Proof Induction on $r \in \mathsf{SN}$.

$x\vec{r}$. By induction hypothesis, $\Gamma_i \vdash r_i : \rho_i$ for every i. Therefore

$$\Gamma_1 + \ldots + \Gamma_n + x : (\rho_1 \to \ldots \to \rho_n \to \rho) \vdash x\vec{r} : \rho$$

for any type ρ.

$\lambda x r$. Let $\lambda x r \in \mathsf{SN}$ due to $r \in \mathsf{SN}$. By induction hypothesis, $\Gamma \vdash r : \rho$. Possibly by Weakening 1, we may assume that $\Gamma = \hat{\Gamma}, x : \sigma$. This yields $\hat{\Gamma} \vdash \lambda x r : \sigma \to \rho$.

$(\lambda x r)s\vec{s}$. Assume $(\lambda x r)s\vec{s} \in \mathsf{SN}$ has been derived from $r[x := s]\vec{s} \in \mathsf{SN}$ and $s \in \mathsf{SN}$. By induction hypothesis, $\Gamma \vdash r[x := s]\vec{s} : \tau$ and $\Gamma' \vdash s : \sigma$. Setting $\Delta := \Gamma + \Gamma'$, we get $\Delta \vdash r[x := s]\vec{s} : \tau$ and $\Delta \vdash s : \sigma$. Writing $r[x := s]\vec{s} = (y\vec{s})[y := r[x := s]]$ and applying Lemma 21(b), we find a type ρ such that $\Delta \vdash y\vec{s} : \tau$ and $\Delta \vdash r[x := s] : \rho$. Again by Lemma 21(b), there is a type σ' such that $\Delta, x : \sigma' \vdash r : \rho$ and $\Delta \vdash s : \sigma'$. Hence, $\Delta \vdash (\lambda x r)s : \rho$ and by Lemma 21(a), $\Delta \vdash (\lambda x r)s\vec{s} : \tau$. □

7 Parametric Polymorphism

Although the situation with intersection types is quite satisfying since the associated typing system exactly captures the strongly normalizing terms, there is interest for other typing systems which type only strongly normalizing terms but fail to type all of them. Why and what for? By intersection typing, one can model that a term has finitely many types in parallel. But very often, terms have infinitely many types all being instances of some pattern. The easiest example is the identity $\lambda x x$ which may receive any type of the form $\rho \to \rho$ in the empty context. It is a natural idea to allow universal quantification over type identifiers (the elements of $\mathcal{V}_T$). In this example, we would assume $\alpha \in \mathcal{V}_T$ and give the type $\forall\alpha.\alpha \to \alpha$ to $\lambda x x$ (in the empty context). And we may instantiate this for any type ρ instead of α to get $\vdash \lambda x x : \rho \to \rho$. The universal quantifier thus expresses the parametric polymorphism of the identity: For every type ρ (which is the parameter) the identity acts as an element of the function space $\rho \to \rho$, hence belongs in some sense to many different spaces, but in a uniform fashion, namely in spaces described uniformly by a type expression depending on the parameter ρ. In the following, this intuition will be made precise. And since we want to study the idea of parametric polymorphism in isolation, we abandon the ad hoc polymorphism stemming from intersection typing.

Definition 19 (Universal types) *The set $\mathcal{T}_u$ of universal types is inductively given by (we will always assume that $\mathcal{V}_T$ is an infinite set):*

- *If $\alpha \in \mathcal{V}_T$ then $\alpha \in \mathcal{T}_u$.*
- *If $\rho \in \mathcal{T}_u$ and $\sigma \in \mathcal{T}_u$ then $(\rho \to \sigma) \in \mathcal{T}_u$.*
- *If $\alpha \in \mathcal{V}_T$ and $\rho \in \mathcal{T}_u$ then $\forall\alpha\rho \in \mathcal{T}_u$.*

Note that the usual name for type identifiers is now changed from ι to α (β and γ will also be used) which emphasizes their variable nature.

Definition 20 (Free type variables) *Define the set $FV(\rho)$ of type variables occurring free in ρ by recursion on ρ:*

- $FV(\alpha) := \{\alpha\}$.
- $FV(\rho \to \sigma) := FV(\rho) \cup FV(\sigma)$.
- $FV(\forall\alpha\rho) := FV(\rho) \setminus \{\alpha\}$.

