2

Assigning types to terms

The topic of this book is one of the simplest current type-theories. It was called TA in the Introduction but in fact it comes in two forms, TA_C for combinatory logic and TA_λ for λ-calculus. Since most readers probably know λ-calculus better than combinatory logic, only TA_λ will be described here. (The reader who wishes to see an outline of TA_C can find one in HS 86 Ch.14; most of its properties are parallel to those of TA_λ.)

The present chapter consists of a definition and description of TA_λ. It is close to the treatment in HS 86 Ch. 15 but differs in some technical details.

2A The system TA_λ

2A1 Definition (Types) An infinite sequence of ***type-variables*** is assumed to be given, distinct from the term-variables. ***Types*** are linguistic expressions defined thus:

(i) each type-variable is a type (called an ***atom***);
(ii) if σ and τ are types then $(\sigma\to\tau)$ is a type (called a ***composite type***).

2A1.1 *Notation* ***Type-variables*** are denoted by "a", "b", "c", "d", "e", "f", "g", with or without number-subscripts, and distinct letters denote distinct variables unless otherwise stated.

Arbitrary types are denoted by lower-case Greek letters except "λ".

Parentheses will often (but not always) be omitted from types, and the reader should restore omitted ones in such a way that, for example,

$$\rho\to\sigma\to\tau \quad\equiv\quad (\rho\to(\sigma\to\tau)).$$

This restoration rule is called ***association to the right.***[1]

2A1.2 ***Informal interpretation*** To interpret types we think of each type-variable as a set and $\sigma\to\tau$ as a set of functions from σ into τ. The precise nature of this set of functions (all functions, all functions definable in some given system, etc.) will depend on the particular interpretation we may have in mind.

[1] It is the opposite of the rule for terms! The reason originates in the fact that terms follow the common notation convention in which an operator's input is written on the right, but in a type $\sigma\to\tau$ the type of the input is on the left of that of the output.

2A2 Definition The total number of occurrences of type-variables in a type τ will be called $|\tau|$ or the ***length*** of τ; more precisely, define

$$|a| \equiv 1, \qquad |\rho{\to}\sigma| \equiv |\rho| + |\sigma|.$$

The number of distinct type-variables occurring in τ will be called

$$\|\tau\|$$

and the set of all these variables will be called

$$Vars(\tau).$$

2A2.1 *Example* If $\tau \equiv (a{\to}b{\to}c){\to}(a{\to}b){\to}a{\to}c$, then

$$|\tau| = 7, \qquad \|\tau\| = 3, \qquad Vars(\tau) = \{a,b,c\}.$$

The structure of an arbitrary type is analysed in detail in 9D–E. The lemmas there will be used in some later chapters but not in this one.

2A3 Discussion (The Church and Curry approaches) In current use there are two main ways of introducing types into λ-calculus, one attributable to Alonzo Church and the other to Haskell Curry.

The former goes back to a type-system introduced in Church 1940. In it, the definition of "λ-term" is restricted by giving each term a unique type as part of its structure and saying that an application PQ is only defined when P has a function-type $\sigma{\to}\tau$ and Q the appropriate argument-type σ.[1]

The effect of Church's restriction can be seen on $\lambda x{\cdot}xx$, which is a well-formed term in type-free λ-calculus but represents the abstract concept of self-application, a concept whose meaningfulness may well be questioned. Self-application was involved in most of the paradoxes that were discovered in mathematics in the early 1900's, and Bertrand Russell devised the first of all type-theories specifically as a language in which these paradoxes could not be expressed. In Church's typed λ-calculus each variable has a unique type, so if x has a function-type $\sigma{\to}\tau$ it cannot also have type σ, and so the application xx cannot be defined as a typed term. Hence also $\lambda x{\cdot}xx$ cannot be a typed term.

Curry took a different approach. He pointed out that if we wish to ask questions about the meaningfulness of $\lambda x{\cdot}xx$, then we need a language in which these questions can be expressed. And Church's type-theory by itself is not adequate for this, because we have just seen that $\lambda x{\cdot}xx$ is excluded from it. Curry proposed a language which would include all the type-free λ-terms, and a type-theory which would contain rules assigning types to some of these terms but not to others. The term $\lambda x{\cdot}xx$ would not be given a type by these rules, but would still remain in the system and hence be discussable. (Curry and Feys 1958 §0B, p.5.)

