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.

λxr. Let $\lambda xr \in \mathsf{SN}$ 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 $\hat{\Gamma} \vdash \lambda xr : \sigma \to \rho$.

$(\lambda xr)s\vec{s}$. Assume $(\lambda xr)s\vec{s} \in \mathsf{SN}$ 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, $\Delta \vdash (\lambda xr)s : \rho$ and by Lemma 21(a), $\Delta \vdash (\lambda xr)s\vec{s} : \tau$. □

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.[6] 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 λxx 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 λxx (in the empty context). And we may instantiate this for any type ρ instead of α to get $\vdash \lambda xx : \rho \to \rho$. 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 $(\rho \to \sigma) \in \mathcal{T}_u$.*

[6] In [Urz96] it is shown that $(\lambda f \lambda x.f(fx))(\lambda f \lambda x.f(fx))(\lambda x \lambda y.x)$ is not typable in the system of universal types presented below, although this term is strongly normalizing.

- *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 $\alpha \notin \{\beta\} \cup FV(\sigma)$.*

Lemma 29 *If $\alpha \notin FV(\rho)$ then $\rho[\alpha := \sigma] = \rho$.*

Proof Induction on ρ. □

Lemma 30 *$FV(\rho[\alpha := \sigma]) \subseteq FV(\forall\alpha\rho) \cup FV(\sigma)$.*

Proof Induction on ρ. □

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:*

$$\frac{}{\Gamma, x : \rho \vdash x : \rho}(V) \qquad \frac{\Gamma, x : \rho \vdash r : \sigma}{\Gamma \vdash \lambda xr : \rho \to \sigma}(\to_I) \qquad \frac{\Gamma \vdash r : \rho \to \sigma \qquad \Gamma \vdash s : \rho}{\Gamma \vdash rs : \sigma}(\to_E)$$

$$\frac{\Gamma \vdash r : \rho \qquad \alpha \notin FV(\Gamma)}{\Gamma \vdash r : \forall\alpha\rho}(\forall_I) \qquad \frac{\Gamma \vdash r : \forall\alpha\rho \qquad \sigma \in \mathcal{T}_u}{\Gamma \vdash r : \rho[\alpha := \sigma]}(\forall_E)$$

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 $(\forall_I)$ 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 $\vdash \lambda xx : \alpha \to \rho$ for any ρ which we never had in mind. The $\forall$-elimination rule $(\forall_E)$ 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

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

and hence $\vdash \lambda xx : \forall\beta.(\forall\alpha.\alpha \to \beta) \to (\forall\alpha.\alpha \to \beta)$. 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, Strengthening (see Lemma 20) and Lemma 21(a). Subject Reduction (see Lemma 23) also holds but requires a more intricate Inversion Lemma (see e.g. [Bar93, p. 174]).

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[7], 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* $(\lambda xr)s\vec{s} \in \mathcal{M}$.

This definition is nearly the same as that in [Bar93, p. 177].[8] Let SAT be the set of saturated sets. Trivially, $\mathsf{SN} \in \mathsf{SAT}$.

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

[8] 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).

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].)

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 $(\lambda xr)s\vec{s} \in \mathsf{cl}(M)$.

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

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

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

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

We get the introduction-based definition $\mathcal{M} \to_I \mathcal{N}$ and the elimination-based definition $\mathcal{M} \to_E \mathcal{N}$ of saturated sets:

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

Lemma 31 $\mathsf{I}(\mathcal{M},\mathcal{N}) \subseteq \mathsf{SN}$, $\mathsf{E}(\mathcal{M},\mathcal{N}) \cap \mathsf{SN} \in \mathsf{SAT}$, *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 $(\lambda x r)s \in \mathcal{N}$. 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[9] 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 $\lambda x r \in \mathcal{M} \to \mathcal{N}$.

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

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:*

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

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

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 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. Finally note that we could easily reprove strong normalization for the system of intersection types by setting $\mathsf{SC}^{\rho \cap \sigma} := \mathsf{SC}^\rho \cap \mathsf{SC}^\sigma$ and by abandoning the notion of candidate assignment altogether.

Lemma 32 (Coincidence) *If $\alpha \notin FV(\rho)$ then* $\mathsf{SC}^\rho[\Gamma, \alpha : \mathcal{M}] = \mathsf{SC}^\rho[\Gamma]$.

