6.5 Termination

As the example at the end of the previous section shows, unrestricted recursive types destroy the normalization property. This also means it is impossible to give all recursive types a logical interpretation. When we examine the inference rules we notice that recursive types are *impredicative*: the binder $\mu\alpha$ in $\mu\alpha.\ A$ ranges over the whole type. This means in the introduction rule, the type in the premiss $[\mu\alpha.\ A/\alpha]A$ generally will be larger than the type $\mu\alpha.\ A$ in the conclusion. That alone is not responsible for non-termination: there are other type disciplines such as the polymorphic λ-calculus which retain a logical interpretation and termination, yet are impredicative.

In this section we focus on the property that all well-typed terms in the linear λ-calculus *without* recursive types and fixpoint operators evaluate to a value. This is related to the normalization theorem for natural deductions (Theorem 3.10): if $\Gamma;\Delta \vdash A$ then $\Gamma;\Delta \vdash A \uparrow$. We proved this by a rather circuitous route: unrestricted natural deductions can be translated to sequent derivations with cut from which we can eliminate cut and translate the result cut-free derivation back to a noraml natural deduction.

Here, we prove directly that every term evaluates using the proof technique of *logical relations* [Sta85] also called *Tait's method* [Tai67]. Because of the importance of this technique, we spend some time motivating its form. Our ultimate goal is to prove:

If $\cdot;\cdot \vdash M : A$ *then* $M \hookrightarrow V$ *for some value* V.

The first natural attempt would be to prove this by induction on the typing derivation. Surprisingly, case for $\multimap$I works, even though we cannot apply the inductive hypothesis, since every linear λ-abstraction immediately evaluates to itself.

In the case for $\multimap$E, however, we find that we cannot complete the proof. Let us examine why.

Case: $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1\\ \cdot;\cdot \vdash M_1 : A_2 \multimap A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \cdot;\cdot \vdash M_2 : A_2\end{array}}{\cdot;\cdot \vdash M_1\,\hat{}\,M_2 : A_1}\multimap\mathrm{E}.$$

We can make the following inferences.

$M_1 \hookrightarrow V_1$ for some V_1	By ind. hyp. on $\mathcal{D}_1$
$V_1 = \hat{\lambda} x{:}A_2.\ M_1'$	By type preservation and inversion
$M_2 \hookrightarrow V_2$ for some V_2	By ind. hyp. on $\mathcal{D}_2$

At this point we cannot proceed: we need a derivation of

$[V_2/x]M_1' \hookrightarrow V$ *for some* V

to complete the derivation of $M_1\,\hat{}\,M_2 \hookrightarrow V$. Unfortunately, the induction hypothesis does not tell us anything about $[V_2/x]M_1'$. Basically, we need to extend it so it makes a statement about the *result* of evaluation ($\hat{\lambda}x{:}A_2.\ M_1'$, in this case).

Sticking to the case of linear application for the moment, we call a term M "good" if it evaluates to a "good" value V. A value V is "good" if it is a function $\hat{\lambda}x{:}A_2.\ M_1'$ and if substituting a "good" value V_2 for x in M_1' results in a "good" term. Note that this is not a proper definition, since to see if V is "good" we may need to substitute any "good" value V_2 into it, possibly including V itself. We can make this definition inductive if we observe that the value V_2 will be of type A_2, while the value V we are testing has type $A_2 \multimap A_1$, and that the resulting term as type A_1. That is, we can fashion a definition which is inductive on the structure of the *type*. Instead of saying "good" we say $M \in \|A\|$ and $v \in |A|$. Still restricting ourselves to linear implication only, we define:

$$\begin{array}{rcl} M \in \|A\| & \text{iff} & M \hookrightarrow V \text{ and } V \in |A| \\ M \in |A_2 \multimap A_1| & \text{iff} & M = \hat{\lambda}x{:}A_2.\ M_1 \text{ and } [V_2/x]M_1 \in \|A_1\| \text{ for any } V_2 \in |A_2| \end{array}$$

From $M \in \|A\|$ we can immediately infer $M \hookrightarrow V$ for some V, so when proving that $\cdot;\cdot \vdash M : A$ implies $M \in \|A\|$ we do indeed have a much stronger induction hypothesis.