Along with this change Curry proposed another, which is best understood by looking at the identity-combinator $\lambda x{\cdot}x$ as an example.

In Church's type-theory there is no term $\lambda x{\cdot}x$. Instead, for each type σ there is

[1] For a definition of typed λ-term and a few examples see HS 86 §13A; for another version, with motivation and more details, see Barendregt 1992 §3.2.

a variable x^σ with type σ and a term $\lambda x^\sigma \cdot x^\sigma$ with type $\sigma \to \sigma$. Informally, this term denotes the identity function on whatever set S may be denoted by σ. Call this function I_S; the only objects it accepts as inputs are members of S, and $I_S(x) = x$ for all $x \in S$. Thus Church's theory has an infinite number of identity functions, one for each set S. This agrees with the view of functions taken by most mathematicians: each function is seen as a set of ordered pairs with a domain and range built into its definition, and the identity functions I_S and I_T on two distinct sets S and T are viewed as different functions.

But this view is not entirely satisfying; an alternative and perhaps more natural view is to see all the separate identity-functions I_S, I_T, etc. as special cases of one intuitive concept, the operation of doing nothing. If we admit that such a concept exists, even though only in an imprecise sense, then a type-theory that tries to make it precise by splitting it into an infinite number of different special cases at the beginning will seem at the very best inefficient.

Curry's aim was a type-theory in which the identity-concept would be expressed by just one term $\lambda x \cdot x$, to which an infinite number of types would be assigned by suitable formal rules. Types would contain variables, and if a term M received a type τ it would also receive all substitution-instances of τ. This kind of theory will be called here a ***type-assignment*** theory or a ***Curry-style*** type-theory. (It is the ancestor of polymorphic type-theories.) In contrast, a theory in which each term has a unique built-in type will be called a ***typed-term*** theory or a ***Church-style*** theory.[1]

TA_λ will be a type-assignment theory.

2A4 Definition A ***type-assignment*** is any expression

$$M : \tau$$

where M is a λ-term and τ is a type; we call M its ***subject*** and τ its ***predicate***.

("$M{:}\tau$" should be read informally as "*assign to M the type* τ" or "*M has type* τ" or "*M denotes a member of whatever set* τ *denotes*".)

2A5 Definition A ***type-context*** Γ is any finite, perhaps empty, set of type-assignments

$$\Gamma = \{x_1 : \rho_1, \ \ldots, \ x_m : \rho_m\}$$

whose subjects are term-variables and which is ***monovalent*** or ***consistent*** in the sense that no variable is the subject of more than one assignment. For any such Γ define

$$Subjects(\Gamma) = \{x_1, \ldots, x_m\}.$$

[1] Curry's and Church's lines of thought were not really as distinct as the above seems to imply. In particular Church did not ignore the possibility that a single identity-concept might be formalizable instead of a multitude of particular identity-functions. Indeed his first systems of λ-calculus in the 1930's were part of an attempt to formalize exactly this single-identity view of functions in a type-free theory, and one of the best available expositions of this view is in the introduction to his book Church 1941. Only after his attempt to do this in an extremely general setting proved inconsistent did Church turn to type-theory and a more restricted approach to functions. Also Curry's type-theories began their development in some of his earliest work and were not simply a response to Church's; see Curry 1934.

2A5.1 *Notation* The result of removing from Γ the assignment whose subject is x (if Γ has one) is called

$$\Gamma - x.$$

(If $x \notin Subjects(\Gamma)$ we define $\Gamma - x = \Gamma$.) The result of removing from Γ all assignments $x_i{:}\rho_i$ with $x_i \notin FV(M)$ (where M is a given term) is called

$$\Gamma \restriction M$$

or "Γ *restricted to* M". And Γ is called an "***M-context***" (for a given M) iff

$$Subjects(\Gamma) = FV(M).$$

2A5.2 *Note* A type-context Γ is a set, not a sequence. Hence it does not change when its members are permuted or repeated. To implement TA_λ as a practical system we would have to represent Γ by an expression in some language and include rewrite-rules to permute Γ's members and make and remove repetitions. Such rules would obscure the main themes of this book so they have been avoided here by simply assuming that contexts are sets.[1]

2A6 Definition We say Γ_1 ***is consistent with*** Γ_2 iff $\Gamma_1 \cup \Gamma_2$ is consistent; and $\Gamma_1, \ldots, \Gamma_n$ are ***mutually consistent*** iff their union is consistent.