Proof Induction on ρ. □

Lemma 33 (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 34 *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 $\vec{x} : \vec{\rho} \vdash \lambda x r : \rho \to \sigma$ thanks to $\vec{x} : \vec{\rho}, x : \rho \vdash r : \sigma$. We have to show that $(\lambda x r)[\vec{x} := \vec{s}] \in \mathsf{SC}^\rho[\Gamma] \to \mathsf{SC}^\sigma[\Gamma]$. We may assume that $x \notin \vec{x} \cup \mathrm{FV}(\vec{s})$, and hence $(\lambda x r)[\vec{x} := \vec{s}] = \lambda x.r[\vec{x} := \vec{s}]$. 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.

($\forall_I$) Let $\vec{x}:\vec{\rho} \vdash r:\forall\alpha\rho$ thanks to $\vec{x}:\vec{\rho} \vdash r:\rho$ and $\alpha \notin FV(\vec{\rho})$. 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 $\alpha \notin FV(\vec{\rho})$, 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.

($\forall_E$) 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[10] 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

This time, we do not alter the type system but the term system.

Definition 26 (Terms of system F) *The set* $\mathcal{T}_{\mathsf{F}}$ *of terms is inductively given by:*

- *If* $x \in \mathcal{V}$ *then* $x \in \mathcal{T}_{\mathsf{F}}$.
- *If* $x \in \mathcal{V}$, $\rho \in \mathcal{T}_u$ *and* $r \in \mathcal{T}_{\mathsf{F}}$ *then* $\lambda x^{\rho} r \in \mathcal{T}_{\mathsf{F}}$.
- *If* $r \in \mathcal{T}_{\mathsf{F}}$ *and* $s \in \mathcal{T}_{\mathsf{F}}$ *then* $(rs) \in \mathcal{T}_{\mathsf{F}}$.
- *If* $r \in \mathcal{T}_{\mathsf{F}}$ *and* $\alpha \in \mathcal{V}_T$ *then* $\Lambda\alpha r \in \mathcal{T}_{\mathsf{F}}$.
- *If* $r \in \mathcal{T}_{\mathsf{F}}$ *and* $\sigma \in \mathcal{T}_u$ *then* $(r\sigma) \in \mathcal{T}_{\mathsf{F}}$.

[10] Annual IEEE Symposium on Logic in Computer Science

The idea is to add type information to the terms. $\lambda x^{\rho} r$ is $\lambda x r$ but with an indication which was the type in the extended context for the typing of r (see the typing rules below). $\Lambda\alpha r$ is r but with its polymorphism in the parameter α made explicit. $(r\sigma)$ is r but with a declaration that it is used with type σ.

Again parentheses are omitted as much as possible (with applications associating to the left).

Definition 27 (Free variables) *Define the set* $FV(r)$ *of variables occurring free in* r *by recursion on* r*:*

- $FV(x) := \{x\}$.
- $FV(\lambda x^{\rho} r) := FV(r) \setminus \{x\}$.
- $FV(rs) := FV(r) \cup FV(s)$.
- $FV(\Lambda\alpha r) := FV(r)$.
- $FV(r\sigma) := FV(r)$.

As before, λx^{ρ} binds the free occurrences of x in r, and we syntactically identify terms which only differ in the names of their bound variables.

Since types may form parts of a term, we now have an additional concept of free type variables of a term:

Definition 28 (Free type variables of a term) *Define the set* $FTV(r)$ *of type variables occurring free in* r *by recursion on* r*:*

- $FTV(x) := \emptyset$.
- $FTV(\lambda x^{\rho} r) := FV(\rho) \cup FTV(r)$.
- $FTV(rs) := FTV(r) \cup FTV(s)$.
- $FTV(\Lambda\alpha r) := FTV(r) \setminus \{\alpha\}$.
- $FTV(r\sigma) := FTV(r) \cup FV(\sigma)$.

Clearly, $\Lambda\alpha$ binds the free occurrences of α in r. We also identify terms which differ only in the names of their bound type variables.

It is straightforward to redefine substitution of terms for term variables in terms:

Definition 29 (Substitution) *Define the result* $r[x:=s]$ *of replacing every free occurrence of the variable* x *in* r *by the term* s *recursively as follows:*

- $x[x:=s] := s$
- $y[x:=s] := y$ *for* $y \neq x$.
- $(\lambda y^{\rho} r)[x:=s] := \lambda y^{\rho}.r[x:=s]$ *where we may assume as usual that* $y \notin \{x\} \cup FV(s)$.