While the case for application now goes through, the case for linear λ-abstraction fails, since we cannot prove the stronger property for the value.

Case: $$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ \cdot; x{:}A_2 \vdash M_1 : A_1\end{array}}{\cdot;\cdot \vdash \hat{\lambda}x{:}A_2.\ M_1 : A_2 \multimap A_1} \multimap \mathrm{I}.$$

Then $\hat{\lambda}x{:}A_2.\ M_1 \hookrightarrow \hat{\lambda}x{:}A_2.\ M_1$ and it remains to show that for every $V_2 \in |A_2|$, $[V_2/x]M_2 \in \|A_1\|$.

This last statement should follow from the induction hypothesis, but presently it is too weak since it only allows for closed terms. The generalization which suggests itself from this case (ignoring the unrestricted context for now) is:

If $\Delta \vdash M : A$, then for any substitution θ which maps the linear variables $x{:}A$ in Δ to values $V \in |A|$, $[\theta]M \in \|A\|$.

This generalization indeed works after we also account for the unrestricted context. During evaluation we substitute values for linear variables and expressions for unrestricted variables. Therefore, the substitutions we must consider for the induction hypothesis have to behave accordingly.

$$\begin{array}{rrcl} \textit{Unrestricted Substitution} & \eta & ::= & \cdot \mid \eta, M/u \\ \textit{Linear Substitution} & \theta & ::= & \cdot \mid \theta, V/x \end{array}$$

We write $[\eta;\theta]M$ for the simultaneous application of the substitutions η and θ to M. For our purposes here, the values and terms in the substitutions are always closed, but we do not need to postulate this explicitly. Instead, we only deal with substitution satisfying the property necessary for the generalization of the induction hypothesis.

$$\begin{array}{lll} \theta \in |\Delta| & \text{iff} & [\theta]x \in |A| \text{ for every } x{:}A \text{ in } \Delta \\ \eta \in \|\Gamma\| & \text{iff} & [\eta]u \in \|A\| \text{ for every } u{:}A \text{ in } \Gamma \end{array}$$

We need just one more lemma, namely that values evaluate to themselves.

Lemma 6.10 (Value Evaluation) *For any value v, $v \hookrightarrow v$*

Proof: See Exercise 6.18. □

Now we have all ingredients to state the main lemma in the proof of termination, the so called *logical relations lemma* [Sta85]. The "logical relations" are $\|A\|$ and $|A|$, seen as unary relations, that is, predicates, on terms and values, respectively. They are "logical" since they are defined by induction on the structure of the type A, which corresponds to a proposition under the Curry-Howard isomorphism.

Lemma 6.11 (Logical Relations) *If $\Gamma; \Delta \vdash M : A$, $\eta \in \|\Gamma\|$ and $\theta \in |\Delta|$ then $[\eta;\theta]M \in \|A\|$.*

Before showing the proof, we extend the definition of the logical relations to all the types we have been considering.

$$\begin{array}{rll} M \in \|A\| & \text{iff} & M \hookrightarrow V \text{ and } V \in |A| \\ V \in |A_2 \multimap A_1| & \text{iff} & V = \hat{\lambda}x{:}A_2.\ M_1 \text{ and } [V_2/x]M_1 \in \|A_1\| \text{ for any } V_2 \in |A_2| \\ V \in |A_1 \otimes A_2| & \text{iff} & V = V_1 \otimes V_2 \text{ where } V_1 \in |A_1| \text{ and } V_2 \in |A_2| \\ V \in |\mathbf{1}| & \text{iff} & V = \star \\ V \in |A_1 \& A_2| & \text{iff} & V = \langle M_1, M_2 \rangle \text{ where } M_1 \in \|A_1\| \text{ and } M_2 \in \|A_2\| \\ V \in |\top| & \text{iff} & V = \langle\,\rangle \\ V \in |A_1 \& A_2| & \text{iff} & \text{either } V = \mathsf{inl}^{A_2}\, V_1 \text{ and } V_1 \in |A_1|, \\ & & \text{or } V = \mathsf{inr}^{A_1}\, V_2 \text{ and } V_2 \in |A_2| \\ V \in |\mathbf{0}| & & \text{never} \\ V \in |!A| & \text{iff} & V = !M \text{ and } M \in \|A\| \\ V \in |A_2 \to A_1| & \text{iff} & V = \lambda u{:}A_2.\ M_1 \text{ and } [M_2/u]M_1 \in \|A_1\| \\ & & \text{for any } M_2 \in \|A_2\| \end{array}$$

