An algebraic view on recursive types

Michael Marz[*]

Fachbereich Mathematik
Technische Hochschule Darmstadt, Germany

May 29, 1996

This is a revised version of Preprint-Nr. 1793 from November 1995.

Introduction

Relatively few λ-terms can be assigned a type in the simple type hierarchy. As is well known, more terms can be typed if recursive types are admitted [Gun92]. In this note a similar extension to simple types is examined.

Our approach is essentially algebraic. We start from the observation that the set of simple types is a totally free algebra with respect to the binary operation $\rightarrow$, i.e. a *free groupoid.* More general groupoids arise as factor structures of free groupoids, so we are lead to discuss free groupoids with a congruence relation. We call these *type systems.*

Not every type system behaves well with respect to reduction; the crucial subject reduction property may fail. We give a necessary and sufficient condition on type systems, called *condition (K)*, which guarantees subject reduction.

Restricting to finitely many atomic types and finitely generated congruences we can effectively *present* a type system. Our first result says that it is decidable whether a such presented type system is a K-*type system*, i.e. whether it satisfies condition (K). Next we show that finitely presented K-type system can always be presented in a particular fashion, namely, by specifying a map from atomic types to type expressions. In other words, the congruence for finitely presented K-type systems is always generated by a list of constraints $\iota_1 \sim \tau_1, \iota_2 \sim \tau_2, \ldots, \iota_n \sim \tau_n$ where $\iota_1, \ldots, \iota_n$ are atomic.

The next question is whether λ-terms typeable in a K-type system are strongly normalizing. This is not always the case. N. MENDLER shows in [Men91] that it is precisely the so called *positive* type systems for which strong normalization holds. In Section 2 we show how to read off positivity from a presentation. In particular, it is decidable whether a finitely presented type system is positive.

[*]e-mail: `marz@mathematik.th-darmstadt.de`

We also show that *principal typing* is possible using K-type systems. This entails that the question whether a given term admits a type in some positive K-type system is decidable. Since the set of strongly normalizing λ-terms is not recursive, it follows that neither a single nor the union of all positive K-type systems will capture strong normalization.

This paper is a translated extract of my diploma thesis, see [Mar95]. I would like to thank Paweł Urzyczyn for some useful hints, Martin Hofmann, Mathias Kegelmann and Hermann Puhlmann for their careful proof reading, and especially Achim Jung for his stimulating supervision.

1 Free groupoids as type systems for the λ-calculus

There are two possible ways of thinking about type systems of the λ-calculus. In *Church*-style each λ-term contains type annotations in λ-abstractions which permit type reconstruction. On the other hand, in *Curry*-style the types appear as predicates over the set of untyped terms. We are interested in *Curry*-style.

In this section we consider a general approach to monomorphic type systems, i.e. type systems consisting only of atomic types and function types. In the simply typed λ-calculus the set of all types forms a groupoid freely generated by the atomic types, the groupoid operation being the arrow $\rightarrow$. To allow for more terms to be typed, one approach is to consider *type constraints*. These are sets of equations of the form $\iota \sim \tau$ where ι must be atomic and τ can be any type. From an algebraic point of view this leads to the study of free groupoids with arbitrary congruences as type systems. For our purposes it is more suitable to explicitly handle congruences instead of working with quotient groupoids.

Definition 1.1 *A* type system $(\underline{\mathcal{F}}, \approx)$ *consists of a free groupoid* $\underline{\mathcal{F}} = (\mathcal{F}, \rightarrow)$*, i.e. an algebra with one binary operation, and a congruence relation* $\approx$*.*

In the following X denotes a countably infinite set of variables and $(\underline{\mathcal{F}}, \approx)$ stands for a type system.

Definition 1.2 *A* basis *is a finite set* Γ *of expressions* $(x : \tau)$ *with* $x \in X$ *and* $\tau \in \mathcal{F}$ *such that for each* $x \in X$ *there is at most one* $\tau \in \mathcal{F}$ *with* $(x : \tau) \in \Gamma$*. We use* $\Gamma, x : \tau$ *as an abbreviation for* $\Gamma \cup \{(x : \tau)\}$ *if* x *does not occur in* Γ*. An expression* $\Gamma \vdash t : \tau$ *is called a* judgement *if it is derivable with the following rules:*

$$\frac{}{\Gamma \vdash x : \tau}\ \textit{(Var)} \quad \textit{if } (x : \tau) \in \Gamma \qquad\qquad \frac{\Gamma \vdash s : \tau \rightarrow \sigma \quad \Gamma \vdash t : \tau}{\Gamma \vdash st : \sigma}\ \textit{(App)}$$

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \rightarrow \tau}\ \textit{(Abs)} \qquad\qquad \frac{\Gamma \vdash t : \sigma}{\Gamma \vdash t : \tau}\ \textit{(Con)} \quad \textit{if } \sigma \approx \tau$$