- $(rt)[x := s] := r[x := s]t[x := s]$.
- $(\Lambda\alpha r)[x := s] := \Lambda\alpha.r[x := s]$ *where we assume that* $\alpha \notin FTV(s)$.
- $(r\sigma)[x := s] := r[x := s]\sigma$.

But we also have substitution of types for type variables in terms:

Definition 30 (Type substitution) *Define the result* $r[\alpha := \sigma]$ *of replacing every free occurrence of the type variable* α *in* r *by the type* σ *recursively as follows:*

- $x[\alpha := \sigma] := x$.
- $(\lambda y^\rho r)[\alpha := \sigma] := \lambda y^{\rho[\alpha:=\sigma]}.r[\alpha := \sigma]$.
- $(rt)[\alpha := \sigma] := r[\alpha := \sigma]t[\alpha := \sigma]$.
- $(\Lambda\beta r)[\alpha := \sigma] := \Lambda\beta.r[\alpha := \sigma]$ *where we assume that* $\beta \notin \{\alpha\} \cup FV(\sigma)$.
- $(r\tau)[\alpha := \sigma] := r[\alpha := \sigma]\tau[\alpha := \sigma]$.

Lemma 35 *If* $x \notin FV(r)$ *then* $r[x := s] = r$. *If* $\alpha \notin FTV(r)$ *then* $r[\alpha := \sigma] = r$.

Proof Induction on r. □

The richer term syntax allows a new reduction in the spirit of β-reduction, namely $(\Lambda\alpha r)\sigma \to r[\alpha := \sigma]$ which perfectly fits with our intuition of Λ-abstraction and type application. For coding purposes, this is yet not enough (as will be clear later). Therefore, we also include the following η-reduction rules: $\lambda x^\rho.rx \to r$ if $x \notin \mathrm{FV}(r)$, and $\Lambda\alpha.r\alpha \to r$ if $\alpha \notin \mathrm{FTV}(r)$. Clearly, we could have added the first rule (without the type superscript) to the pure untyped λ-calculus. Unfortunately, the addition of $\lambda x.rx \to_\beta r$ for $x \notin \mathrm{FV}(r)$ would have destroyed Subject Reduction: Take three different type variables α, β, γ and apply rule $(\to_I)$ to $z : \alpha \to \forall\gamma\beta, x : \alpha \vdash zx : \beta$ or to $z : \alpha \to \beta, x : \alpha \vdash zx : \forall\gamma\beta$. Note that neither $z : \alpha \to \forall\gamma\beta \vdash z : \alpha \to \beta$ nor $z : \alpha \to \beta \vdash z : \alpha \to \forall\gamma\beta$ holds.

Definition 31 ($\beta\eta$-reduction) *Inductively define the relation* $\to_{\beta\eta}$ *as follows:*

(β) $(\lambda x^\rho r)s \to_{\beta\eta} r[x := s]$ *(outer β-reduction).*

(η) $\lambda x^\rho.rx \to_{\beta\eta} r$ *if* $x \notin FV(r)$ *(outer η-reduction).*

(β_F) $(\Lambda\alpha r)\sigma \to_{\beta\eta} r[\alpha := \sigma]$ *(outer type-β-reduction).*

(η_F) $\Lambda\alpha.r\alpha \to_{\beta\eta} r$ *if* $\alpha \notin FTV(r)$ *(outer type-η-reduction).*

(ξ) $r \to_{\beta\eta} r' \Rightarrow \lambda x^\rho r \to_{\beta\eta} \lambda x^\rho r'$ *(reduction under a λ-abstraction).*

(ξ_F) $r \to_{\beta\eta} r' \Rightarrow \Lambda\alpha r \to_{\beta\eta} \Lambda\alpha r'$ *(reduction under a Λ-abstraction).*

(r) $r \to_{\beta\eta} r' \Rightarrow rs \to_{\beta\eta} r's$ *(right application).*

(l) $r \to_{\beta\eta} r' \Rightarrow sr \to_{\beta\eta} sr'$ *(left application).*

(t) $r \to_{\beta\eta} r' \Rightarrow r\sigma \to_{\beta\eta} r'\sigma$ *(type application).*

If $r \to_{\beta\eta} s$ we say that r reduces by one $\beta\eta$*-reduction step* to s.