These definitions are motivated directly from the form of values in the language. One can easily see that it is indeed inductive on the structure of the type. If we tried to add recursive types in a similar way, the proof below would still go through, except that the definition of the logical relation would no longer be well-founded.

Proof: (of the logical relations lemma 6.11). The proof proceeds by induction on the structure of the typing derivation $\mathcal{D} :: (\Gamma; \Delta \vdash M : A)$. We show three cases—all others are similar.

Case: $\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma; \Delta \vdash M_1 : A_2 \multimap A_1 & \Gamma; \Delta \vdash M_2 : A_2 \end{array}}{\Gamma; \Delta \vdash M_1 \hat{\ } M_2 : A_1} \multimap \mathrm{E}.$

$\eta \in \lVert\Gamma\rVert$	Assumption
$\theta \in \lvert\Delta\rvert$	Assumption
$[\eta;\theta]M_1 \in \lVert A_2 \multimap A_1\rVert$	By ind. hyp. on $\mathcal{D}_1$
$\mathcal{E}_1 :: ([\eta;\theta]M_1 \hookrightarrow V_1)$ and $V_1 \in \lvert A_2 \multimap A_1\rvert$	By definition of $\lVert A_2 \multimap A_1\rVert$
$V_1 = \hat{\lambda}x{:}A_1.\ M_1'$ and	
$[V_2/x]M_1' \in \lVert A_1\rVert$ for any $V_2 \in \lvert A_2\rvert$	By definition of $\lvert A_2 \multimap A_1\rvert$
$[\eta;\theta]M_2 \in \lVert A_2\rVert$	By ind. hyp. on $\mathcal{D}_2$
$\mathcal{E}_2 :: ([\eta;\theta]M_2 \hookrightarrow V_2)$ and $V_2 \in \lvert A_2\rvert$	By definition of $\lVert A_2\rVert$
$[V_2/x]M_1' \in \lVert A_1\rVert$	Since $V_2 \in \lvert A_2\rvert$
$\mathcal{E}_3 :: ([V_2/x]M_1' \hookrightarrow V)$ and $V \in \lvert A_1\rvert$	by definition of $\lVert A_1\rVert$
$\mathcal{E} :: ([\eta;\theta](M_1 \hat{\ } M_2) \hookrightarrow V)$	by $\multimap$Ev from $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$
$[\eta;\theta](M_1 \hat{\ } M_2) \in \lVert A_1\rVert$	by definition of $\lVert A_1\rVert$

Case: $\mathcal{D} = \dfrac{\begin{array}{c} \mathcal{D}_1 \\ \Gamma; (\Delta, x{:}A_2) \vdash M_1 : A_1 \end{array}}{\Gamma; \Delta \vdash \hat{\lambda}x{:}A_2.\ M_1 : A_2 \multimap A_1} \multimap \mathrm{I}.$