Most of the main syntactic properties of the simply typed λ-calculus remain true for type systems as presented here. Notably, we have the following two lemmas which can be proved by induction:

Lemma 1.3 (Generation Lemma) *Let Γ be a basis such that $\Gamma \vdash t : \tau$ is a judgement.*

a) If $t = x \in X$, then $(x : \tau_1) \in \Gamma$ for some τ_1 with $\tau_1 \approx \tau$.

b) If $t = t_1 t_2$, then there is a $\tau_1 \in \mathcal{F}$ such that $\Gamma \vdash t_1 : \tau_1 \to \tau$ and $\Gamma \vdash t_2 : \tau_1$ are judgements.

c) If $t = \lambda x.t_1$, then there are $\tau_1, \tau_2 \in \mathcal{F}$ such that $\tau \approx \tau_2 \to \tau_1$ and $\Gamma, x : \tau_2 \vdash t_1 : \tau_1$ is a judgement.

Lemma 1.4 (Substitution Lemma) *Let Γ be a basis such that $\Gamma, x : \sigma \vdash t : \tau$ and $\Gamma \vdash s : \sigma$ are judgements for λ-terms s and t. Then $\Gamma \vdash t[s/x] : \tau$ is also a judgement.*

The β-reduction is defined as usual (see e.g. [Bar92]), we write $s \xrightarrow[\beta]{} \!\!\!\!\rightarrow t$ if s reduces to t. Unfortunately, the judgements are not always stable under β-reduction.

Definition 1.5 *Let $(\underline{\mathcal{F}}, \approx)$ be a type system. The $\lambda_{(\underline{\mathcal{F}},\approx)}$-calculus has the β-*subject reduction property *if, for any judgement $\Gamma \vdash s : \tau$ and for any reduction $s \xrightarrow[\beta]{} \!\!\!\!\rightarrow t$, the expression $\Gamma \vdash t : \tau$ is a judgement.*

Proposition 1.6 *The following are equivalent:*

(1) The $\lambda_{(\underline{\mathcal{F}},\approx)}$-calculus has the β-subject reduction property.

(2) The following property holds for every $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}$:

$$(K) \qquad \sigma_1 \to \tau_1 \approx \sigma_2 \to \tau_2 \quad \Rightarrow \quad \sigma_1 \approx \sigma_2 \text{ and } \tau_1 \approx \tau_2$$

Proof: (2) implies (1): Straightforward using the Generation Lemma and the Substitution Lemma.

(1) implies (2): First, we show that $\sigma_1 \to \tau_1 \approx \sigma_2 \to \tau_2$ implies $\tau_1 \approx \tau_2$ (for $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}$). For $x, y, z \in X$ there is a deduction for $y : \tau_1, z : \sigma_2 \vdash (\lambda x.y)z : \tau_2$. Since we have the β-subject reduction property we get $y : \tau_1, z : \sigma_2 \vdash y : \tau_2$. Hence, $\tau_1 \approx \tau_2$ follows from the Generation Lemma.

For the second part, we take again $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}$ with $\sigma_1 \to \tau_1 \approx \sigma_2 \to \tau_2$. Choosing $\Gamma = \{u : \sigma_1 \to \tau_1, v : \sigma_1 \to \sigma_1, y : \sigma_2\}$ we can derive $\Gamma \vdash (\lambda x.u(vx))y : \tau_2$. Since we assume the subject reduction property, $\Gamma \vdash u(vy) : \tau_2$ is also a judgement. Because of the Generation Lemma there is a $\psi \in \mathcal{F}$ with $\Gamma \vdash u : \psi \to \tau_2$ and $\Gamma \vdash vy : \psi$. Again we apply the Generation Lemma, hence, we have $\sigma_1 \to \sigma_1 \approx \sigma_2 \to \psi$.

As shown in the first part this leads to $\sigma_1 \approx \psi$, thus $\sigma_1 \to \sigma_1 \approx \sigma_2 \to \sigma_1$ follows. Because $y : \sigma_2 \vdash (\lambda x.x)y : \sigma_1$ is a judgement we get $y : \sigma_2 \vdash y : \sigma_1$ by the subject reduction property. This leads to $\sigma_1 \approx \sigma_2$. □

We have shown that (K) is a sufficient and necessary condition for the β-subject reduction property. The idea of this proof is due to R. Statman, see [Sta94], however, the proof as presented here is much shorter.

If we add *η-reduction* to the system we do not get any problems: the calculus is stable under η-reduction.