2A7 Definition (TA_λ-formulae) For any Γ, M and τ the triple $\langle \Gamma, M, \tau \rangle$ is called a ***TA_λ-formula*** and is written as

$$\Gamma \quad \mapsto \quad M{:}\tau$$

(or just $\mapsto M{:}\tau$ when Γ is empty). We shall call M the ***subject*** of this formula and τ its ***predicate*** (despite the fact that in general it contains other subjects and predicates too, namely those of the assignments in Γ).

2A7.1 *Notation* The following abbreviations will often be used:

$$x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n \ \vdash \ M{:}\tau \qquad \textit{for} \qquad \{x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n\} \ \vdash \ M{:}\tau,$$

$$\Gamma, y_1{:}\sigma_1, \ldots, y_n{:}\sigma_n \ \vdash \ M{:}\tau \qquad \textit{for} \qquad \Gamma \cup \{y_1{:}\sigma_1, \ldots, y_n{:}\sigma_n\} \ \vdash \ M{:}\tau.$$

2A8 Definition (The system TA_λ) TA_λ has an infinite set of axioms and two deduction-rules (called $(\to E)$ or $\to$-*elimination* and $(\to I)$ or $\to$-*introduction*), as follows.

Axioms of TA_λ: for every term-variable x and every type τ, TA_λ has an axiom

$$x{:}\tau \quad \mapsto \quad x{:}\tau.$$

[1] Type-contexts are also called *environments* in the literature. They play a different role from the sets called *bases* in HS 86 Chs.14–15: there a basis was a set of axioms for a theory, whereas here a context will be used as a set of assumptions for a particular deduction in a theory.

Deduction-rules of TA_λ:

$$(\to\text{E})\quad \frac{\Gamma_1 \mapsto P:(\sigma\to\tau)\quad \Gamma_2 \mapsto Q:\sigma}{\Gamma_1 \cup \Gamma_2 \mapsto (PQ):\tau,} \qquad [\textit{if } \Gamma_1 \cup \Gamma_2 \textit{ is consistent}]$$

$$(\to\text{I})\quad \frac{\Gamma \mapsto P:\tau}{\Gamma - x \mapsto (\lambda x.P):(\sigma\to\tau).} \qquad [\textit{if } \Gamma \textit{ is consistent with } x:\sigma]$$

A **TA_λ-*deduction*** Δ is a tree of TA_λ-formulae, those at the tops of branches being axioms and those below being deduced from those immediately above them by a rule. (A detailed definition of such deductions is given in 9C1.) The bottom formula in Δ is called its ***conclusion***; if it is

$$\Gamma \mapsto M:\tau$$

we call Δ a ***deduction of*** $\Gamma \mapsto M:\tau$ or a ***deduction of*** $M:\tau$ ***from*** Γ, and say that $\Gamma \mapsto M:\tau$ is ***TA_λ-deductible***. In the special case $\Gamma = \emptyset$, Δ may be called a ***proof*** of the assignment $M:\tau$.

2A8.1 *Note* (Rule $(\to\text{I})$) The condition in $(\to\text{I})$ that Γ be consistent with $x:\sigma$ means that either Γ contains $x:\sigma$ or Γ contains no assignment at all whose subject is x. In the first case the rule is said to ***discharge*** or ***cancel*** x from Γ. In the second case it is said to ***discharge*** x ***vacuously***.