As is indicated by the preceding definition, the universal quantifier $\forall$ is considered as a binder like the λ in terms. We will follow the same conventions concerning the irrelevance of the given variable name, e.g., we syntactically identify the types $\forall\alpha.\alpha \to \alpha$ and $\forall\beta.\beta \to \beta$ and also use (like in the previous examples) the dot notation for "invisible parentheses".

Definition 21 (Type substitution) *Define the result $\rho[\alpha := \sigma]$ of replacing every free occurrence of the variable α in ρ by the type σ recursively as follows:*

- $\alpha[\alpha := \sigma] := \sigma$.
- $\beta[\alpha := \sigma] := \beta$ *for* $\beta \neq \alpha$.
- $(\rho \to \tau)[\alpha := \sigma] := \rho[\alpha := \sigma] \to \tau[\alpha := \sigma]$.
- $(\forall\beta\rho)[\alpha := \sigma] := \forall\beta.\rho[\alpha := \sigma]$ *where we may assume by renaming of the bound type variable α that $\alpha \notin \{\beta\} \cup FV(\sigma)$.*

Let us describe the typing system known under the name "system F in Curry-style [Bar93]", and overload the symbol $\vdash$ once more.

Definition 22 (Universal typing) *The relation $\Gamma \vdash r : \rho$ (term r has type ρ in context Γ) is inductively defined by the following rules:*

$$\frac{}{\Gamma, x : \rho \vdash x : \rho}(\mathrm{V}) \qquad \frac{\Gamma, x : \rho \vdash r : \sigma}{\Gamma \vdash \lambda x r : \rho \to \sigma}(\to_\mathrm{I}) \qquad \frac{\Gamma \vdash r : \rho \to \sigma \qquad \Gamma \vdash s : \rho}{\Gamma \vdash rs : \sigma}(\to_\mathrm{E})$$

$$\frac{\Gamma \vdash r : \rho \qquad \alpha \notin \Gamma}{\Gamma \vdash r : \forall\alpha\rho}(\forall_\mathrm{I}) \qquad \frac{\Gamma \vdash r : \forall\alpha\rho \qquad \sigma \in \mathcal{T}_u}{\Gamma \vdash r : \rho[\alpha := \sigma]}(\forall_\mathrm{E})$$

The conventions concerning contexts are as before, and since contexts are lists of declarations for term variables, $\alpha \notin FV(\Gamma)$ shall mean that $\alpha \notin FV(\rho)$ for ρ being one of the assigned types in Γ.

The rule $(\forall_\mathrm{I})$ of $\forall$-introduction has the proviso called "eigenvariable condition" that nothing depending on α may have been assumed on the free variables in r. Otherwise, we would derive $x : \alpha \vdash x : \forall\alpha\alpha$, then $x : \alpha \vdash x : \rho$ for any type ρ, and finally $\vdash \lambda x x : \alpha \to \rho$ for any ρ which we never had in mind. The $\forall$-elimination rule $(\forall_\mathrm{E})$ expresses that every instance of ρ (with ρ replaced by

any type σ) is also a type of r if it received the type $\forall\alpha\rho$. Note that the type σ may again involve universal quantifiers, e.g., we have that

$$\vdash \lambda x x : (\forall\alpha.\alpha \to \beta) \to (\forall\alpha.\alpha \to \beta)$$

and hence $\vdash \lambda x x : \forall\beta.(\forall\alpha.\alpha \to \beta) \to (\forall\alpha.\alpha \to \beta)$. We may also type ω: E.g., $\vdash \omega : \forall\alpha\alpha \to \forall\alpha\alpha$.

We again have Exchange 1 (p. 23) as part of our understanding of contexts. It is quite easy to establish Weakening 1 and Strengthening (see Lemma 20), Lemma 21(a) and Subject Reduction (see Lemma 23).

7.1 Strong Normalization of Typable Terms

We again want to establish strong normalization, i.e., that whenever $\Gamma \vdash r : \rho$ then $r \in \mathsf{SN}$. For proof-theoretic reasons which require the strength of the metatheory exceeding that of second-order arithmetic[6], our direct proof for intersection typing in section 6.3 that SN is closed under typed substitution cannot work. Therefore, the powerful and versatile candidate method (also called computability predicates method) is introduced by which a stronger statement than $r \in \mathsf{SN}$ for every typable term r is proved. For this we need the concept of saturated sets which are subsets of SN with good closure properties. First we define them, then explain the method, and finally apply the method to our setting.