Lemma 36 *If* $r \to_{\beta\eta} r'$ *then* $FV(r') \subseteq FV(r)$ *and* $FTV(r') \subseteq FTV(r)$.

Proof Induction on $\to_{\beta\eta}$. For (β), one first has to prove $\mathrm{FV}(r[x := s]) \subseteq (\mathrm{FV}(r) \setminus \{x\}) \cup \mathrm{FV}(s)$ and $\mathrm{FTV}(r[x := s]) \subseteq \mathrm{FTV}(r) \cup \mathrm{FTV}(s)$. For (β_F), we need $\mathrm{FV}(r[\alpha := \sigma]) = \mathrm{FV}(r)$ and $\mathrm{FTV}(r[\alpha := \sigma]) \subseteq (\mathrm{FTV}(r) \setminus \{\alpha\}) \cup \mathrm{FV}(\sigma)$. □

Lemma 6 and Lemma 7 also hold for the extended syntax. Moreover, we get substitutivity w. r. t. type substitution:

Lemma 37 (Substitutivity and Compatibility)

If $r \to_{\beta\eta} r'$ *then* $r[x := s] \to_{\beta\eta} r'[x := s]$ *and* $r[\alpha := \sigma] \to_{\beta\eta} r'[\alpha := \sigma]$.

If $s \to_{\beta\eta} s'$ *then* $r[x := s] \to^*_{\beta\eta} r[x := s']$, *and* $r[x := s] \to_{\beta\eta} r[x := s']$ *if* x *occurs exactly once free in* r.

Proof Substitutivity is proved by induction on $r \to_{\beta\eta} r'$, compatibility is proved by induction on r. □

Unfortunately, $\to_{\beta\eta}$ is *not* locally confluent: Consider

$$\begin{tikzcd}
 & \lambda x^\rho.(\lambda y^\sigma r)x \arrow[dl, "\beta\eta"'] \arrow[dr, "\beta\eta"] & \\
\lambda x^\rho.r[y := x] \arrow[d, equal] & & \lambda y^\sigma r \\
\lambda y^\rho r & &
\end{tikzcd}$$

with $\rho \neq \sigma$ and $x \notin \mathrm{FV}(\lambda y r)$. This problem will be overcome by typing.

Definition 32 (Typing for system F) *The relation* $\Gamma \vdash r : \rho$ *(term* r *has type* ρ *in context* Γ*) is inductively defined by the following rules:*

$$\frac{}{\Gamma, x : \rho \vdash x : \rho}(V) \qquad \frac{\Gamma, x : \rho \vdash r : \sigma}{\Gamma \vdash \lambda x^\rho r : \rho \to \sigma}(\to_I) \qquad \frac{\Gamma \vdash r : \rho \to \sigma \qquad \Gamma \vdash s : \rho}{\Gamma \vdash rs : \sigma}(\to_E)$$

$$\frac{\Gamma \vdash r : \rho \qquad \alpha \notin FV(\Gamma)}{\Gamma \vdash \Lambda\alpha r : \forall\alpha\rho}(\forall_I) \qquad \frac{\Gamma \vdash r : \forall\alpha\rho \qquad \sigma \in \mathcal{T}_u}{\Gamma \vdash r\sigma : \rho[\alpha := \sigma]}(\forall_E)$$

The usual conventions concerning contexts apply.

Note how we restored the property that we always know which typing rule has been applied last (like with simple typing and unlike intersection typing or even universal typing).

Lemma 38 *If* $\Gamma \vdash r : \rho$ *then* $FV(\rho) \subseteq FV(\Gamma) \cup FTV(r)$.