In these two cases the rule takes two slightly different forms which may be displayed as follows (using "Γ_1" below to correspond to "$\Gamma - x$" above).

$$(\to\text{I})_{\text{main}}\quad \frac{\Gamma_1, x:\sigma \mapsto P:\tau}{\Gamma_1 \mapsto (\lambda x.P):(\sigma\to\tau),} \qquad [\textit{if } x \notin \textit{Subjects}(\Gamma_1)]$$

$$(\to\text{I})_{\text{vac}}\quad \frac{\Gamma_1 \mapsto P:\tau}{\Gamma_1 \mapsto (\lambda x.P):(\sigma\to\tau).} \qquad [\textit{if } x \notin \textit{Subjects}(\Gamma_1)]$$

2A8.2 *Example* Let $\mathbf{B} \equiv \lambda xyz\cdot x(yz)$ as in the list in 1A10.1; the following is a deduction of

$$\mapsto \mathbf{B}:(a\to b)\to(c\to a)\to c\to b.$$

(In it, "Γ" will denote the set $\{x:a\to b,\ y:c\to a,\ z:c\}$.)

$$\dfrac{\dfrac{\dfrac{\dfrac{y:c\to a \mapsto y:c\to a \qquad z:c \mapsto z:c}{}(\to\text{E})}{x:a\to b \mapsto x:a\to b \qquad y:c\to a,\ z:c \mapsto yz:a}(\to\text{E})}{\Gamma \mapsto (x(yz)):b}(\to\text{I})_{\text{main}}}{\Gamma - z \mapsto (\lambda z\cdot x(yz)):c\to b}(\to\text{I})_{\text{main}}}{\Gamma - z - y \mapsto (\lambda yz\cdot x(yz)):(c\to a)\to c\to b}(\to\text{I})_{\text{main}}$$

$$\mapsto (\lambda xyz\cdot x(yz)):(a\to b)\to(c\to a)\to c\to b$$

2A8.3 *Example* Let $\mathbf{I} \equiv \lambda x{\cdot}x$; the following is a deduction of

$$\frac{x{:}a \;\mapsto\; x{:}a}{\mapsto\; (\lambda x{\cdot}x){:}a{\to}a}\,(\to\mathrm{I})_{\mathrm{main}}$$

$$\mapsto\; \mathbf{I}{:}a{\to}a.$$

2A8.4 *Example* Let $\mathbf{K} \equiv \lambda xy{\cdot}x$; the following is a deduction of

$$\cfrac{\cfrac{x{:}a \;\mapsto\; x{:}a}{x{:}a \;\mapsto\; (\lambda y{\cdot}x){:}b{\to}a}\,(\to\mathrm{I})_{\mathrm{vac}}}{\mapsto\; (\lambda xy{\cdot}x){:}a{\to}b{\to}a}\,(\to\mathrm{I})_{\mathrm{main}}$$

$$\mapsto\; \mathbf{K}{:}a{\to}b{\to}a.$$

2A8.5 *Example* The following is a deduction of

$$\cfrac{\cfrac{x{:}a{\to}a \;\mapsto\; x{:}a{\to}a}{\mapsto\; (\lambda x{\cdot}x){:}(a{\to}a){\to}a{\to}a}\,(\to\mathrm{I}) \qquad \cfrac{x{:}a \;\mapsto\; x{:}a}{\mapsto\; (\lambda x{\cdot}x){:}(a{\to}a){\to}a}\,(\to\mathrm{I})}{\mapsto\; (\lambda x{\cdot}x)(\lambda x{\cdot}x){:}a{\to}a}\,(\to\mathrm{E})$$

$$\mapsto\; \mathbf{II}{:}a{\to}a.$$

2A8.6 *Remark* (Self-application) The above example gave a type to a term involving self-application, namely **II**. This was done by giving a different type to each of the two occurrences of **I**, and to do this we had to give two different types to the one variable x; but there was no inconsistency problem when $(\to\mathrm{E})$ was applied because the two applications of $(\to\mathrm{I})$ above $(\to\mathrm{E})$ removed x from the contexts on the left of " $\mapsto$ ". Similarly it is possible to give types to several other self-applications in TA_λ, for example **KK** and **BB**.

This may seem surprising, in view of the claim in 2A3 that the original aim of a type-theory was to avoid self-application. But in fact the "dangerous" self-application to be avoided is not any one simple particular case like **II**, but the overall general concept of self-application as represented by the term $\lambda x.xx$. And $\lambda x.xx$ does not receive a type in TA_λ.

