(ix) *If M has the $\lambda\mathsf{K}^-$ property then M β-reduces to only finitely many N.* This follows by (vii) and (viii).

(x) *If M has the $\lambda\mathsf{K}^-$ property then M is strongly β-normalizable.* By (i), (iii) and (ix).

(xi) *If M has the $\lambda\mathsf{K}^-$ property then M is strongly $\beta\eta$-normalizable.* By (v) and (x).

(xii) *For each M there is an N with the $\lambda\mathsf{K}^-$ property such that $N \twoheadrightarrow_{\boldsymbol{\beta\eta}} M$.* First expand M by η expansion so that every subterm of M beginning with a lambda is a lambda prefix followed by a matrix of type 0. Let $a : \alpha$ and $f : 0{\to}(0{\to}0)$ be new variables. For each type $T = T_1{\to}\ldots{\to}T_t{\to}\alpha$ with $T_i = T_{i,1}{\to}\ldots{\to}T_{i,k_i}{\to}\alpha_i$ for $i = 1,\ldots,t$ define terms $U_A : T$ recursively by

$$\begin{aligned} U_0 &= a; \\ U_T &= \lambda x_1 \ldots x_t.f(x_1 U_{T_{1,1}} \ldots U_{T_{1,k_1}}) \ldots \\ &\quad (f(x_{t-1}U_{T_{t-1,1}} \ldots U_{T_{t-1,k_{t-1}}})(x_t U_{T_{t,1}} \ldots U_{T_{t,k_t}}))..). \end{aligned}$$

Now recursively replace each dummy λx occurring $\lambda x\lambda y \ldots \lambda z.X$ with $x : T$ and $X : 0$ by $\lambda x\lambda y \ldots \lambda z.KX(xU_{T_1} \ldots U_{T_t})$. Clearly the resulting N satisfies $N \twoheadrightarrow_{\boldsymbol{\beta\eta}} M$ and the $\lambda\mathsf{K}^-$ property, since all dummy lambdas appear in $K : 1_2$.

(xiii) *Every typable term is strongly $\beta\eta$ normalizable.* By (xi) and (xii). ■

Still another proof is to be found in de Vrijer [1987] in which for a typed term M a computation is given of the langest reduction path to $\boldsymbol{\beta}$-nf.

2.3. Checking and finding types

There are several natural problems concerning type systems.

2.3.1. Definition. (i) The problem of *type checking* consists of determining, given basis Γ, term M and type A whether $\Gamma \vdash M : A$.

(ii) The problem of *typeability* consists of given a term M determining whether M has some type with respect to some Γ.

(iii) The problem of *type reconstruction* ('finding types') consists of finding all possible types A and bases Γ that type a given M.

(iv) The *inhabitation problem* consists of finding out whether a given type A is inhabited by some term M in a given basis Γ.

(v) The *enumeration problem* consists of determining for a given type A and a given context Γ all possible terms M such that $\Gamma \vdash M : A$.

The five problems may be summarized stylistically as follows.

$$\begin{aligned} \Gamma \;&\vdash_{\lambda\to}\; M : A\,? && \textit{type checking}; \\ \exists A, \Gamma\,[\Gamma \;&\vdash_{\lambda\to}\; M : A]\,? && \textit{typeability}; \\ ? \;&\vdash_{\lambda\to}\; M : \;? && \textit{type reconstruction}; \\ \exists M\,[\Gamma \;&\vdash_{\lambda\to}\; M : A]\,? && \textit{inhabitation}; \\ \Gamma \;&\vdash_{\lambda\to}\; \;? : A && \textit{enumeration.} \end{aligned}$$

In another notation this is the following.

$$\begin{array}{rll} & M \in \Lambda^{\Gamma}_{\to}(A)\,? & \textit{type checking};\\ \exists A,\Gamma & M \in \Lambda^{\Gamma}_{\to}(A)? & \textit{typeability};\\ & M \in \Lambda^{?}_{\to}(?) & \textit{type reconstruction};\\ & \Lambda^{\Gamma}_{\to}(A) \neq \emptyset\,? & \textit{inhabitation};\\ & ? \in \Lambda^{\Gamma}_{\to}(A) & \textit{enumeration.}\end{array}$$