7.1.1 Saturated sets

The following concept was first used by Tait [Tai75].

Definition 23 (Saturated set) *A set $\mathcal{M}$ of terms is* saturated *if the following conditions are met:*

1. *If $r \in \mathcal{M}$ then $r \in \mathsf{SN}$.*
2. *If $\vec{r} \subset \mathsf{SN}$ then $x\vec{r} \in \mathcal{M}$.*
3. *If $r[x := s]\vec{s} \in \mathcal{M}$ and $s \in \mathsf{SN}$ then $(\lambda x r)s\vec{s} \in \mathcal{M}$.*

This definition is nearly the same as that in [Bar93, p. 177].[7] Let SAT be the set of saturated sets. Trivially, $\mathsf{SN} \in \mathsf{SAT}$.

The transformation of the definition of SN into that of saturatedness is well motivated by typing considerations: It would be possible to relativize the definition to τ-saturatedness for a type τ by restricting to terms of type τ (in a given context) only. For this to work it is essential to have no requirement in the rules that a term of a different type shall be in $\mathcal{M}$ in order to conclude that some term belongs to $\mathcal{M}$. Therefore we omit the rule which changes the type (hence leave out the abstraction rule), and in the other rules replace every occurrence of SN by $\mathcal{M}$ in the conclusion and for the terms in the premisses receiving the same type. In general, the other terms do not get the same type and therefore can only be required to belong to SN.

The candidate method goes as follows: By means of saturated sets we define (by recursion on types) predicates of strong computability with respect to an assignment of saturated sets for type variables (a candidate assignment) and finally show (by induction on typings) that every typable term is strongly computable under substitution. Hence every typable term is contained in a saturated set (the computability predicate) which only consists of strongly normalizing terms (due to $\mathsf{SN} \subseteq \mathsf{sn}$).

This all works if there are constructions for saturated sets corresponding to the type constructs of the system for which the introduction rules and the elimination rules are sound. (This is presented at length in [Mat98].)

[6] a consequence of Gödel's second incompleteness theorem, see chapter 15 in [GLT89]

[7] Note, however, that in our definition SN stands for the syntax-directed definition, whereas [Bar93] considers strong normalization which is intensionally different (and extensionally the same).

7.1.2 Calculating with Saturated Sets

It is always possible to produce a saturated set from any set M of terms by the *saturated closure* $\mathsf{cl}(M)$, defined by induction as follows:

- If $r \in M \cap \mathsf{SN}$ then $r \in \mathsf{cl}(M)$.
- If $\vec{r} \subset \mathsf{SN}$ then $x\vec{r} \in \mathsf{cl}(M)$.
- If $r[x := s]\vec{s} \in \mathsf{cl}(M)$ and $s \in \mathsf{SN}$ then $(\lambda x r)s\vec{s} \in \mathsf{cl}(M)$.

Since $\mathsf{cl}(M) \subseteq \mathsf{SN}$ (proved by induction on the definition), it is the least saturated set containing $M \cap \mathsf{SN}$.

Given a saturated set $\mathcal{M}$ and a saturated set $\mathcal{N}$, we want to construct a saturated set $\mathcal{M} \to \mathcal{N}$ which will later serve to define strong computability for function types. Define

$$\mathsf{S}_x(\mathcal{M},\mathcal{N}) := \{r \mid \forall s \in \mathcal{M}\, r[x := s] \in \mathcal{N}\},$$

$$\mathsf{I}(\mathcal{M},\mathcal{N}) := \{\lambda x r \mid x \in \mathcal{V} \text{ and } r \in \mathsf{S}_x(\mathcal{M},\mathcal{N})\} \quad \text{and}$$

$$\mathsf{E}(\mathcal{M},\mathcal{N}) := \{r \mid \forall s \in \mathcal{M}, rs \in \mathcal{N}\}.$$

We get the introduction-based definition $\mathcal{M} \to_{\mathrm{I}} \mathcal{N}$ and the elimination-based definition $\mathcal{M} \to_{\mathrm{E}} \mathcal{N}$ of saturated sets:

$$\mathcal{M} \to_{\mathrm{I}} \mathcal{N} := \mathsf{cl}(\mathsf{I}(\mathcal{M},\mathcal{N})) \text{ and } \mathcal{M} \to_{\mathrm{E}} \mathcal{N} := \mathsf{cl}(\mathsf{E}(\mathcal{M},\mathcal{N})).$$

