot - \cdot = \cdot} \qquad \frac{\Delta_I - \Delta_O = \Delta}{(\Delta_I, u{:}A[1]) - (\Delta_I, u{:}A[0]) = (\Delta, u{:}A)}$$

$$\frac{\Delta_I - \Delta_O = \Delta}{(\Delta_I, u{:}A[1]) - (\Delta_I, u{:}A[1]) = \Delta} \qquad \frac{\Delta_I - \Delta_O = \Delta}{(\Delta_I, u{:}A[0]) - (\Delta_I, u{:}A[0]) = \Delta}$$

We can then prove soundness directly.

Theorem 5.1 (Soundness of I/O Resource Management)

1. *If* $\Gamma; \Delta_I \backslash \Delta_O \Longrightarrow A \Uparrow$ *then* $\Delta_I - \Delta_O = \Delta$ *and* $\Gamma; \Delta \Longrightarrow A \Uparrow$
2. *If* $\Gamma; \Delta_I \backslash \Delta_O; A \Downarrow \Longrightarrow P$ *then* $\Delta_I - \Delta_O = \Delta$ *and* $\Gamma; \Delta; A \Downarrow \Longrightarrow P$

Proof: By mutual induction on the structure of the given derivation.[4] □

[4] [*check for lemmas and write out some cases*]

For the completness direction we need to generalize the induction hypothesis somewhat differently.

Theorem 5.2 (Completeness of I/O Resource Management)

1. *If* $\Gamma; \Delta \Longrightarrow A \Uparrow$ *then* $\Gamma; \Delta[1], \Delta_O \backslash \Delta[0], \Delta_O \Longrightarrow A \Uparrow$ *for any* Δ_O.

2. *If* $\Gamma; \Delta; A \Downarrow \Longrightarrow P$ *then* $\Gamma; \Delta[1], \Delta_O \backslash \Delta[0], \Delta_O; A \Downarrow \Longrightarrow P$ *for andy* Δ_O.

Proof: By mutual induction on the structure of the given derivation.[5] □

5.4 Some Example Programs

We start with some simple programs. Following the tradition of logic programming, we write implications in the program (Γ) in reverse so that $A \circ\!\!- B$ means $B \multimap A$. Implication in this direction is left associative, and subgoals solved (visually) from left-to-right. So,

$$P \circ\!\!- Q \circ\!\!- R$$

stands for $(P \circ\!\!- Q) \circ\!\!- R$ which is the same as $R \multimap (Q \multimap P)$. If P matches the current atomic goal, then first subgoal to be solved is Q and then R. This is consistent with the informal operational semantics explained above.

The first program is non-terminating for the simple query p.

$u_1 \; : \; p \circ\!\!- p.$
$u_0 \; : \; p.$

Then a goal $\Longrightarrow p$ under this program will diverge, since it will use u_1 first, which produces the identical subgoal of $\Longrightarrow p$. If we reorder the clauses

$u_0 \; : \; p.$
$u_1 \; : \; p \circ\!\!- p.$

the query $\Longrightarrow p$ will produce the immediate proof (u_0) first and, if further answer are requested, succeed arbitrarily often with different proofs. We can slightly complicate this example by adding an argument to p.

$u_0 \; : \; p(0).$
$u_s \; : \; \forall x.\, p(\mathsf{s}(x)) \circ\!\!- p(x).$

In a query we can now leave an existential variable, indicated by an uppercase letter, $\Longrightarrow p(X)$. this query will succeed and print the answer substitution $X = 0$. If further solutions are requested, the program will enumerate $X = \mathsf{s}(0)$, $X = \mathsf{s}(\mathsf{s}(0))$, etc. In general, most logic programming language implementation print only substitutions for existential variables in a query, but not other aspects of the proof it found.

The trivial examples above do not take advantage of the expressive power of linear logic and could equally well be written, for example, in Prolog.

For the next example we introduce lists as terms, using constructors nil and cons. For example, the list $1, 2, 3$ would be written as $\mathsf{cons}(1, \mathsf{cons}(2, \mathsf{cons}(3, \mathsf{nil})))$. A program to enumerate all permutations of a list is the following.

[5][*check for lemmas and write out some cases*]

$$
\begin{array}{lll}
p_0 & : & \mathsf{perm}(\mathsf{cons}(X, L), K) \circ\!- (\mathsf{elem}(X) \multimap \mathsf{perm}(L, K)) \\
p_1 & : & \mathsf{perm}(\mathsf{nil}, \mathsf{cons}(X, K)) \circ\!- \mathsf{elem}(X) \circ\!- \mathsf{perm}(\mathsf{nil}, K) \\
p_2 & : & \mathsf{perm}(\mathsf{nil}, \mathsf{nil})
\end{array}
$$

Here we have left universal quantifiers on X, L, and K implicit in each declaration in order to shorten the program. This is also supported by implementations of logic programming languages.