Type systems that satisfy (K) are called *K-type systems.* There are many properties of the untyped λ-calculus that remain true for the $\lambda_{(\underline{\mathcal{F}},\approx)}$-calculus. In case the calculus has a K-type system, we get the *Church-Rosser-Theorem* as a consequence of the subject reduction property. There are no non-trivial *finite* K-type systems:

Remark 1.7 *The number of congruence classes of a K-type system is 1 or infinite.*

Proof: Let $(\{\tau_1, \ldots, \tau_n\}, \to)$ be a groupoid with $n > 1$. Since there are $n^2 > n$ possibilities to combine elements of $\mathcal{T}$ with $\to$, there exist $i_0, i_1, i_2, j_1, j_2 \in \{1, \ldots, n\}$ such that $\tau_{i_0} = \tau_{i_1} \to \tau_{i_2} = \tau_{j_1} \to \tau_{j_2}$ with $i_1 \neq j_1$ or $i_2 \neq j_2$. In this case the free groupoid $\underline{\mathcal{F}_\mathcal{T}}$ with the corresponding congruence relation is not a K-type system. □

The type system with the congruence relation $\mathcal{F} \times \mathcal{F}$ induces the untyped λ-calculus. Using the free groupoid $\underline{\mathcal{F}_\mathcal{J}}$ with the smallest congruence relation we get the simply typed λ-calculus with $\mathcal{J}$ as the set of atomic types.

For a decidability result for K-type systems we need two lemmas:

Lemma 1.8 *Let $(\underline{\mathcal{F}_\mathcal{J}}, \approx)$ be a type system that is generated by a symmetric relation $\sim$. Then $(\underline{\mathcal{F}_\mathcal{J}}, \approx)$ is a K-type system iff the following condition is true:*

$$\sigma_1 \to \tau_1 \sim \iota_1 \sim \ldots \sim \iota_n \sim \sigma_2 \to \tau_2 \quad \textit{implies} \quad \sigma_1 \approx \sigma_2 \textit{ and } \tau_1 \approx \tau_2$$

for all $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}_\mathcal{J}, n \in \mathbb{N}_0,$ and $\iota_1, \ldots, \iota_n \in \mathcal{J}$.

Proof: If $\sigma_1 \to \tau_1 \sim \iota_1 \sim \ldots \sim \iota_n \sim \sigma_2 \to \tau_2$ holds for $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}_\mathcal{J}$ and $\iota_1, \ldots, \iota_n \in \mathcal{J}$ and $(\underline{\mathcal{F}_\mathcal{J}}, \approx)$ is a K-type system then $\sigma_1 \approx \sigma_2$ and $\tau_1 \approx \tau_2$ hold as well.

Let $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{F}_\mathcal{J}$ be types such that $\sigma_1 \to \tau_1 \approx \sigma_2 \to \tau_2$. We have to show $\sigma_1 \approx \sigma_2$ and $\tau_1 \approx \tau_2$. As well known from universal algebra we can get from $\sigma_1 \to \tau_1$ to $\sigma_2 \to \tau_2$ by replacing occurrences of subtypes δ by δ' in $\sigma_1 \to \tau_1$ with $\delta \sim \delta'$. Thus, we get a sequence $\sigma_1 \to \tau_1 \rightsquigarrow \varphi_1 \rightsquigarrow \ldots \rightsquigarrow \varphi_n \rightsquigarrow \sigma_2 \to \tau_2$. Consider subsequences of the form $\varphi_i = \sigma_1' \to \tau_1' \rightsquigarrow \varphi_{i+1} = \iota_1 \rightsquigarrow \ldots \rightsquigarrow \varphi_{i+k} = \iota_k \rightsquigarrow \varphi_{i+k+1} = \sigma_2' \to \tau_2'$ with $\iota_1, \ldots, \iota_k \in \mathcal{J}$. For $k \geq 1$ the condition of the proposition yields $\sigma_1' \approx \tau_1'$ and $\sigma_2' \approx \tau_2'$. So we only have to deal with the case $k = 0$: we must show that $\sigma_1' \to \tau_1' \rightsquigarrow \sigma_2' \to \tau_2'$ implies $\sigma_1' \approx \sigma_2'$ and $\tau_1' \approx \tau_2'$ when $\sigma_2' \to \tau_2'$ is the result of the replacement of one occurrence of a subtype δ as described above. This is clear if δ is a proper subtype of σ_1' or τ_1'. The case $\delta = \sigma_1' \to \tau_1'$ follows from the assumption choosing $n = 0$. □

Lemma 1.9 *For any freely generated groupoid $\underline{\mathcal{F}}$ with a finite binary relation $\sim$ it is decidable whether $\sigma \approx \tau$ holds for the generated relation $\approx$ and for $\sigma, \tau \in \mathcal{F}$.*