Proof Induction on $\vdash$. Note that in case $(\to_I)$ we need that the information on the type of the abstracted variable is included in the syntax: If $\alpha \in \mathrm{FV}(\rho \to \sigma)$ then $\alpha \in \mathrm{FV}(\rho)$ or $\alpha \in \mathrm{FV}(\sigma)$. In the first case, $\alpha \in \mathrm{FTV}(\lambda x^\rho r)$. In the second case, the induction hypothesis gives that $\alpha \in \mathrm{FV}(\Gamma) \cup \mathrm{FV}(\rho) \cup \mathrm{FTV}(r) = \mathrm{FV}(\Gamma) \cup \mathrm{FTV}(\lambda x^\rho r)$. The case $(\forall_E)$ uses Lemma 30. $\square$

Lemma 39 *Every typable term, i. e., every term* r *such that there are* Γ *and* ρ *with* $\Gamma \vdash r : \rho$, *has exactly one of the following forms:*

$$x\vec{S} \qquad \lambda x^\rho r \qquad (\lambda x^\rho r)s\vec{S} \qquad \Lambda\alpha r \qquad (\Lambda\alpha r)\sigma\vec{S}$$

where $\vec{S}$ *shall denote a finite list of* terms and types.

Proof It is clear that *every* term has exactly one of the following forms:

$$x\vec{S} \qquad \lambda x^\rho r \qquad (\lambda x^\rho r)s\vec{S} \qquad (\lambda x^\rho r)\sigma\vec{S} \qquad \Lambda\alpha r \qquad (\Lambda\alpha r)s\vec{S} \qquad (\Lambda\alpha r)\sigma\vec{S}.$$

Typability rules out the fourth and sixth possibility (note that the typability of $r\vec{S}$ implies that of r). $\square$

Once again have Exchange 1 (p. 23) as part of our understanding of contexts, and it is again quite easy to establish Weakening 1, Strengthening (see Lemma 20) and Lemma 21(a). Moreover, we have a version of Lemma 21(a) pertaining to type substitution:

Lemma 40 *If* $\Gamma \vdash r : \rho$ *then* $\Gamma[\alpha := \sigma] \vdash r[\alpha := \sigma] : \rho[\alpha := \sigma]$.

Proof $\Gamma[\alpha := \sigma]$ clearly denotes the context where all the types of the variables are substituted. The proof is by induction on $\vdash$. $\square$

Lemma 41 (Subject reduction) *If* $r \to_{\beta\eta} r'$ *and* $\Gamma \vdash r : \rho$ *then* $\Gamma \vdash r' : \rho$.

Proof Induction on $\to_{\beta\eta}$. We only consider the initial cases.

(β) Let $\Gamma \vdash (\lambda x^\rho r)s : \sigma$. Then $\Gamma \vdash \lambda x^\rho r : \rho \to \sigma$ and $\Gamma \vdash s : \rho$. Hence $\Gamma, x : \rho \vdash r : \sigma$, and by the analogue of Lemma 21(a), $\Gamma \vdash r[x := s] : \sigma$.

(η) Let $\Gamma \vdash \lambda x^\rho.rx : \rho \to \sigma$. Hence, $\Gamma, x : \rho \vdash rx : \sigma$, and therefore, $\Gamma, x : \rho \vdash r : \rho \to \sigma$. By Strengthening, $\Gamma \vdash r : \rho \to \sigma$.

(β_F) Let $\Gamma \vdash (\Lambda\alpha r)\sigma : \tau$. Then $\Gamma \vdash \Lambda\alpha r : \forall\alpha\rho$ and $\rho[\alpha := \sigma] = \tau$. Consequently, $\Gamma \vdash r : \rho$ and $\alpha \notin \mathrm{FV}(\Gamma)$. Hence $\Gamma[\alpha := \sigma] = \Gamma$ and by the preceding lemma $\Gamma \vdash r[\alpha := \sigma] : \rho[\alpha := \sigma] = \tau$.

(η_F) Let $\Gamma \vdash \Lambda\alpha.r\alpha : \forall\alpha\rho$ with $\alpha \notin \mathrm{FTV}(r)$. Then $\Gamma \vdash r\alpha : \rho$ and $\alpha \notin \mathrm{FV}(\Gamma)$. Hence, $\Gamma \vdash r : \forall\beta\sigma$ and $\sigma[\beta := \alpha] = \rho$. If $\alpha = \beta$, then we are done. Otherwise, by Lemma 38, $\alpha \notin \mathrm{FV}(\sigma)$, hence $\forall\alpha\rho = \forall\beta\sigma$ by renaming of the bound variable. $\square$

Note that the examples on page 37 are no longer critical: The first one yields $z : \alpha \to \forall\gamma\beta \vdash \lambda x^\alpha.zx\gamma : \alpha \to \beta$, the second $z : \alpha \to \beta \vdash \lambda x^\alpha \Lambda\gamma.zx : \alpha \to \forall\gamma\beta$. In both cases, we cannot apply an outer η-reduction.