To see this, suppose there were a TA_λ-deduction of

$$\mapsto\; (\lambda x{\cdot}xx){:}\tau$$

for some τ. Then its last step would have to be an application of $(\to\mathrm{I})$ to a deduction of

$$x : \rho \;\mapsto\; xx : \sigma$$

for some ρ and σ such that $\tau \equiv \rho{\to}\sigma$; and the last step in this deduction would have to be an application of $(\to\mathrm{E})$ to two deductions of

$$x : \sigma_1{\to}\sigma \;\mapsto\; x : \sigma_1{\to}\sigma, \qquad x : \sigma_1 \;\mapsto\; x : \sigma_1$$

for some σ_1. But $\sigma_1{\to}\sigma \not\equiv \sigma_1$ so the consistency condition in $(\to\mathrm{E})$ would be violated by these deductions.

Thus the consistency condition in (→E) prevents $\lambda x{\cdot}xx$ from having a type. This is in fact its main purpose.[1]

2A8.7 *Exercise** Deduce the following in TA_λ, where $\mathbf{B}' \equiv \lambda xyz{\cdot}y(xz)$, $\mathbf{C} \equiv \lambda xyz{\cdot}xzy$, $\mathbf{S} \equiv \lambda xyz{\cdot}xz(yz)$ and $\mathbf{W} \equiv \lambda xy{\cdot}xyy$ as in 1A10.1.

(i) $\mapsto \mathbf{B}' : (a{\to}b){\to}(b{\to}c){\to}a{\to}c$,
(ii) $\mapsto \mathbf{C} : (a{\to}b{\to}c){\to}b{\to}a{\to}c$,
(iii) $\mapsto \mathbf{S} : (a{\to}b{\to}c){\to}(a{\to}b){\to}a{\to}c$,
(iv) $\mapsto \mathbf{W} : (a{\to}a{\to}b){\to}a{\to}b$.

2A8.8 *Exercise** Deduce the following in TA_λ, where

$$P \equiv (\lambda vxyz{\cdot}v(y(vxz)))\mathbf{I}, \qquad Q \equiv \lambda xyz{\cdot}\mathbf{I}(y(\mathbf{I}xz));$$

(i) $\mapsto P : (a{\to}b){\to}(b{\to}a{\to}b){\to}a{\to}a{\to}b$;
(ii) $\mapsto Q : (a{\to}b){\to}(b{\to}c){\to}a{\to}c$.

2A8.9 *Note* (Comparison with HS 86) The format of TA_λ is what is known as the "***Natural Deduction***" style and was originated by Gerhard Gentzen in his thesis Gentzen 1935. The system called "TA_λ" in HS 86 §15B is another variant of the same style; its main differences from the above system TA_λ are as follows.

(i) In HS 86 the discharging of assumptions by rule (→I) was shown by enclosing the assumption in brackets at the top of the deduction-tree. But here the set of undischarged assumptions at each stage of the deduction is displayed on the left of the " ↦ " symbol and when rule (→I) is used this set is simply reduced. This notation is perhaps more explicit than that in HS 86 and is in common use in recent literature. In both notations deductions have the same tree-structure.

(ii) The version in HS 86 included an α-rule that is not in the present version. This was to ensure that the set of provable formulae would be closed under α-conversion even when the basis of axioms was not. But there are no axioms here in the sense of HS 86 so α-closure will turn out to be provable without adding an α-rule; see 2B6.

2A9 Definition Let Γ be a type-context. Iff there is a TA_λ-deduction of a formula $\Gamma' \mapsto M{:}\tau$ for some $\Gamma' \subseteq \Gamma$ we shall say

$$\Gamma \vdash_\lambda M{:}\tau.$$

In the special case $\Gamma = \emptyset$ we shall say ***M has type τ in*** $\mathbf{TA}_\lambda$, or ***τ is a type of M in*** $\mathbf{TA}_\lambda$, or

$$\vdash_\lambda M{:}\tau.$$

The phrase "in TA_λ" may be omitted when no confusion is likely.

2A9.1 *Lemma* (Weakening) $\Gamma \vdash_\lambda M{:}\tau,\ \Gamma^+ \supseteq \Gamma \implies \Gamma^+ \vdash_\lambda M{:}\tau$.