Proof: Let $\mathcal{S} := \{\delta \in \mathcal{F} \mid \exists \delta' \in \mathcal{F} : \delta \sim \delta' \text{ or } \delta' \sim \delta\} \cup \{\sigma, \tau\}$ and $\mathcal{T}$ be the set of all subtypes of elements of $\mathcal{S}$. The relation $\approx'$ is defined as the smallest reflexive, symmetric and transitive extension of $\sim$ that satisfies

$$\sigma_1 \approx' \tau_1,\ \sigma_2 \approx' \tau_2,\ \sigma_1 \to \sigma_2, \tau_1 \to \tau_2 \in \mathcal{T} \qquad \Rightarrow \qquad \sigma_1 \to \sigma_2 \approx' \tau_1 \to \tau_2$$

for all $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \mathcal{T}$. Since $\mathcal{T}$ is finite, so is $\approx'$. Given $\sigma \approx \tau$, there are $\sigma_0, \sigma_1, \ldots, \sigma_{n+1} \in \mathcal{F}$ with $\sigma = \sigma_0 \approx \sigma_1 \approx \ldots \approx \sigma_{n+1} = \tau$ such that σ_{i+1} can be obtained from σ_i ($i = 0, \ldots, n$) by replacing a subtype δ by some δ' with $\delta \sim \delta'$ or $\delta' \sim \delta$. By induction on the sum of the length of the σ_i one can be prove $\sigma = \sigma_0 \approx' \sigma_{n+1} = \tau$ which is left to the reader. This completes the proof because $\approx'$ is finite. $\square$

Of particular interest are presentations for type systems using a function which maps atomic types to arbitrary ones. In other words, the generating relation is of the form $\{\iota_i \sim \tau_i \mid \iota_i \in \mathcal{J}, \tau_i \in \mathcal{F}_{\mathcal{J}}\}$. If a type system $(\underline{\mathcal{F}_{\mathcal{J}}}, \approx)$ is *finitely presented*, i.e. $\mathcal{J}$ is finite and there is a finite relation $\sim$ generating $\approx$, we get the following results as consequences of the two lemmas:

Theorem 1.10 *It is decidable whether a finitely presented type system is a K-type system.*

Corollary 1.11 *Each type system generated by a function is a K-type system.*

Proof: Follows from Lemma 1.8 because there are no nontrivial sequences. $\square$

Because of Proposition 1.6, type systems generated by functions lead to calculi with the subject reduction property. For finitely presented K-type systems, even the converse direction of Corollary 1.11 is true:

Theorem 1.12 *Each finitely presented K-type system is generated by a function.*

Proof: We sketch an algorithm that takes a finite presentation $\sim$ of $(\underline{\mathcal{F}_{\mathcal{J}}}, \approx)$ and gives a function $\varphi \colon \mathcal{J} \to \mathcal{F}_{\mathcal{J}}$ such that $\approx$ is the closure of $\{(\iota, \varphi(\iota)) \mid \iota \in \mathcal{J}\}$. Each pair $\sigma_1 \to \sigma_2 \sim \tau_1 \to \tau_2$ is replaced by $\sigma_1 \sim \tau_1$ and $\sigma_2 \sim \tau_2$. This is done at the very beginning of the algorithm and whenever such a pair occurs. Every two pairs $\iota \sim \sigma$ and $\iota \sim \tau$ with $len(\sigma) \geq len(\tau)$ are replaced by $\sigma \sim \tau$ and $\iota \sim \tau$, where $len(\tau)$ denotes the number of arrows in τ. (It does not matter which one to choose in case of $len(\sigma) = len(\tau)$.) At the end the resulting relation defines the function φ.

For proving the termination let l denote the maximal length of types related by $\sim$ and $m_k := |\{\tau \in \mathcal{F}_{\mathcal{J}} \mid \mathrm{len}(\tau) = k \text{ and } \exists i = 1, \ldots, n \text{ with } \iota_i \sim \tau \text{ or } \tau \sim \iota_i\}|$ for $k = 2, \ldots, l$. One has to verify that the weight $(m_l, m_{l-1}, \ldots, m_2) \in \mathbb{N}^{l-1}$ decreases w.r.t. to the lexicographical order during each loop, i.e. while doing one replacement of the second form followed by several ones of the first. Since the lexicographical order is well founded the algorithm terminates. $\square$

2 Strong normalization of $\lambda_{(\underline{\mathcal{F}},\approx)}$-terms

N. Mendler has given a condition characterizing the normalization of a special kind of K-type systems. In this section we generalize this for arbitrary finitely presented K-type systems by using the results of Section 1. Moreover, we give a graph theoretical characterization for checking this condition. As in Section 1, we consider type systems $(\underline{\mathcal{F}}_{\mathcal{J}}, \approx)$.