In this section we will treat the problems of type checking, typeability and type reconstruction for the three versions of $\lambda_{\to}$. It turns out that these problems are decidable for all versions. The solutions are essentially simpler for $\lambda^{\mathrm{Ch}}_{\to}$ and $\lambda^{\mathrm{dB}}_{\to}$ than for $\lambda^{\mathrm{Cu}}_{\to}$. The problems of inhabitation and enumeration will be treated in the next section.

One may wonder what is the role of the context Γ in these questions. The problem

$$\exists\Gamma\exists A\,\Gamma \vdash M : A.$$

can be reduced to one without a context. Indeed, for $\Gamma = \{x_1{:}A_1, \ldots, x_n{:}A_n\}$

$$\Gamma \vdash M : A \;\Leftrightarrow\; \vdash (\lambda x_1(:A_1)\ldots\lambda x_n(:A_n).M) : (A_1 \to \ldots \to A_n \to A).$$

Therefore

$$\exists\Gamma\exists A\,[\Gamma \vdash M : A] \iff \exists B\,[\vdash \lambda\vec{x}.M : B].$$

On the other hand the question

$$\exists\Gamma\exists M\,[\Gamma \vdash M : A]\,?$$

is trivial: take $\Gamma = \{x{:}A\}$ and $M \equiv x$. So we do not consider this question.

The solution of the problems like type checking for a fixed context will have important applications for the treatment of constants.

Checking and finding types for $\lambda^{\mathrm{dB}}_{\to}$ and $\lambda^{\mathrm{Ch}}_{\to}$

We will see again that the systems $\lambda^{\mathrm{Ch}}_{\to}$ and $\lambda^{\mathrm{dB}}_{\to}$ are essentially equivalent. For these systems the solutions to the problems of type checking, typeability and type reconstruction are easy. All of the solutions are computable with an algorithm of linear complexity.

2.3.2. Proposition (Type checking for $\lambda^{\mathrm{dB}}_{\to}$). *Let Γ be a basis of $\lambda^{\mathrm{dB}}_{\to}$. Then there is a computable function $\mathtt{type}_\Gamma : \Lambda_{\mathbb{T}} \to \mathbb{T} \cup \{\mathtt{error}\}$ such that*

$$M \in \Lambda^{\Gamma}_{\to\mathrm{Ch}}(A) \iff \mathtt{type}_\Gamma(M) = A.$$

Proof. Define

$$\begin{array}{rcll} \mathtt{type}_\Gamma(x) & = & \Gamma(x); & \\ \mathtt{type}_\Gamma(MN) & = & B, & \text{if } \mathtt{type}_\Gamma(M) = \mathtt{type}_\Gamma(N){\to}B,\\ & = & \mathtt{error}, & \text{else;}\\ \mathtt{type}_\Gamma(\lambda x{:}A.M) & = & A{\to}\mathtt{type}_{\Gamma\cup\{x:A\}}(M), & \text{if } \mathtt{type}_{\Gamma\cup\{x:A\}}(M) \neq \mathtt{error},\\ & = & \mathtt{error}, & \text{else.}\end{array}$$

Then the statement follows by induction on the structure of M. ■

2.3.3. Corollary. *Typeability and type reconstruction for $\lambda^{\mathrm{dB}}_{\rightarrow}$ are computable. In fact one has the following.*

(i) $M \in \Lambda^{\Gamma}_{\rightarrow\mathrm{dB}} \iff \mathtt{type}_{\Gamma}(M) \neq \mathtt{error}$.