Lemma 29 *$\mathsf{I}(\mathcal{M},\mathcal{N}) \subseteq \mathsf{SN}$, $\mathsf{E}(\mathcal{M},\mathcal{N}) \cap \mathsf{SN} \in \mathsf{SAT}$, and $\mathsf{I}(\mathcal{M},\mathcal{N}) \subseteq \mathsf{E}(\mathcal{M},\mathcal{N})$.*

Proof

(1) Let $r \in \mathsf{S}_x(\mathcal{M},\mathcal{N})$. Then for $s := x \in \mathcal{M}$, we get $r = r[x := s] \in \mathcal{N} \subseteq \mathsf{SN}$, hence also $\lambda x r \in \mathsf{SN}$.

(2) Check the conditions of saturatedness for $\mathsf{E}(\mathcal{M},\mathcal{N}) \cap \mathsf{SN}$:

1. Trivial.

2. Let $\vec{r} \subset \mathsf{SN}$ and $s \in \mathcal{M}$. Since $s \in \mathsf{SN}$ and $\mathcal{N} \in \mathsf{SAT}$, $x\vec{r}s \in \mathcal{N}$.

3. Simply append s and use saturatedness of $\mathcal{N}$.

(3) Let $r \in \mathsf{S}_x(\mathcal{M},\mathcal{N})$ and $s \in \mathcal{M}$. Show that $(\lambda xr)s \in \mathcal{N}$. Since $\mathcal{N} \in \mathsf{SAT}$, it suffices to show $r[x := s] \in \mathcal{N}$ and $s \in \mathsf{SN}$ which follow by definition of $\mathsf{S}_x(\mathcal{M},\mathcal{N})$ and $\mathcal{M} \subseteq \mathsf{SN}$. □

From the lemma, we get $\mathsf{I}(\mathcal{M},\mathcal{N}) \subseteq \mathcal{M} \to_{\mathrm{I}} \mathcal{N}$, $\mathcal{M} \to_{\mathrm{E}} \mathcal{N} = \mathsf{E}(\mathcal{M},\mathcal{N}) \cap \mathsf{SN}$ and, due to monotonicity of cl, $\mathcal{M} \to_{\mathrm{I}} \mathcal{N} \subseteq \mathcal{M} \to_{\mathrm{E}} \mathcal{N}$.

Define $\mathcal{M} \to \mathcal{N} := \mathcal{M} \to_{\mathrm{X}} \mathcal{N}$ with $\mathrm{X} \in \{\mathrm{I},\mathrm{E}\}$. We never use any property depending on this choice[8] but only the following three properties which are valid for both choices and follow immediately from the preceding remarks:

(SAT) $\mathcal{M} \to \mathcal{N} \in \mathsf{SAT}$.

($\to_{\mathrm{I}}$) If $r \in \mathsf{S}_x(\mathcal{M},\mathcal{N})$ then $\lambda xr \in \mathcal{M} \to \mathcal{N}$.

($\to_{\mathrm{E}}$) $r \in \mathcal{M} \to \mathcal{N} \wedge s \in \mathcal{M} \Rightarrow rs \in \mathcal{N}$.

7.1.3 Strong Computability

Definition 24 (Candidate assignment) *Any finite set of pairs (written $\alpha : \mathcal{M}$), consisting of a type variable and a saturated set, such that no type variable occurs twice.*

Candidate assigments are the counterpart to contexts. We will again use the letter Γ to denote a candidate assignment and write $\Gamma, \alpha : \mathcal{M}$ for the extended candidate assignment (with the implicit proviso that α does not occur in Γ).

Definition 25 (Strong computability) *Define the saturated set $\mathsf{SC}^\rho[\Gamma]$ of* strongly computable terms *w. r. t. type ρ and the candidate assigment Γ by recursion on ρ:*

- $\mathsf{SC}^\alpha[\Gamma] := \begin{cases} \mathcal{M} & \text{if } \alpha : \mathcal{M} \in \Gamma, \\ \mathsf{SN} & \text{otherwise.} \end{cases}$
- $\mathsf{SC}^{\rho\to\sigma}[\Gamma] := \mathsf{SC}^\rho[\Gamma] \to \mathsf{SC}^\sigma[\Gamma]$.
- $\mathsf{SC}^{\forall\alpha\rho}[\Gamma] := \bigcap_{\mathcal{M}\in\mathsf{SAT}} \mathsf{SC}^\rho[\Gamma, \alpha : \mathcal{M}]$ *(with set-theoretic intersection that clearly does not lead outside* SAT*; note that we may assume that α does not occur in Γ).*