Before studying confluence and strong normalization of the typable terms, we consider several examples showing the expressivity of system F.

Examples 10 *1. Set* $0 := \forall\alpha\alpha$. *Then* $\Gamma \vdash r : 0$ *implies* $\Gamma \vdash r\rho : \rho$.

2. *Set* $1 := \forall\alpha.\alpha \to \alpha$ *and* $\mathsf{IN1} := \Lambda\alpha\lambda x^\alpha x$. *Then* $\vdash \mathsf{IN1} : 1$.

3. *Set* $\rho \times \sigma := \forall\alpha.(\rho \to \sigma \to \alpha) \to \alpha$ *for some* $\alpha \notin FV(\rho) \cup FV(\sigma)$. *Set* $\langle r, s\rangle_{\rho,\sigma} := \Lambda\alpha\lambda z^{\rho\to\sigma\to\alpha}.zrs$ *for some* $z \notin FV(r) \cup FV(s)$ *and assume that* $\alpha \notin FTV(r) \cup FTV(s)$. *Hence, if* $\Gamma \vdash r : \rho$ *and* $\Gamma \vdash s : \sigma$ *then* $\Gamma \vdash \langle r, s\rangle_{\rho,\sigma} : \rho \times \sigma$. *Set* $r\mathsf{L}_{\rho,\sigma} := r\rho(\lambda x^\rho \lambda y^\sigma.x)$ *and* $r\mathsf{R}_{\rho,\sigma} := r\sigma(\lambda x^\rho \lambda y^\sigma.y)$. *Hence, if* $\Gamma \vdash r : \rho \times \sigma$ *then* $\Gamma \vdash r\mathsf{L}_{\rho,\sigma} : \rho$ *and* $\Gamma \vdash r\mathsf{R}_{\rho,\sigma} : \sigma$. *Moreover,* $\langle r, s\rangle_{\rho,\sigma}\mathsf{L}_{\rho,\sigma} \to^*_{\beta\eta} r$ *and* $\langle r, s\rangle_{\rho,\sigma}\mathsf{R}_{\rho,\sigma} \to^*_{\beta\eta} s$.

4. *Set* $\rho + \sigma := \forall\alpha.(\rho \to \alpha) \to (\sigma \to \alpha) \to \alpha$ *for some* $\alpha \notin FV(\rho) \cup FV(\sigma)$. *Set* $\mathsf{INL}_{\rho,\sigma}r := \Lambda\alpha\lambda x^{\rho\to\alpha}\lambda y^{\sigma\to\alpha}.xr$ *and* $\mathsf{INR}_{\rho,\sigma}r := \Lambda\alpha\lambda x^{\rho\to\alpha}\lambda y^{\sigma\to\alpha}.yr$ *for some* $x, y \notin FV(r)$ *(we assume that* $\alpha \notin FTV(r)$*). Hence, if* $\Gamma \vdash r : \rho$ *then* $\Gamma \vdash \mathsf{INL}_{\rho,\sigma}r : \rho + \sigma$, *and if* $\Gamma \vdash r : \sigma$ *then* $\Gamma \vdash \mathsf{INR}_{\rho,\sigma}r : \rho + \sigma$. *Moreover, if* $\Gamma \vdash r : \rho + \sigma$, $\Gamma \vdash s : \rho \to \tau$ *and* $\Gamma \vdash t : \sigma \to \tau$ *then* $\Gamma \vdash r\tau st : \tau$ *which gives a construct for case distinction as follows:* $\mathsf{INL}_{\rho,\sigma}r\tau st \to^*_{\beta\eta} sr$ *and* $\mathsf{INR}_{\rho,\sigma}r\tau st \to^*_{\beta\eta} tr$.