(ii) *Each $M \in \Lambda^{\Gamma}_{\rightarrow\mathrm{dB}}(\mathtt{type}_{\Gamma})$ has a unique type; in particular*

$$M \in \Lambda^{\Gamma}_{\rightarrow\mathrm{dB}}(\mathtt{type}_{\Gamma}(M)).$$

Proof. By the proposition. ■

For $\lambda^{\mathrm{Ch}}_{\rightarrow}$ things are essentially the same, except that there are no bases needed, since variables come with their own types.

2.3.4. Proposition (Type checking for $\lambda^{\mathrm{Ch}}_{\rightarrow}$). *There is a computable function* $\mathtt{type} : \Lambda_{\rightarrow\mathrm{Ch}} \to \mathbb{T} \cup \{\mathtt{error}\}$ *such that*

$$M \in \Lambda_{\rightarrow\mathrm{Ch}}(A) \iff \mathtt{type}(M) = A.$$

Proof. Define

$$\begin{array}{rcll} \mathtt{type}(x^A) & = & A; & \\ \mathtt{type}(MN) & = & B, & \text{if } \mathtt{type}(M) = \mathtt{type}(N){\rightarrow}B, \\ & = & \mathtt{error}, & \text{else;} \\ \mathtt{type}(\lambda x^A.M) & = & A{\rightarrow}\mathtt{type}(M), & \text{if } \mathtt{type}(M) \neq \mathtt{error}, \\ & = & \mathtt{error}, & \text{else.} \end{array}$$

Then the statement follows again by induction on the structure of M. ■

2.3.5. Corollary. *Typeability and type reconstruction for $\lambda^{\mathrm{Ch}}_{\rightarrow}$ are computable. In fact one has the following.*

(i) $M \in \Lambda_{\rightarrow\mathrm{Ch}} \iff \mathtt{type}(M) \neq \mathtt{error}$.

(ii) *Each $M \in \Lambda_{\rightarrow\mathrm{Ch}}$ has a unique type; in particular $M \in \Lambda_{\rightarrow\mathrm{Ch}}(\mathtt{type}(M))$.*

Proof. By the proposition. ■

Checking and finding types for $\lambda^{\mathrm{Cu}}_{\rightarrow}$

We now will show the computability of the three questions for $\lambda^{\mathrm{Cu}}_{\rightarrow}$. This occupies 2.3.6 - 2.3.16 and in these items $\vdash$ stands for $\vdash^{\mathbb{T}^{\infty}}_{\lambda_{\rightarrow\mathrm{Cu}}}$.

Let us first make the easy observation that in $\lambda^{\mathrm{Cu}}_{\rightarrow}$ types are not unique. For example $\mathsf{I} \equiv \lambda x.x$ has as possible type $\alpha{\rightarrow}\alpha$, but also $(\boldsymbol{\beta}{\rightarrow}\boldsymbol{\beta}){\rightarrow}(\boldsymbol{\beta}{\rightarrow}\boldsymbol{\beta})$ and $(\alpha{\rightarrow}\boldsymbol{\beta}{\rightarrow}\boldsymbol{\beta}){\rightarrow}(\alpha{\rightarrow}\boldsymbol{\beta}{\rightarrow}\boldsymbol{\beta})$. Of these types $\alpha{\rightarrow}\alpha$ is the 'most general' in the sense that the other ones can be obtained by a substitution in α.

2.3.6. Definition. (i) A substitutor is an operation $* : \mathbb{T} \to \mathbb{T}$ such that

$$*(A \to B) \equiv *(A) \to *(B).$$

(ii) We write A^* for $*(A)$.

(iii) Usually a substitution $*$ has a finite support, that is, for all but finitely many type variables α one has $\alpha^* \equiv \alpha$ (the support of $*$ being

$$\sup(*) = \{\alpha \mid \alpha^* \not\equiv \alpha\}).$$

In that case we write

$$*(A) = A[\alpha_1 := \alpha_1^*, \ldots, \alpha_n := \alpha_n^*],$$

where $\{\alpha_1, \ldots, \alpha_n\} \supseteq \sup(*)$. We also write

$$* = [\alpha_1 := \alpha_1^*, \ldots, \alpha_n := \alpha_n^*]$$

and

$$* = [\,]$$

for the identity substitution.