Definition 2.1 *Let $\tau \in \mathcal{F}$ be a type. The sets Pos(τ) and Neg(τ) are defined by structural recursion: For atomic types $\iota \in \mathcal{J}$ we set*

$$\begin{aligned} Pos(\iota) &= \{\iota\}, \\ Neg(\iota) &= \phi, \end{aligned}$$

and for function types $\sigma \to \tau$:

$$\begin{aligned} Pos(\sigma \to \tau) &= \{\sigma \to \tau\} \cup Neg(\sigma) \cup Pos(\tau), \\ Neg(\sigma \to \tau) &= Pos(\sigma) \cup Neg(\tau). \end{aligned}$$

A type σ occurs positively *in τ if $\sigma \in Pos(\tau)$ and* negatively *if $\sigma \in Neg(\tau)$. A relation $\sim$ is* negative *if there are $\sigma, \tau \in \mathcal{F}_{\mathcal{J}}$ with $\sigma \approx \tau$ (where $\approx$ is the congruence relation generated by $\sim$) such that σ occurs negatively in τ. Otherwise $\sim$ is* positive.

In [Men91], N. Mendler has shown the normalization property for calculi with positive type systems:

Proposition 2.2 *Let $(\underline{\mathcal{F}}_{\mathcal{J}}, \approx)$ be a type system generated by a function $\varphi: \mathcal{J} \to \mathcal{F}_{\mathcal{J}}$ corresponding to a relation $\sim$ on $\mathcal{J} \times \mathcal{F}_{\mathcal{J}}$. Then the $\lambda_{(\underline{\mathcal{F}},\approx)}$-calculus is strongly normalizing iff $\sim$ is positive.*

Unfortunately, it is not obvious whether an arbitrary relation $\{\iota_i \sim \tau_i \mid i \in I\}$ on a free groupoid $\underline{\mathcal{F}}_{\mathcal{J}}$ is positive or not. In order to characterize the positivity of $\sim$ we consider the type graph of the induced type system.

Definition 2.3 *Let $(\underline{\mathcal{F}}_{\mathcal{J}}, \approx)$ be a type system generated by a relation $\sim$ as in Proposition 2.2. The* generating type graph *$\mathcal{E}^{\sim}_{\mathcal{J}}$ of $(\underline{\mathcal{F}}_{\mathcal{J}}, \approx)$ consists of the set of vertices $\mathcal{J} \cup \{\tau \in \mathcal{F} \mid \exists \iota \in \mathcal{J}: \iota \sim \tau\}$ and the edges*

- *$\sigma \xrightarrow{-1} \tau$ if σ occurs negatively in τ,*
- *$\sigma \xrightarrow{1} \tau$ if $\sigma \neq \tau$ and σ occurs positively in τ,*
- *$\sigma \xleftrightarrow{1} \tau$ if $\sigma \neq \tau$ and $\sigma \sim \tau$ or $\tau \sim \sigma$.*

(We write $\sigma \xleftrightarrow{1} \tau$ for $\sigma \underset{1}{\overset{1}{\rightleftarrows}} \tau$.)

Lemma 2.4 *Let $(\underline{\mathcal{F}_{\mathcal{J}}}, \approx)$ be a type system where $\approx$ is generated by the relation $\sim$. Then the following are equivalent:*

(1) The relation $\sim$ is positive.

(2) The product of the labels of the edges of every circle

$$\iota_{i_1} \xrightarrow{x_1} \tau_{i_2} \xleftrightarrow{1} \iota_{i_2} \xrightarrow{x_2} \ldots \xleftrightarrow{1} \iota_{i_n} \xrightarrow{x_n} \tau_{i_1} \xleftrightarrow{1} \iota_{i_1}$$

in the generating type graph $\mathcal{E}^{\sim}_{\mathcal{J}}$ of $(\underline{\mathcal{F}_{\mathcal{J}}}, \approx)$ is positive.

Proof: The proof of (1) implies (2) is clear. For the reverse direction let us assume that $\sim$ is negative. We construct a circle in the generating type graph where the product of the labels is negative. Consider the *total type graph* $\mathcal{G}^{\sim}_{\mathcal{J}}$ of $(\underline{\mathcal{F}_{\mathcal{J}}}, \approx)$ which is defined as follows:

- The vertices of $\mathcal{G}^{\sim}_{\mathcal{J}}$ are all the types of $\mathcal{F}_{\mathcal{J}}$.
- The labelled edges $\sigma \xrightarrow{1} \tau$ and $\sigma \xrightarrow{-1} \tau$ are defined as in the generating type graph.
- $\sigma \xleftrightarrow{1} \tau$ is an edge if $\sigma \neq \tau$ and τ can be obtained by replacing a subtype φ of σ by some ψ with $\varphi \sim \psi$ or $\psi \sim \varphi$.