[1] There is at least one interesting type-theory in which this consistency condition is relaxed, the theory of intersection-types that originated in Coppo and Dezani 1978 and Sallé 1978. In this theory xx receives a type and types play a significantly more complex role than in TA_λ, see for example the comment in Hindley 1992 §1.1.

Proof Trivial from 2A9. □

2A9.2 *Warning* Do not confuse "$\mapsto$" with "$\vdash$". The former is part of the language of TA_λ and serves merely to separate two parts of a formula. But "$\vdash_\lambda$" is part of the meta-language in which TA_λ is described, and asserts the existence of a TA_λ-deduction; it is the traditional deducibility symbol.[1] In particular do not confuse the two statements

(a) *the formula* $\Gamma \mapsto M{:}\tau$ *is deducible*,
(b) $\Gamma \vdash_\lambda M{:}\tau$.

Statement (b) has the weakening property, as we have just seen in 2A9.1. But (a) does not; the rules of TA_λ have been formulated so that if (a) holds then the subjects of Γ coincide exactly with the free variables of M, and we cannot modify a deduction of $\Gamma \mapsto M{:}\tau$ to make a deduction of $\Gamma^+ \mapsto M{:}\tau$ if $\Gamma^+ \supseteq \Gamma$.[2]

The next two lemmas express the above comments more formally.

2A10 Lemma *If* $\Gamma \mapsto M{:}\tau$ *is deducible in* TA_λ *then Subjects* $(\Gamma) = FV(M)$.

Proof By an easy induction on lengths of deductions. □

2A11 Lemma (i) $\Gamma \vdash_\lambda M{:}\tau$ *iff Subjects*$(\Gamma) \supseteq FV(M)$ *and there exists a* TA_λ*-deduction of the formula* $\Gamma \restriction M \mapsto M{:}\tau$.

(ii) $(\exists\Gamma)(\Gamma \vdash_\lambda M{:}\tau) \iff (\exists\Gamma)\{\Gamma$ *is an M-context and* $\Gamma \vdash_\lambda M{:}\tau\}$.

(iii) *For closed terms M,*

$$(\exists\Gamma)(\Gamma \vdash_\lambda M{:}\tau) \iff \vdash_\lambda M{:}\tau.$$

Proof By 2A9 and 2A10. (The definition of $\Gamma \restriction M$ is in 2A5.1.) □

2A12 Historical Comment Although this book focuses on λ-calculus, type-assignment in fact began in its sister-theory, combinatory logic. The first systems appeared in Curry 1934 and were developed further in Curry and Feys 1958 Chs. 8–10, Seldin 1968 and Curry et al. 1972 Ch. 14, though in his earlier work Curry was aiming at building type-theories with the greatest possible generality and strength and simple type-assignment formed only a small part of each of these. However, when he found his strongest system inconsistent in 1954 he turned to the study of weaker ones and gradually realized that their $\rightarrow$-fragment formed a very neat core system that was worth studying on its own and stood a good chance of having some practical value.

This system was essentially a combinatory-logic analogue of TA_λ: Curry called it "*modified basic functionality*" though later in HS86 it was called "TA_C". Its basic properties first appeared in Curry and Feys 1958 §§8C and 9A–F, though even then those authors' main interest was in a slightly stronger system obtained by adding to

[1] Many works use "$\vdash$" where this book uses "$\mapsto$", and introduce no special notation for deducibility.
[2] Many other versions of Natural Deduction in the literature do not have this restriction but I believe its use slightly simplifies the proofs of some properties that depend on analysing the structure of a deduction.

TA_C an equality-invariance rule (see Chapter 4 below). The first papers to feature TA_C exclusively were Curry 1969 and Hindley 1969.

As for TA_λ itself, its first appearance was in Curry and Feys 1958 §9F1 under the name "$F_1(\lambda)^T$", though none of its properties were stated at that date other than a theorem relating it to TA_C. Its properties as an independent system were not described until ten years later in the theses Morris 1968 and Seldin 1968. Seldin's results on TA_λ appeared in Curry et al. 1972 §14D, but Morris' thesis was never published. The next study devoted to TA_λ was another unpublished thesis, Ben-Yelles 1979; some of its material will be the subject of Chapter 8 below.