5. *Set* $\exists\alpha\rho := \forall\beta.(\forall\alpha.\rho \to \beta) \to \beta$ *for some* $\beta \notin \{\alpha\} \cup FV(\rho)$. *Set* $\mathsf{C}_{\exists\alpha\rho,\tau}r := \Lambda\beta\lambda x^{\forall\alpha.\rho\to\beta}.x\tau r$ *for some* $x \notin FV(r)$ *and* $\beta \notin FTV(r) \cup FV(\tau)$. *Hence, if* $\Gamma \vdash r : \rho[\alpha := \tau]$ *then* $\Gamma \vdash \mathsf{C}_{\exists\alpha\rho,\tau}r : \exists\alpha\rho$. *Also, if* $\Gamma \vdash r : \exists\alpha\rho$ *and* $\Gamma \vdash s : \forall\alpha.\rho \to \sigma$ *with* $\alpha \notin FV(\sigma)$, *then* $\Gamma \vdash r\sigma s : \sigma$. *Moreover,* $\mathsf{C}_{\exists\alpha\rho,\tau}r\sigma s \to^*_{\beta\eta} s\tau r$.

6. *Set* $\mathsf{nat} := \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha$. *Set* $0 := \Lambda\alpha\lambda x^{\alpha\to\alpha}\lambda y^\alpha y$ *(this should not be confused with the type* $0 = \forall\alpha\alpha$*). Then* $\vdash 0 : \mathsf{nat}$. *Set* $\mathsf{S}r := \Lambda\alpha\lambda x^{\alpha\to\alpha}\lambda y^\alpha.x(r\alpha xy)$ *for some* $x, y \notin FV(r)$ *and* $\alpha \notin FTV(r)$. *Hence, if* $\Gamma \vdash r : \mathsf{nat}$ *then* $\Gamma \vdash \mathsf{S}r : \mathsf{nat}$. *This gives back iteration on naturals as follows: If* $\Gamma \vdash r : \mathsf{nat}$, $\Gamma \vdash s : \rho \to \rho$ *and* $\Gamma \vdash t : \rho$ *then* $\Gamma \vdash r\rho st : \rho$. *Moreover,* $0\rho st \to^*_{\beta\eta} t$ *and* $(\mathsf{S}r)\rho st \to^*_{\beta\eta} s(r\rho st)$, *hence* t *is the initial term of the iteration, and* s *is the step term.*

Lemma 42 (Typed local confluence) *If* $\Gamma \vdash r : \rho$, $r \to_{\beta\eta} r'$ *and* $r \to_{\beta\eta} r''$ *then there is a term* t *such that* $r' \to_{\beta\eta} t$ *and* $r'' \to_{\beta\eta} t$.

Proof Induction on r, case distinction according to the last rule of $\to_{\beta\eta}$ used to establish $r \to_{\beta\eta} r'$ and $r \to_{\beta\eta} r''$. Hence, we have to distinguish 81 cases. *** $\square$

In order to prove that even typed confluence holds, we first establish strong normalization.
***

Lemma 43 (Typed confluence) *If* $\Gamma \vdash r : \rho$, $r \to^*_{\beta\eta} r'$ *and* $r \to^*_{\beta\eta} r''$ *then there is a term* t *such that* $r' \to_{\beta\eta} t$ *and* $r'' \to_{\beta\eta} t$.

Proof Induction on $r \in \mathsf{sn}$.[11] If $r = r'$ or $r = r''$ then the claim is trivial (set $t = r''$ or $t = r'$, respectively). Otherwise, $r \to_{\beta\eta} r'_0 \to^*_{\beta\eta} r'$ and $r \to_{\beta\eta} r''_0 \to^*_{\beta\eta} r''$ for some terms r'_0 and r''_0. By typed local confluence, there is a term s such that $r'_0 \to^*_{\beta\eta} s$ and $r''_0 \to^*_{\beta\eta} s$. By Subject Reduction, $\Gamma \vdash r'_0 : \rho$ and $\Gamma \vdash r''_0 : \rho$. Hence, by induction hypothesis for r'_0, there is s' such that $r' \to^*_{\beta\eta} s'$ and $s \to^*_{\beta\eta} s'$, hence $r''_0 \to^*_{\beta\eta} s'$. We now apply the induction hypothesis to r''_0 and get the term t with and $r'' \to^*_{\beta\eta} t$ and $s' \to^*_{\beta\eta} t$, hence also $r' \to^*_{\beta\eta} t$. □

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.

[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.

[11] This proof is in essence the proof of Newman's Lemma saying that a locally confluent and strongly normalizing binary relation is confluent.

[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.

[Urz96] Paweł Urzyczyn. Positive recursive type assignment. *Fundamenta Informaticae*, 28(1–2):197–209, 1996.

[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.