W.l.o.g we assume that no ι_i occurs negatively in τ_i and $\tau_i \notin \mathcal{J}$ (for $\iota_i \sim \tau_i$). We use σ^k as an abbreviation for

$$\sigma_{k,1} \xleftrightarrow{1} \sigma_{k,2} \xleftrightarrow{1} \ldots \xleftrightarrow{1} \sigma_{k,n_k}.$$

If the relation $\sim$ is negative, there is a circle

$$\mathcal{K} = \sigma_{1,n_1} \xrightarrow{x_1} \sigma^2 \xrightarrow{x_2} \ldots \xrightarrow{x_{m-1}} \sigma^m \xrightarrow{x_m} \sigma^1$$

such that $x_1 \cdot \ldots \cdot x_m$ is negative. We sketch an algorithm that computes a closed path

$$\iota_{i_1} \xrightarrow{y_1} \tau_{i_2} \xleftrightarrow{1} \iota_{i_2} \xrightarrow{y_2} \ldots \xleftrightarrow{1} \iota_{i_n} \xrightarrow{y_n} \tau_{i_1} \xleftrightarrow{1} \iota_{i_1}$$

in the generating type graph $\mathcal{E}^{\sim}_{\mathcal{J}}$ such that $x_1 \cdot \ldots \cdot x_m = y_1 \cdot \ldots \cdot y_n = -1$. If there is a path $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{x} \sigma \xleftrightarrow{1} \sigma'$ in $\mathcal{K}$ with $\sigma' \notin \mathcal{J}$, there are four cases:

- If $x = 1$ and ι_i occurs positively in σ':
  replace $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{1} \sigma \xleftrightarrow{1} \sigma'$ *by* $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{1} \sigma'$.
- If $x = -1$ and ι_i occurs negatively in σ':
  replace $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{-1} \sigma \xleftrightarrow{1} \sigma'$ *by* $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{-1} \sigma'$.

- If σ' can be obtained by replacing some τ_j by ι_j in σ such that for $x = 1$ the type ι_i does not occur positively, for $x = -1$ it does not occur negatively in σ': *replace* $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{x} \sigma \xleftrightarrow{1} \sigma'$ *by* $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{y} \tau_j \xleftrightarrow{1} \iota_j \xrightarrow{z} \sigma'$.
  If the occurrence of τ_j that has to be replaced is positive then choose $y = x$ and $z = 1$, otherwise $y = -x$ and $z = -1$.

- If σ' can be obtained by replacing ι_i by τ_i in σ such that ι_i is not a subtype of τ_i: *replace* $\tau_i \xleftrightarrow{1} \iota_i \xrightarrow{x} \sigma \xleftrightarrow{1} \sigma'$ *by* $\tau_i \xrightarrow{x} \sigma'$.

If $\mathcal{K}$ does not contain any atomic type we have $\sigma_{k,i} = \varphi_{k,i} \to \psi_{k,i}$ for all $k = 1, \ldots, m$, $i = 1, \ldots, n_k$. In this case, replace every sequence

$$\sigma^k \xrightarrow{x_k} \varphi_{k+1,1} \to \psi_{k+1,1} \xleftrightarrow{1} \ldots \xleftrightarrow{1} \varphi_{k+1,n_{k+1}} \to \psi_{k+1,n_{k+1}} \xrightarrow{x_{k+1}} \sigma^{k+2}$$

$(k = 1, \ldots, m)$ by

- $\sigma^k \xrightarrow{-x_k} \varphi_{k+1,1} \xleftrightarrow{1} \ldots \xleftrightarrow{1} \varphi_{k+1,n_{k+1}} \xrightarrow{-x_{k+1}} \sigma^{k+2}$, if σ^k is contained in $\varphi_{k+1,k}$,

- $\sigma^k \xrightarrow{x_k} \psi_{k+1,1} \xleftrightarrow{1} \ldots \xleftrightarrow{1} \psi_{k+1,n_{k+1}} \xrightarrow{x_{k+1}} \sigma^{k+2}$, if σ^k is contained in $\psi_{k+1,k}$.

Deleting trivial edges $\sigma \xleftrightarrow{1} \sigma$ and contracting edges $\sigma_1 \xrightarrow{x} \sigma_2 \xrightarrow{y} \sigma_3$ to $\sigma_1 \xrightarrow{x \cdot y} \sigma_3$ completes the algorithm. During each loop the number of edges $\sigma \xleftrightarrow{1} \sigma'$ with $\sigma \not\leadsto \sigma'$ and $\sigma' \not\leadsto \sigma$ or the length of types that are related by $\xleftrightarrow{1}$ increases, hence, the algorithm terminates. $\square$

As a consequence of Lemma 2.4 we get:

Corollary 2.5 *It is decidable whether a finite relation being defined by a function is positive.*

In the light of Theorem 1.12 and Proposition 2.2, Corollary 2.5 implies:

Theorem 2.6 *Let $(\underline{\mathcal{F}_\mathcal{J}}, \approx)$ be a finitely presented K-type system. Then it is decidable whether the $\lambda_{(\underline{\mathcal{F}_\mathcal{J}}, \approx)}$-calculus is strongly normalizing.*