We assume a query of the form $\Longrightarrow \mathsf{perm}(l, K)$ where l is a list and K is a free existential variable. The program iterates over the list l with p_0, creating a linear hypothesis $\mathsf{elem}(t)$ for every element t of the list. Then it repeatedly uses clause p_1 to consume the linear hypothesis in the output list K. When there are no longer any linear hypotheses, the last clause p_2 will succeed and therefore the whole program.

Bibliography

[ABCJ94] D. Albrecht, F. Bäuerle, J. N. Crossley, and J. S. Jeavons. Curry-Howard terms for linear logic. In ??, editor, *Logic Colloquium '94*, pages ??–?? ??, 1994.

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[ACS98] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchrounous π-calculus. *Theoretical Computer Science*, 195(2):291–423, 1998.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

[AP91] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems: A linear logic approach. In P. Schröder-Heister, editor, *Proceedings of Workshop to Extensions of Logic Programming, Tübingen, 1989*, pages 1–30. Springer-Verlag LNAI 475, 1991.

[Bar96] Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Department of Computer Science, University of Edinburgh, September 1996.

[Bib86] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Bie94] G. Bierman. On intuitionistic linear logic. Technical Report 346, University of Cambridge, Computer Laboratory, August 1994. Revised version of PhD thesis.

[BS92] G. Bellin and P. J. Scott. On the π-calculus and linear logic. Manuscript, 1992.

[Cer95] Iliano Cervesato. Petri nets and linear logic: a case study for logic programming. In M. Alpuente and M.I. Sessa, editors, *Proceedings of the Joint Conference on Declarative Programming (GULP-PRODE'95)*, pages 313–318, Marina di Vietri, Italy, September 1995. Palladio Press.

[CHP00] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1–2):133–163, February 2000. Special issue on Proof Search in Type-Theoretic Languages, D. Galmiche and D. Pym, editors.

[Doš93] Kosta Došen. A historical introduction to substructural logics. In Peter Schroeder-Heister and Kosta Došen, editors, *Substructural Logics*, pages 1–30. Clarendon Press, Oxford, England, 1993.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. Translated under the title *Investigations into Logical Deductions* in [Sza69].

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

[Gir93] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[Her30] Jacques Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et de Lettres de Varsovic*, 33, 1930.

[HM94] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. A preliminary version appeared in the Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science, pages 32–42, Amsterdam, The Netherlands, July 1991.

[Hod94] Joshua S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[HP97] James Harland and David Pym. Resource-distribution via boolean constraints. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, pages 222–236, Townsville, Australia, July 1997. Springer-Verlag LNAI 1249.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'91)*, pages 133–147, Geneva, Switzerland, July 1991. Springer-Verlag LNCS 512.

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, September 1976.

[Kni89] Kevin Knight. Unification: A multi-disciplinary survey. *ACM Computing Surveys*, 2(1):93–124, March 1989.

[Lin92] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[Mil92] D. Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of the Workshop on Extensions of Logic Programming*, pages 242–265. Springer-Verlag LNCS 660, 1992.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[MM76] Alberto Martelli and Ugo Montanari. Unification in linear time and space: A structured presentation. Internal Report B76-16, Istituto di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, July 1976.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[MOM91] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *Journal on Foundations of Computer Science*, 2(4):297–399, December 1991.

[PD01] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

[Pra65] Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

[PW78] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, April 1978.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Rob71] J. A. Robinson. Computational logic: The unification computation. *Machine Intelligence*, 6:63–72, 1971.

[Sce93] A. Scedrov. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Company, 1993. Also in Bulletin of the European Association for Theoretical Computer Science, volume 41, pages 154–165.

[SHD93] Peter Schroeder-Heister and Kosta Došen, editors. *Substructural Logics*. Number 2 in Studies in Logic and Computation. Clarendon Press, Oxford, England, 1993.

[Sza69] M. E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. North-Holland Publishing Co., Amsterdam, 1969.

[Tro92] A. S. Troelstra. *Lectures on Linear Logic*. CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.

[Tro93] A. S. Troelstra. Natural deduction for intuitionistic linear logic. Prepublication Series for Mathematical Logic and Foundations ML-93-09, Institute for Language, Logic and Computation, University of Amsterdam, 1993.