For subsequent work on TA_λ see the references in Chapters 3–9 below.

2B The subject-construction theorem

Deductions in TA_λ have one very important property that is not shared by deductions in many more complex type-theories; the tree-structure of a deduction of $\Gamma \mapsto M{:}\tau$ follows the tree-structure of M exactly. To make this correspondence precise we really need the detailed definition of construction-tree of a term given in 9A4 and that of a deduction given in 9C1; but the following example gives a very good idea of what it means.

2B1 Example Let $\mathbf{B} \equiv \lambda xyz \cdot x(yz)$. A deduction of a type-assignment for **B** was shown in 2A8.2. If all but the subject is erased from each formula in this deduction the result is the tree shown in Fig. 2B1a. This is the same as the construction-tree of **B** (with certain details called position-labels omitted, for these see the full definition of construction-tree in 9A4).

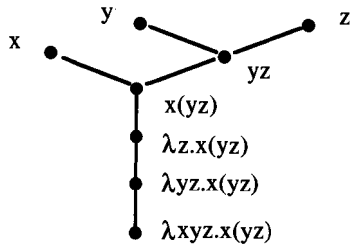

Fig. 2B1a.

The following theorem describes the deductions-to-terms correspondence formally.

2B2 Subject-construction Theorem (Seldin 1968 §3D Thm. 1, Curry et al. 1972 §14D Thm. 1.) *Let* Δ *be a* TA_λ*-deduction of a formula* $\Gamma \mapsto M{:}\tau$.

(i) *If we remove from each formula in* Δ *everything except its subject,* Δ *changes to a tree of terms which is exactly the construction-tree for* M.

(ii) *If M is an atom, say $M \equiv x$, then $\Gamma = \{x{:}\tau\}$ and Δ contains only one formula, namely the axiom*

$$x{:}\tau \;\mapsto\; x{:}\tau.$$

(iii) *If $M \equiv PQ$ the last step in Δ must be an application of $(\to\mathrm{E})$ to two formulae with form*

$$\Gamma \upharpoonright P \;\mapsto\; P{:}\sigma\to\tau, \qquad \Gamma \upharpoonright Q \;\mapsto\; Q{:}\sigma,$$

for some σ.

(iv) *If $M \equiv \lambda x{\cdot}P$ then τ must have form $\rho\to\sigma$; further, if $x \in FV(P)$ the last step in Δ must be an application of $(\to\mathrm{I})_{\mathrm{main}}$ to*

$$\Gamma, x{:}\rho \;\mapsto\; P{:}\sigma,$$

and if $x \notin FV(P)$ the last step in Δ must be an application of $(\to\mathrm{I})_{\mathrm{vac}}$ to

$$\Gamma \;\mapsto\; P{:}\sigma.$$

Proof Induction on $|M|$. Parts (i)–(iii) follow immediately from the full definition of deduction in 9C1. For (iv): if $M \equiv \lambda x{\cdot}P$ then by 2A8.1 the last step in Δ must have one of the forms

$$(\to\mathrm{I})_{\mathrm{main}}\ \frac{\Gamma, x{:}\rho \;\mapsto\; P{:}\sigma}{\Gamma \;\mapsto\; \lambda x{\cdot}P{:}\rho\to\sigma}, \qquad (\to\mathrm{I})_{\mathrm{vac}}\ \frac{\Gamma \;\mapsto\; P{:}\sigma}{\Gamma \;\mapsto\; \lambda x{\cdot}P{:}\rho\to\sigma}$$

and by 2A10 $(\to\mathrm{I})_{\mathrm{main}}$ is used when $x \in FV(P)$ and $(\to\mathrm{I})_{\mathrm{vac}}$ is used otherwise. Hence result. □

2B2.1 *Warning* (Deductions are not unique) Given M, let Δ be a deduction of $\Gamma \mapsto M{:}\tau$. By the subject-construction theorem the structure of M determines both the tree-structure of Δ and the terms at all the nodes in Δ. But this does not mean that the whole of Δ is completely determined by its conclusion, because there is some freedom of choice of the types assigned to terms at non-bottom nodes in Δ. For example, let $\Gamma = \emptyset$ and

$$M \;\equiv\; (\lambda xy{\cdot}y)(\lambda z{\cdot}z), \qquad \tau \;\equiv\; a\to a\,,$$

and consider the deduction in Fig. 2B2.1a; the type σ in that figure can be arbitrary.