3 Typeability and principal typing with positive K-type systems

We study the subject of typeability of terms in K-type systems. We call a λ-term t *typeable* if there is a K-type system such that we can derive a judgement $\Gamma \vdash t : \tau$. In [Urz95], P. Urzyczyn has shown that the λ-term $\overline{2}\,\overline{2}K$ with $\overline{2} := \lambda yx.y(yx)$ and $K := \lambda xy.x$ is not typeable in System F. However, using the positive K-type system $(\underline{\mathcal{F}_\mathcal{J}}, \approx)$ generated by $\mathcal{J} := \{\iota_1, \iota_2\}$ and $\iota_1 \sim \iota_2 \to \iota_1$, there is a deduction for $\vdash \overline{2}\,\overline{2}K :$

$\iota_1 \to \iota_1$. On the other hand, the λ-term $\lambda x.xx$ is easily typeable in System F, but not with a positive K-type system.

As shown in [Wel94], the typeability problem is undecidable in System F. In the following we will show that this is not the case for positive K-type systems.

Definition 3.1 *Let* $(\underline{\mathcal{F}}, \approx)$ *and* $(\underline{\mathcal{F}}', \approx')$ *be type systems and* $\varphi\colon \underline{\mathcal{F}} \to \underline{\mathcal{F}}'$ *a groupoid homomorphism (i.e.* $\varphi(\sigma \to \tau) = \varphi(\sigma) \to \varphi(\tau)$*). We call* $\varphi\colon (\underline{\mathcal{F}}, \approx) \to (\underline{\mathcal{F}}', \approx')$ *a* type system homomorphism *if* $\sigma \approx \tau$ *implies* $\varphi(\sigma) \approx' \varphi(\tau)$.

Let t be a λ-term. A K-type system $(\underline{\mathcal{F}}, \approx)$ *is called* principal K-type system for *t if t is typeable in* $(\underline{\mathcal{F}}, \approx)$ *and for every other such type K-type system* $(\underline{\mathcal{F}}', \approx')$ *there is a type system homomorphism* $\varphi\colon (\underline{\mathcal{F}}, \approx) \to (\underline{\mathcal{F}}', \approx')$.

Lemma 3.2 *Let t be a λ-term and* $(\underline{\mathcal{F}}, \approx)$ *a K-type system such that* $\Gamma \vdash t : \sigma$ *and* $\Gamma \vdash t : \tau$ *are judgements. If Γ contains all bounded variables of t then* $\sigma \approx \tau$ *holds.*

Proof: By induction on the structure of t. □

F. Cardone and M. Coppo showed in [CC91] the principal typing property for the λ-calculus with recursive types. Here is the analogous result for K-type systems:

Theorem 3.3 (Principal Typing Theorem) *For each λ-term t exists a principal K-type system.*

Proof: We assume that no variable in t occurs freely and bounded or bounded more than once. Given the set $\{x_i \mid i \in I\}$ of all variables contained in t we define $\mathcal{J} := \{\iota_i \mid i \in I\}$ and $\Gamma := \{(x_i : \iota_i) \mid i \in I\}$. For a subterm s of t, the type σ_s is defined by structural recursion:

- If s is a variable with $(s : \iota) \in \Gamma$, then $\sigma_s := \iota$.
- If s is an application $s_1 s_2$, take a fresh variable κ, define $\sigma_s := \kappa$, and let $\sigma_{s_1} \sim \sigma_{s_2} \to \kappa$.
- If s is an abstraction $\lambda x.s_1$ with $(x : \iota) \in \Gamma$, then $\sigma_s := \iota \to \sigma_{s_1}$.

For all subterms s of t, we get judgements $\Gamma \vdash s : \sigma_s$ in the type system $(\underline{\mathcal{F}}_{\mathcal{J}\cup\mathcal{L}}, \approx)$ where $\mathcal{L}$ is the set of all fresh variables κ used in the definition above and $\approx$ being the K-closure of $\sim$ (i.e. the smallest symmetric, transitive extension of $\sim$ satisfying condition (K)).

Let $(\underline{\mathcal{F}}', \approx')$ be a K-type system with a basis Γ' and a type $\tau \in \mathcal{F}'$ such that $\Gamma' \vdash' t : \tau$ is a judgement in this type system. Assume Γ' to contain every (free or bound) variable in t. We define a map $\varphi_0\colon \mathcal{J} \to \mathcal{F}'$ by $\varphi_0(\iota_i) = \tau_i$ where $\{(x_i : \tau_i)\} \in \Gamma'$. This map can be extended to $\varphi\colon \mathcal{F}_{\mathcal{J}\cup\mathcal{L}} \to \mathcal{F}'$ in the following way:

$$\varphi(\sigma) = \begin{cases} \varphi_0(\sigma) & \text{if } \sigma \in \mathcal{J} \\ \tau & \text{if } \sigma = \sigma_{s_1 s_2} \in \mathcal{L} \text{ for } \tau \in \mathcal{F} \text{ with } \Gamma' \vdash' s_1 s_2 : \tau \\ \varphi(\sigma_1) \to \varphi(\sigma_2) & \text{if } \sigma = \sigma_1 \to \sigma_2 \end{cases}$$

Clearly, φ defines a groupoid homomorphism. For proving that φ is a type system homomorphism it is sufficient to prove that $\sigma \sim \sigma'$ implies $\varphi(\sigma) \approx' \varphi(\sigma')$.