$\eta \in \lVert\Gamma\rVert$	by assumption
$\theta \in \lvert\Delta\rvert$	by assumption
$\mathcal{E} :: ([\eta;\theta](\hat{\lambda}x{:}A_2.\ M_1) \hookrightarrow [\eta;\theta](\hat{\lambda}x{:}A_2.\ M_1))$	by $\multimap$Iv
$V_2 \in \lvert A_2\rvert$	assumption
$(\theta, V_2/x) \in \lvert\Delta, x{:}A_2\rvert$	by definition of $\lvert\Delta\rvert$
$[\eta; (\theta, V_2/x)]M_1 \in \lVert A_1\rVert$	by ind. hyp. on $\mathcal{D}_1$
$[V_2/x]([\eta; (\theta, x/x)]M_1) \in \lVert A_1\rVert$	by properties of substitution
$(\hat{\lambda}x{:}A_2.\ [\eta; (\theta, x/x)]M_1) \in \lvert A_2 \multimap A_1\rvert$	by definition of $\lvert A_2 \multimap A_1\rvert$
$[\eta;\theta](\hat{\lambda}x{:}A_2.\ M_1) \in \lvert A_2 \multimap A_1\rvert$	by properties of substitution
$[\eta;\theta](\hat{\lambda}x{:}A_2.\ M_1) \in \lVert A_2 \multimap A_1\rVert$	by definition of $\lVert A_2 \multimap A_1\rVert$

Case: $\mathcal{D} = \dfrac{}{\Gamma; x{:}A \vdash x : A}\,\mathsf{hyp}$

$\theta \in \lvert\cdot, x{:}A\rvert$	by assumption
$[\theta]x \in \lvert A\rvert$	by definition of $\lvert\cdot, x{:}A\rvert$
$\mathcal{E} :: ([\eta;\theta]x \hookrightarrow [\eta;\theta]x)$	by Lemma 6.10
$[\eta;\theta]x \in \lVert A\rVert$	by definition of $\lVert A\rVert$

□

The termination theorem follows directly from the logical relations lemma. Note that the theorem excludes recursive types and the fixpoint operator by a general assumption for this section.

Theorem 6.12 (Termination) *If* $\cdot;\cdot \vdash M : A$ *then* $M \hookrightarrow V$ *for some* V.

Proof: We have $\cdot \in \|\cdot\|$ and $\cdot \in |\cdot|$ since the conditions are vacuously satisfied. Therefore, by the logical relations lemma 6.11, $[\cdot;\cdot]M \in \|A\|$. By the definition of $\|A\|$ and the observation that $[\cdot;\cdot]M = M$, we conclude that $M \hookrightarrow V$ for some V. □

6.6 Exercises

Exercise 6.1 Prove that if $\Gamma;\Delta \vdash M : A$ and $\Gamma;\Delta \vdash M : A'$ then $A = A'$.

Exercise 6.2 A function in a functional programming language is called *strict* if it is guaranteed to use its argument. Strictness is an important concept in the implementation of lazy functional languages, since a strict function can evaluate its argument eagerly, avoiding the overhead of postponing its evaluation and later memoizing its result.

In this exercise we design a λ-calculus suitable as the core of a functional language which makes strictness explicit at the level of types. Your calculus should contain an unrestricted function type $A \rightarrow B$, a strict function type $A \twoheadrightarrow B$, a vacuous function type $A \dashrightarrow B$, a full complement of operators refining product and disjoint sum types as for the linear λ-calculus, and a modal operator to internalize the notion of closed term as in the linear λ-calculus. Your calculus should not contain quantifiers.

1. Show the introduction and elimination rules for all types, including their proof terms.
2. Given the reduction and expansions on the proof terms.
3. State (without proof) the valid substitution principles.
4. If possible, give a translation from types and terms in the strict λ-calculus to types and terms in the linear λ-calculus such that a strict term is well-typed if and only if its linear translation is well-typed (in an appropriately translated context).
5. Either sketch the correctness proof for your translation in each direction by giving the generalization (if necessary) and a few representative cases, or give an informal argument why such a translation is not possible.

Exercise 6.3 Give an example which shows that the substitution $[M/w]N$ must be capture-avoiding in order to be meaningful. *Variable capture* is a situation where a bound variable w' in N occurs free in M, and w occurs in the scope of w'. A similar definition applies to unrestricted variables.

Exercise 6.4 Give a counterexample to the conjecture that if $M \longrightarrow_\beta M'$ and $\Gamma;\Delta \vdash M' : A$ then $\Gamma;\Delta \vdash M : A$. Also, either prove or find a counterexample to the claim that if $M \longrightarrow_\eta M'$ and $\Gamma;\Delta \vdash M' : A$ then $\Gamma;\Delta \vdash M : A$.