2.3.7. Definition. (i) Let $A, B \in \mathbb{T}$. A *unifier* for A and B is a substitutor $*$ such that $A^* \equiv B^*$.

(ii) The substitutor $*$ is a *most general unifier for A and B if*

- $A^* \equiv B^*$
- $A^{*_1} \equiv B^{*_1} \;\Rightarrow\; \exists\, *_2 \,.\, *_1 \equiv *_2 \circ *$.

(iii) *Let $E = \{A_1 = B_1, \ldots, A_n = B_n\}$ be a finite set of equations between types. The equations do not need to be valid. A* unifier *for E is a substitutor $*$ such that $A_1^* \equiv B_1^*$ & $\cdots$ & $A_n^* \equiv B_n^*$. In that case one writes $* \models E$. Similarly one defines the notion of a most general unifier for E.*

2.3.8. Examples. The types $\beta \to (\alpha \to \beta)$ and $(\gamma \to \gamma) \to \delta$ have a unifier. For example $* = [\beta := \gamma \to \gamma,\ \delta := \alpha \to (\gamma \to \gamma)]$ or $*_1 = [\beta := \gamma \to \gamma,\ \alpha := \varepsilon \to \varepsilon,\ \delta := \varepsilon \to \varepsilon \to (\gamma \to \gamma)]$. The unifier $*$ is most general, $*_1$ is not.

2.3.9. Definition. A is a *variant* of B if for some $*_1$ and $*_2$ one has

$$A = B^{*_1} \text{ and } B = A^{*_2}.$$

2.3.10. Example. $\alpha \to \beta \to \beta$ is a variant of $\gamma \to \delta \to \delta$ but not of $\alpha \to \beta \to \alpha$.

Note that if $*_1$ and $*_2$ are both most general unifiers of say A and B, then A^{*_1} and A^{*_2} are variants of each other and similarly for B.

The following result due to Robinson (1965) states that unifiers can be constructed effectively.

2.3.11. THEOREM (Unification theorem). (i) *There is a recursive function U having (after coding) as input a pair of types and as output either a substitutor or* `fail` *such that*

$$\begin{array}{rcl} A \text{ and } B \text{ have a unifier} & \Rightarrow & U(A,B) \text{ is a most general unifier} \\ & & \text{for } A \text{ and } B; \\ A \text{ and } B \text{ have no unifier} & \Rightarrow & U(A,B) = \mathsf{fail}. \end{array}$$

(ii) *There is (after coding) a recursive function U having as input finite sets of equations between types and as output either a substitutor or* `fail` *such that*

$$\begin{array}{rcl} E \text{ has a unifier} & \Rightarrow & U(E) \text{ is a most general unifier for } E; \\ E \text{ has no unifier} & \Rightarrow & U(E) = \texttt{fail}. \end{array}$$

PROOF. Note that $A_1 \to A_2 \equiv B_1 \to B_2$ holds iff $A_1 \equiv B_1$ and $A_2 \equiv B_2$ hold.
(i) Define $U(A,B)$ by the following recursive loop, using case distinction.

$$\begin{array}{rcll} U(\alpha, B) & = & [\alpha := B], & \text{if } \alpha \notin \mathrm{FV}(B), \\ & = & [\,], & \text{if } B = \alpha, \\ & = & \texttt{fail}, & \text{else;} \end{array}$$

$$\begin{array}{rcl} U(A_1 \to A_2, \alpha) & = & U(\alpha, A_1 \to A_2); \\ U(A_1 \to A_2, B_1 \to B_2) & = & U(A_1^{U(A_2,B_2)}, B_1^{U(A_2,B_2)}) \circ U(A_2, B_2), \end{array}$$

where this last expression is considered to be `fail` if one of its parts is. Let $\#_{var}(A,B)$ ='the number of variables in $A \to B$' and $\#_{\to}(A,B)$='the number of arrows in $A \to B$'. By induction on $(\#_{var}(A,B), \#_{\to}(A,B))$ ordered lexicographically one can show that $U(A,B)$ is always defined. Moreover U satisfies the specification.