The definition of $\mathsf{SC}^{\rho\to\sigma}[\Gamma]$ is a variant of the computability predicate definition in the famous [Tai67], its relativization to a candidate assignment and the big intersection in the definition of $\mathsf{SC}^{\forall\alpha\rho}[\Gamma]$ have been invented in [Gir72] and only today seem to be the straightforward extension of Tait's ideas. It has to be stressed that exactly this big intersection shows the impredicativity of

[8] The sets can indeed be different, e. g., for $\mathcal{M} := \mathsf{SN}$ and $\mathcal{N} := \mathsf{cl}(\{(\lambda xx)t \mid t \in \mathsf{SN}\})$.

the system of universal types: We need the intersection over any saturated set $\mathcal{M}$ in order to define a specific saturated set, namely $\mathsf{SC}^{\forall\alpha\rho}[\Gamma]$. This definition cannot be dealt with by second-order arithmetic, and, as remarked above, strong normalization also cannot be established by other means taken from second-order arithmetic.

Lemma 30 (Coincidence) *If $\alpha \notin FV(\rho)$ then $\mathsf{SC}^\rho[\Gamma, \alpha : \mathcal{M}] = \mathsf{SC}^\rho[\Gamma]$.*

Proof Induction on ρ. □

Lemma 31 (Substitution) $\mathsf{SC}^{\rho[\alpha:=\sigma]}[\Gamma] = \mathsf{SC}^\rho[\Gamma, \alpha : \mathsf{SC}^\sigma[\Gamma]]$.

Proof Induction on ρ, using the previous lemma. □

We want to show that every typable term is strongly normalizing. By using the inductive characterization, we only need to show that they are in SN. Since saturated sets are contained in SN, it suffices to show that $r \in \mathsf{SC}^\rho[\emptyset]$ whenever r gets type ρ. This is achieved by applying the following lemma to the identity substitution. Unfortunately, we first have to extend the notion of substitution $r[x := s]$ to the simultaneous substitution $r[\vec{x} := \vec{s}]$ of all occurrences of x_i by s_i (for every i, with different variables x_i) in r which may be defined by recursion on r like ordinary substitution. We will also use the notation $\vec{x} : \vec{\rho}$ for $x_1 : \rho_1, \ldots, x_n : \rho_n$ and $\vec{s} \in \mathsf{SC}^{\vec{\rho}}[\Gamma]$ for $s_1 \in \mathsf{SC}^{\rho_1}[\Gamma], \ldots, s_n \in \mathsf{SC}^{\rho_n}[\Gamma]$.

Lemma 32 *If $\vec{x} : \vec{\rho} \vdash r : \rho$ and $\vec{s} \in \mathsf{SC}^{\vec{\rho}}[\Gamma]$ then $r[\vec{x} := \vec{s}] \in \mathsf{SC}^\rho[\Gamma]$.*

Proof By induction on $\vec{x} : \vec{\rho} \vdash r : \rho$ simultaneously for every candidate assignment Γ.

(V) $r : \rho = x_i : \rho_i$. Obvious.

($\to_{\mathrm{I}}$) Let $\vec{x} : \vec{\rho} \vdash \lambda xr : \rho \to \sigma$ thanks to $\vec{x} : \vec{\rho}, x : \rho \vdash r : \sigma$. We have to show that $(\lambda xr)[\vec{x} := \vec{s}] \in \mathsf{SC}^\rho[\Gamma] \to \mathsf{SC}^\sigma[\Gamma]$. We may assume that $x \notin \vec{x} \cup \mathrm{FV}(\vec{s})$, and hence $(\lambda xr)[\vec{x} := \vec{s}] = \lambda x.r[\vec{x} := \vec{s}]$. It suffices to show that $r[\vec{x} := \vec{s}] \in \mathsf{S}_x(\mathsf{SC}^\rho[\Gamma], \mathsf{SC}^\sigma[\Gamma])$. So assume $s \in \mathsf{SC}^\rho[\Gamma]$ and show $r[\vec{x} := \vec{s}][x := s] \in \mathsf{SC}^\sigma[\Gamma]$. This follows from the induction hypothesis since $r[\vec{x} := \vec{s}][x := s] = r[\vec{x}, x := \vec{s}, s]$ by our assumption.