Exercise 6.5 The proof term assignment for sequent calculus identifies many distinct derivations, mapping them to the same natural deduction proof terms. Design an alternative system of proof terms from which the sequent derivation can be reconstructed uniquely (up to weakening of unrestricted hypotheses and absorption of linear hypotheses in the $\top R$ rule).

1. Write out the term assignment rules for all propositional connectives.
2. Give a calculus of reductions which corresponds to the initial and principal reductions in the proof of admissibility of cut.
3. Show the reduction rule for the dereliction cut.
4. Show the reduction rules for the left and right commutative cuts.
5. Sketch the proof of the subject reduction properties for your reduction rules, giving a few critical cases.
6. Write a translation judgment $S \Longrightarrow M$ from faithful sequent calculus terms to natural deduction terms.
7. Sketch the proof of type preservation for your translation, showing a few critical cases.

Exercise 6.6 Supply the missing rules for $\oplus$E in the definition of the judgment $\Gamma; \Delta_I \setminus \Delta_O \vdash_i M : A$ and show the corresponding cases in the proof of Lemma 6.4.

Exercise 6.7 In this exercise we explore the syntactic expansion of *extended case expressions* of the form $\mathsf{case}\, M \,\mathsf{of}\, m$.

1. Define a judgment which checks if an extended case expression is valid. This is likely to require some auxiliary judgments. You must verify that the cases are exhaustive, circumscribe the legal patterns, and check that the overall expression is linearly well-typed.
2. Define a judgment which relates an extended case expression to its expansion in terms of the primitive let, case, and abort constructs in the linear λ-calculus.
3. Prove that an extended case expression which is valid according to your criteria can be expanded to a well-typed linear λ-term.
4. Define an operational semantics directly on extended case expressions.
5. Prove that your direct operational semantics is correct on valid patterns with respect to the translational semantics from questions 2.

Exercise 6.8 Define the judgment $M \longrightarrow_\beta^* M'$ via inference rules. The rules should directly express that it is the congruent, reflexive and transitive closure of the β-reduction judgment $M \longrightarrow_\beta M'$. Then prove the generalized subject reduction theorem 6.6 for your judgment. You do not need to show all cases, but you should carefully state your induction hypothesis in sufficient generality and give a few critical parts of the proof.

Exercise 6.9 Define *weak β-reduction* as allows simple β-reduction under $\otimes$, inl, and inr constructs and in all components of the elimination form. Show that if M weakly reduces to a value v then $M \hookrightarrow v$.

Exercise 6.10 Prove type preservation (Theorem 6.8) directly by induction on the structure of the evaluation derivation, using the substitution lemma 6.2 as necessary, but without appeal to subject reduction.

Exercise 6.11 Prove the subject reduction and expansion properties for recursive type computation rules.

Exercise 6.12 [*An exercise exploring the use of type conversion rules without explicit term constructors.*]

Exercise 6.13 Define a linear multiplication function mult : nat $\multimap$ nat $\multimap$ nat using the functions copy and delete.

Exercise 6.14 Defined the following functions on lists. Always explicitly state the type, which should be the most natural type of the function.

1. append to append two lists.
2. concat to append all the lists in a list of lists.
3. map to map a function f over the elements of a list. The result of mapping f over the list $x_1, x_2, \ldots, x_n$ should be the list $f(x_1), f(x_2), \ldots f(x_n)$, where you should decide if the application of f to its argument should be linear or not.
4. foldr to reduce a list by a function f. The result of folding f over a list $x_1, x_2, \ldots x_n$ should be the list $f(x_1, f(x_2, \ldots, f(x_n, \mathit{init})))$, where *init* is an initial value given as argument to foldr. You should decide if the application of f to its argument should be linear or not.
5. copy, delete, and promote.

Exercise 6.15 For one of the form of lazy lists on Page 130, define the functions from Exercise 6.14 plus a function toList which converts the lazy to an eager list (and may therefore not terminate if the given lazy lists is infinite). Make sure that your functions exhibit the correct amount of laziness. For example, a map function applied to a lazy list should not carry out any non-trivial computation until the result is examined.