(ii) If $E = \{A_1 = B_1, \ldots, A_n = B_n\}$, then define $U(E) = U(A,B)$, where $A = A_1 \to \cdots \to A_n$ and $B = B_1 \to \cdots \to B_n$. ■

See [???] for more on unification. The following result due to Parikh [1973] for propositional logic (interpreted by the propositions-as-types interpretation) and Wand [1987] simplifies the proof of the decidability of type checking and typeability for $\lambda_{\to}$.

2.3.12. PROPOSITION. *For every basis Γ, term $M \in \Lambda$ and $A \in \mathbb{T}$ such that $\mathrm{FV}(M) \subseteq \mathrm{dom}(\Gamma)$ there is a finite set of equations $E = E(\Gamma, M, A)$ such that for all substitutors $*$ one has*

$$\begin{array}{rcll} * \models E(\Gamma, M, A) & \Rightarrow & \Gamma^* \vdash M : A^*, & (1) \\ \Gamma^* \vdash M : A^* & \Rightarrow & *_1 \models E(\Gamma, M, A), & (2) \\ & & \text{for some } *_1 \text{ such that } * \text{ and } *_1 \text{ have the same} & \\ & & \text{effect on the type variables in } \Gamma \text{ and } A. & \end{array}$$

PROOF. Define $E(\Gamma, M, A)$ by induction on the structure of M:

$$\begin{array}{rcl} E(\Gamma, x, A) & = & \{A = \Gamma(x)\}; \\ E(\Gamma, MN, A) & = & E(\Gamma, M, \alpha{\to}A) \ \cup \ E(\Gamma, N, \alpha), \\ & & \quad \text{where } \alpha \text{ is a fresh variable;} \\ E(\Gamma, \lambda x.M, A) & = & E(\Gamma \cup \{x{:}\alpha\}, M, \beta) \ \cup \ \{\alpha{\to}\beta = A\}, \\ & & \quad \text{where } \alpha, \beta \text{ are fresh.} \end{array}$$

By induction on M one can show (using the generation lemma (2.1.3)) that (1) and (2) hold. ■

2.3.13. DEFINITION. (i) Let $M \in \Lambda$. Then (Γ, A) is a *principal pair* (pp) for M if

(1) $\Gamma \vdash M : A$.

(2) $\Gamma' \vdash M : A' \ \Rightarrow \ \exists * \, [\Gamma^* \subseteq \Gamma' \ \& \ A^* \equiv A']$.

Here $\{x_1{:}A_1, \ldots\}^* = \{x_1{:}A_1^*, \ldots\}$.

(ii) Let $M \in \Lambda$ be closed. Then A is a *principal type* (pt) for M if

(1) $\vdash M : A$

(2) $\vdash M : A' \ \Rightarrow \ \exists * \ [A^* \equiv A']$.

Note that if (Γ, A) is a *pp* for M, then every variant (Γ', A') of (Γ, A), in the obvious sense, is also a *pp* for M. Conversely if (Γ, A) and (Γ', A') are *pp*'s for M, then (Γ', A') is a variant of (Γ, A). Similarly for closed terms and *pt*'s. Moreover, if (Γ, A) is a *pp* for M, then $\mathrm{FV}(M) = \mathrm{dom}(\Gamma)$.

The following result is independently due to Curry (1969), Hindley (1969) and Milner (1978). It shows that for $\lambda_{\to}$ the problems of type checking and typeability are decidable.