($\to_{\mathrm{E}}$) This is an immediate consequence of the induction hypothesis and the rule ($\to_{\mathrm{E}}$) for saturated sets.

($\forall_{\mathrm{I}}$) Let $\vec{x} : \vec{\rho} \vdash r : \forall\alpha\rho$ thanks to $\vec{x} : \vec{\rho} \vdash r : \rho$ and $\alpha \notin \mathrm{FV}(\vec{\rho})$. Let $\mathcal{M} \in \mathsf{SAT}$. We have to show that $r[\vec{x} := \vec{s}] \in \mathsf{SC}^\rho[\Gamma, \alpha : \mathcal{M}]$. Since $\alpha \notin \mathrm{FV}(\vec{\rho})$, we may apply the Coincidence Lemma and get $\vec{s} \in \mathsf{SC}^{\vec{\rho}}[\Gamma, \alpha : \mathcal{M}]$. Hence, we are done by the induction hypothesis.

($\forall_{\mathrm{E}}$) Let $\vec{x} : \vec{\rho} \vdash r : \rho[\alpha := \sigma]$ thanks to $\vec{x} : \vec{\rho} \vdash r : \forall\alpha\rho$. By induction hypothesis, $r[\vec{x} := \vec{s}] \in \mathsf{SC}^{\forall\alpha\rho}[\Gamma]$. Set $\mathcal{M} := \mathsf{SC}^\sigma[\Gamma]$. Then $r[\vec{x} := \vec{s}] \in \mathsf{SC}^\rho[\Gamma, \alpha : \mathcal{M}] = \mathsf{SC}^{\rho[\alpha:=\sigma]}[\Gamma]$ by the Substitution Lemma. □

By setting $s_i := x_i \in \mathsf{SC}^{\rho_i}[\emptyset]$ for $\Gamma = \vec{x} : \vec{\rho}$, and by using $\mathsf{SC}^\rho[\emptyset] \subseteq \mathsf{SN}$, we get the following

Theorem 3 (Strong Normalization) *If $\Gamma \vdash r : \rho$ then $r \in \mathsf{SN}$.* □

7.2 Undecidability of Type Checking

The problem of type checking is to find out whether $\Gamma \vdash r : \rho$ holds for given Γ, r and ρ. The problem of typability is to answer the question whether for given Γ and r, there is a type ρ such that $\Gamma \vdash r : \rho$. For both of these problems it was an open question whether they may be solved algorithmically [Bar93, p. 183]. It was generally believed that they are both undecidable. Nevertheless, the result was an achievement much applauded at the 1994 LICS[9] conference.

Theorem 4 ([**Wel94**]) *Type checking and typability are undecidable for the system of universal types.*

Proof See the 46 pages paper [Wel99] which rests on the undecidability of semi-unification. □

Therefore, in the sequel we will study a variant of the pure calculus with universal typing which we call system F. It has type information inside the term system which makes type checking decidable again.

7.3 An Explicit System of Parametric Polymorphism

8 Monotone Inductive Types

This section is based on [Mat99].

References

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, principles, techniques, and tools*. Addison-Wesley, 1986.

[Bar93] Henk P. Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and Tom S. E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1993.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. Thèse de Doctorat d'État, Université de Paris VII, 1972.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[Kri93] Jean-Louis Krivine. *Lambda-calculus, types and models*. Masson, Paris and Ellis Horwood, Hemel Hempstead, 1993. English translation of Lambda-calcul, types et modèles, Masson, 1990.

[9] Annual IEEE Symposium on Logic in Computer Science

[Mat98] Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. Doktorarbeit (PhD thesis), University of Munich, 1998. Available via the homepage `http://www.tcs.informatik.uni-muenchen.de/~matthes/`.

[Mat99] Ralph Matthes. Tarski's fixed-point theorem and lambda calculi with monotone inductive types. To appear in: Benedikt Löwe and Florian Rudolph, Foundations of the Formal Sciences, Refereed Papers of a Research Colloquium, Humboldt-Universität zu Berlin, May 7-9, 1999.

[Tai67] William W. Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tai75] William W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium Boston 1971/72*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer Verlag, 1975.

[Wel94] Joe B. Wells. Typability and type checking in the second-order lambda-calculus are equivalent and undecidable. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*, pages 176–185. IEEE Computer Society Press, 1994.

[Wel99] Joe B. Wells. Typability and type checking in system F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1–3):111–156, 1999.