Further for your choice of lazy list, define the infinite lazy list of eager natural numbers $0, 1, 2, \ldots$.

Exercise 6.16 Prove that there is no term v such that $\omega\,(\mathrm{fold}^{\Omega}\,\omega) \hookrightarrow v$.

Exercise 6.17 [*An exercise about the definability of fixpoint operators at various type.*]

Exercise 6.18 Prove Lemma 6.10 which states that all values evaluate to themselves.

Exercise 6.19 In this exercise we explore strictness as a derived, rather than a primitive concept. Recall that a function is *strict* if it uses its argument at least once. The strictness of a function from A to B can be enforced by the type $(A \otimes {!A}) \multimap B$.

1. Show how to represent a strict function $\lambda^s x{:}A.\ M$ under this encoding.
2. Show how to represent an application of a strict function M to an argument N.
3. Give natural evaluation rules for strict functions and strict applications.
4. Show the corresponding computation under the encoding of strict functions in the linear λ-calculus.
5. Discuss the merits and difficulties of the given encoding of the strict in the linear λ-calculus.

Exercise 6.20 In the exercise we explore the *affine* λ-calculus. In an affine hypothetical judgment, each assumption can be used at most once. Therefore, it is like linear logic except that affine hypotheses need not be used.

The affine hypothetical judgment $\Gamma; \Delta \vdash^a A\ \mathit{true}$ is characterized by the hypothesis rule

$$\frac{}{\Gamma; \Delta, x{:}A\ \mathit{true} \vdash^a A\ \mathit{true}}$$

and the substitution principle

if $\Gamma; \Delta \vdash^a A\ \mathit{true}$ *and* $\Gamma; \Delta', A\ \mathit{true} \vdash^a C\ \mathit{true}$ *then* $\Gamma; \Delta, \Delta' \vdash^a C\ \mathit{true}$.

1. State the remaining hypothesis rule and substitution principle for unrestricted hypotheses.
2. Give introduction and elimination rules for affine implication $(A \rightsquigarrow B)$ simultaneous conjunction, alternative conjunction, and truth. Note that there is only one form of truth: since assumptions need not be used, the multiplicative and additive form coincide.
3. Give a proof term assignment for affine logic.
4. We can map affine logic to linear logic by translating every affine function $A \rightsquigarrow B$ to a linear function $(A \& 1) \multimap B$. Give a corresponding translation for all proof terms from the affine logic to linear logic.

5. We can also map affine logic to linear logic by translating every affine function $A \rightsquigarrow B$ into function $A \multimap (B \otimes \top)$. Again give a corresponding translation for all proof terms from affine logic to linear logic.

6. Discuss the relative merits of the two translations.

7. [Extra Credit] Carefully formulate and prove the correctness of one of the two translations.

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.

[AK91] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press, 1991.

[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.

[AS01] Klaus Aehlig and Helmut Schwichtenberg. A syntactical analysis of non-size-increasing polynomial time computation. *Submitted*, 2001. A previous version presented at LICS'00.

[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.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, 1958.

[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.

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, Princeton, New Jersey, 1941.

[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.

[Hof00a] Martin Hofmann. Linear types and non-size increasing polynomial time computation. *Theoretical Computer Science*, 2000. To appear. A previous version was presented at LICS'99.

[Hof00b] Martin Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, November 2000. To appear. A previous version was presented as ESOP'00.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard.

[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.

[ML94] Per Martin-Löf. Analytic and synthetic judgements in type theory. In Paolo Parrini, editor, *Kant and Contemporary Epistemology*, pages 87–99. Kluwer Academic Publishers, 1994.

[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.

[Sta85] Richard Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

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

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal Of Symbolic Logic*, 32:198–212, 1967.

[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.

[WW01] David Walker and Kevin Watkins. On linear types and regions. In *Proceedings of the International Conference on Functional Programming (ICFP'01)*. ACM Press, September 2001.