2.3.14. THEOREM (Principal type theorem for $\lambda_{\to}^{\mathrm{Cu}}$). (i) *There exists a computable function pp such that one has*

$$\begin{array}{rcl} M \textit{ has a type} & \Rightarrow & pp(M) = (\Gamma, A), \textit{ where } (\Gamma, A) \textit{ is a pp for } M; \\ M \textit{ has no type} & \Rightarrow & pp(M) = \mathtt{fail}. \end{array}$$

(ii) *There exists a computable function pt such that for closed terms M one has*

$$\begin{array}{rcl} M \textit{ has a type} & \Rightarrow & pt(M) = A, \textit{ where } A \textit{ is a pt for } M; \\ M \textit{ has no type} & \Rightarrow & pt(M) = \mathsf{fail}. \end{array}$$

PROOF. (i) Let $\mathrm{FV}(M) = \{x_1, \ldots, x_n\}$ and set $\Gamma_0 = \{x_1{:}\alpha_1, \ldots, x_n{:}\alpha_n\}$ and $A_0 = \beta$. Note that

$$\begin{array}{rcl} M \text{ has a type} & \Rightarrow & \exists\Gamma \, \exists A \ \ \Gamma \vdash M : A \\ & \Rightarrow & \exists * \ \ \Gamma_0^* \vdash M : A_0^* \\ & \Rightarrow & \exists * \ \ * \models E(\Gamma_0, M, A_0). \end{array}$$

Define

$$\begin{array}{rcll} pp(M) & = & (\Gamma_0^*, A_0^*), & \text{if } U(E(\Gamma_0, M, A_0)) = *; \\ & = & \mathtt{fail}, & \text{if } U(E(\Gamma_0, M, A_0)) = \mathtt{fail}. \end{array}$$

Then $pp(M)$ satisfies the requirements. Indeed, if M has a type, then

$$U(E(\Gamma_0, M, A_0)) = *$$

is defined and $\Gamma_0^* \vdash M : A_0^*$ by (1) in proposition 2.3.12. To show that (Γ_0^*, A_0^*) is a pp, suppose that also $\Gamma' \vdash M : A'$. Let $\widetilde{\Gamma} = \Gamma' \restriction \mathrm{FV}(M)$; write $\widetilde{\Gamma} = \Gamma_0^{*0}$ and $A' = A_0^{*0}$. Then also $\Gamma_0^{*0} \vdash M : A_0^{*0}$. Hence by (2) in proposition 2.3.12 for some $*_1$ (acting the same as $*_0$ on Γ_0, A_0) one has $*_1 \models E(\Gamma_0, M, A_0)$. Since $*$ is a most general unifier (proposition 2.3.11) one has $*_1 = *_2 \circ *$ for some $*_2$. Now indeed

$$(\Gamma_0^*)^{*_2} = \Gamma_0^{*_1} = \Gamma_0^{*0} = \widetilde{\Gamma} \subseteq \Gamma'$$

and

$$(A_0^*)^{*_2} = A_0^{*_1} = A_0^{*0} = A'.$$

If M has no type, then $\neg\exists * \ * \models E(\Gamma_0, M, A_0)$ hence

$$U(\Gamma_0, M, A_0) = \mathtt{fail} = pp(M).$$

(ii) Let M be closed and $pp(M) = (\Gamma, A)$. Then $\Gamma = \emptyset$ and we can put $pt(M) = A$. ■

2.3.15. Corollary. *Type checking and typeability for $\lambda_{\to}$ are decidable.*

Proof. As to type checking, let M and A be given. Then

$$\vdash M : A \iff \exists * \, [A = pt(M)^*].$$

This is decidable (as can be seen using an algorithm—*pattern matching*—similar to the one in Theorem 2.3.11).

As to the question of typeability, let M be given. Then M has a type iff $pt(M) \neq \mathtt{fail}$. ■

The following result is due to Hindley [1969].