Let s be a subterm of t and $\Gamma \vdash s : \sigma_s$. It can be shown by induction of the structure of s that $\Gamma' \vdash' s : \varphi(\sigma_s)$ is a judgement for s in $(\underline{\mathcal{F}}', \approx')$.

Now assume $\sigma \sim \sigma'$ for $\sigma, \sigma' \in \mathcal{F}_{\mathcal{J}\cup\mathcal{L}}$. Because of the definition of $\sim$ there is a subterm $s = s_1 s_2$ of t and a variable $\kappa \in \mathcal{L}$ such that $\sigma = \sigma_{s_1}$, $\sigma' = \sigma_{s_2} \to \kappa$, and $\sigma_s = \kappa$. As shown above, $\Gamma' \vdash' s_1 : \varphi(\sigma)$, $\Gamma' \vdash' s_2 : \varphi(\sigma_{s_2})$, and $\Gamma' \vdash' s_1 s_2 : \varphi(\kappa)$ are judgements. Because of the Generation Lemma 1.3 there is a $\varrho \in \mathcal{F}'$ with $\Gamma' \vdash' s_1 : \varrho \to \varphi(\kappa)$ and $\Gamma' \vdash' s_2 : \varrho$. By Lemma 3.2 we get $\varphi(\sigma) \approx' \varrho \to \varphi(\kappa)$ and $\varphi(\sigma_{s_2}) \approx' \varrho$. This implies $\varphi(\sigma) \approx' \varphi(\sigma_{s_2}) \to \varphi(\kappa) = \varphi(\sigma_{s_2} \to \kappa) = \varphi(\sigma')$ which shows that φ is a type system homomorphism. This completes the proof. □

Corollary 3.4 *It is decidable whether there is a positive K-type system in which a given λ-term t is typeable.*

Proof: Suppose the principal K-type system $(\underline{\mathcal{F}}, \approx)$ for t being negative and let $(\underline{\mathcal{F}}', \approx')$ be another K-type system in which t is typeable. Since the relation $\approx$ is negative there are types $\sigma, \tau \in \mathcal{F}$ with $\sigma \approx \tau$ such that σ occurs negatively in τ. Because of Theorem 3.3 there is a type system homomorphism $\varphi\colon (\underline{\mathcal{F}}, \approx) \to (\underline{\mathcal{F}}', \approx')$ with $\varphi(\sigma) \approx' \varphi(\tau)$. The type $\varphi(\sigma)$ occurs negatively in $\varphi(\tau)$, thus, the type system $(\underline{\mathcal{F}}', \approx')$ is negative as well. Since the generating relation $\sim$ of $\approx$ satisfies the K-property, so does $\approx$ (this follows from Lemma 1.8 because every pair $\sigma_1 \sim \sigma_2$ contains an atomic type). Moreover, $\sim$ is finite, thus we can apply Corollary 2.5 and check whether the principal K-type system $(\underline{\mathcal{F}}, \approx)$ is positive. □

In this paper we have examined positive type congruences for the λ-calculus. As shown by P. Urzyczyn in [Urz95] these are equivalent to positive recursive types with respect to typeability, i.e. a λ-term is typeable using positive recursive types iff it is typeable in a positive K-type system. In this light we have given a new proof for the decidability of typeability with positive recursive types.

References

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 118–309. Clarendon Press, 1992.

[CC91] F. Cardone and M. Coppo. Type inference with recursive types: syntax and semantics. *Information and Computation*, 92:48–80, 1991.

[Gun92] C. Gunter. *Semantics of Programming Languages. Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[Mar95] M. Marz. Monomorphe Typsysteme des λ-Kalküls, Juni 1995. Diplomarbeit, 46pp.

[Men91] N. P. Mendler. Inductive types and type constraints in the second-order lambda-calculus. *Annals of Pure and Applied Logic*, 51:159–172, 1991.

[Sta94] R. Statman. Recursive types and the subject reduction theorem. Technical Report 94-164, Carnegie Mellon University, March 1994.

[Urz95] P. Urzyczyn. Positive recursive type assignment. Technical Report 95-03(203), Instytut Informatyki, Uniwersytet Warszawski, 1995.

[Wel94] J. B. Wells. Typability and type checking in the second order λ-calculus are equivalent and undecidable. In *9th LICS conference*, pages 176–185. IEEE Computer Society Press, 1994.