2.3.16. Theorem (Second principal type theorem for $\lambda_{\to}^{\mathrm{Cu}}$). (i) *For every type $A \in \mathbb{T}$ one has*

$$\vdash M : A \ \Rightarrow \ \exists M' [M' \twoheadrightarrow_{\beta\eta} M \ \& \ \mathtt{pt}(M') = A].$$

(ii) *For every type $A \in \mathbb{T}$ there exists a basis Γ and term $M \in \Lambda$ such that (Γ, A) is a pp for M.*

PROOF. (i) We present a proof by examples. We choose three situations in which we have to construct an M' that are representative for the general case. Do exercise ?? for the general proof.

Case $M \equiv \lambda x.x$ and $A \equiv (\alpha{\to}\boldsymbol{\beta}){\to}\alpha{\to}\boldsymbol{\beta}$. Then $\mathtt{pt}(M) \equiv \alpha{\to}\alpha$. Take $M' \equiv \lambda xy.xy$. The η-expansion of $\lambda x.x$ to $\lambda xy.xy$ makes subtypes of A correspond to unique subterms of M'.

Case $M \equiv \lambda xy.y$ and $A \equiv (\alpha{\to}\gamma){\to}\boldsymbol{\beta}{\to}\boldsymbol{\beta}$. Then $\mathtt{pt}(M) \equiv \alpha{\to}\boldsymbol{\beta}{\to}\boldsymbol{\beta}$. Take $M' \equiv \lambda xy.\mathsf{K}y(\lambda z.xz)$. The $\boldsymbol{\beta}$-expansion forces x to have a functional type.

Case $M \equiv \lambda xy.x$ and $A \equiv \alpha{\to}\alpha{\to}\alpha$. Then $\mathtt{pt}(M) \equiv \alpha{\to}\boldsymbol{\beta}{\to}\alpha$. Take $M' \equiv \lambda xy.\mathsf{K}x(\lambda f.[fx, fy])$. The $\boldsymbol{\beta}$-expansion forces x and y to have the same types.

(ii) Let A be given. We know that $\vdash \mathsf{I} : A{\to}A$. Therfore by (i) there exists an $\mathsf{I}' \twoheadrightarrow_{\boldsymbol{\beta\eta}} \mathsf{I}$ such that $\mathtt{pt}(\mathsf{I}') = A{\to}A$. Then take $M \equiv \mathsf{I}'x$. We have $\mathtt{pp}(\mathsf{I}'x) = (\{x{:}A\}, A)$. ■

Complexity

The space and time complexity of finding a type for a typable term is exponential, see exercise 2.5.18.

In order to decide whether for two typed terms $M, N \in \Lambda_{\to}(A)$ one has

$$M =_{\beta\eta} N,$$

one can normalize both terms and see whether the results are syntactically equal (up to α-conversion). In exercise 2.5.17 it will be shown that the time and space costs of doing this is at least hyper-exponential (in the size of MN). The reason is that the type-free application of Church numerals

$$\mathbf{c}_n\mathbf{c}_m = \mathbf{c}_{m^n}$$

can be typed, even when applied iteratively

$$\mathbf{c}_{n_1}\mathbf{c}_{n_2}\ldots\mathbf{c}_{n_k}.$$

In exercise 2.5.16 it is shown that the costs are also at most hyper-exponential. The reason is that Turing's proof of normalization for terms in $\lambda_{\to}$ uses a succesive development of redexes of 'highest' type. Now the length of each such development depends exponentially on the length of the term, whereas the length of a term increases at most quadratically at each reduction step. The result even holds for typable terms $M, N \in \Lambda_{\to\mathrm{Cu}}(A)$, as the cost of finding types only ads a simple exponential to the cost.

One may wonder whether there is not a more efficient way to decide $M =_{\beta\eta} N$, for example by using memory for the reduction of the terms, rather than a pure reduction strategy that only depends on the state of the term reduced so far. The sharpest question is whether there is any Turing computable method, that has a better complexity class. In Statman [1979] it is shown that this is not the case, by showing that every elementary time bounded Turing machine computation can be coded as a a convertibility problem for terms of some type in $\lambda^o_{\to}$. A shorter proof of this result can be found in Mairson [1992].