$$\dfrac{\dfrac{\dfrac{y{:}a \;\mapsto\; y{:}a}{\mapsto\; (\lambda xy{\cdot}y){:}a\to a}\,(\to\mathrm{I}) \qquad \dfrac{z{:}\sigma \;\mapsto\; z{:}\sigma}{\mapsto\; (\lambda z{\cdot}z){:}\sigma\to\sigma}\,(\to\mathrm{I})}{\mapsto\; (\lambda xy{\cdot}y){:}(\sigma\to\sigma)\to a\to a}\,(\to\mathrm{E})}{\mapsto\; (\lambda xy{\cdot}y)(\lambda z{\cdot}z){:}a\to a.}$$

Fig. 2B2.1a.

However, if M is a normal form or a λI-term this freedom will disappear and Δ will be completely determined by M, as we shall see in the next lemma and the exercise below it.

2B3 Lemma (Uniqueness of deductions for nf's) (Ben-Yelles 1979 Cor. 3.2.) *Let M be a β-nf and Δ a TA_λ-deduction of $\Gamma \mapsto M{:}\tau$. Then*

(i) *every type in Δ has an occurrence in τ or in a type in Γ,*
(ii) *Δ is unique, i.e. if Δ' is also a deduction of $\Gamma \mapsto M{:}\tau$ then $\Delta' \equiv \Delta$.*

Proof Use induction on $|M|$. The cases $M \equiv y$ and $M \equiv \lambda x{\cdot}P$ are easy. Since M is a β-nf, by 1B10 the only other possible case is

$$M \equiv yP_1 \ldots P_n \qquad (n \geq 1).$$

In this case any deduction Δ of $\Gamma \mapsto M{:}\tau$ must contain an axiom

$$y : (\rho_1 \to \ldots \to \rho_n \to \tau) \mapsto y : (\rho_1 \to \ldots \to \rho_n \to \tau),$$

as well as n deductions $\Delta_1, \ldots, \Delta_n$ giving

$$\Gamma_1 \mapsto P_1{:}\rho_1, \ \ldots, \Gamma_n \mapsto P_n{:}\rho_n$$

followed by n applications of $(\to\mathrm{E})$ to deduce

$$\{y : (\rho_1 \to \ldots \to \rho_n \to \tau)\} \cup \Gamma_1 \cup \ldots \cup \Gamma_n \mapsto (yP_1 \ldots P_n) : \tau.$$

And Γ must be

$$\{y : (\rho_1 \to \ldots \to \rho_n \to \tau)\} \cup \Gamma_1 \cup \ldots \cup \Gamma_n.$$

To prove (i): by part (i) of the induction hypothesis every type in a Δ_i occurs in ρ_i or Γ_i and hence in Γ; also the type of y occurs in Γ. Hence (i) holds.

To prove (ii): the argument above shows that Δ' must use the same rules at the same positions as in Δ. And the type assigned to y in Δ' is determined by Γ and the assumption that type-contexts are consistent; then the types of $P_1, \ldots, P_n$ are determined by the type of y. □

2B3.1 *Note* (Subformula property) Part (i) of 2B3 corresponds to what is usually called in logic the ***subformula property;*** this says that in a Natural Deduction system every formula in an irreducible deduction occurs in either the conclusion or an undischarged assumption. (The correspondence between types and propositional logic will be fully described in Chapter 6.)

In contrast the TA_λ-deduction in Fig. 2B2.1a contains a type σ that does not occur in an undischarged assumption or the conclusion.

2B3.2 *Exercise** (Uniqueness of deductions for λI-terms) Show that if Δ is a TA_λ-deduction of $\Gamma \mapsto M{:}\tau$ and M is a λI-term, then Δ is unique; i.e. if Δ' is also a TA_λ-deduction of $\Gamma \mapsto M{:}\tau$ then $\Delta' \equiv \Delta$. (Hint (Thierry Coquand): use 2B3 and some facts from 2C and 2D below, and the leftmost-reduction theorem (1B9), plus some thought on the form that a leftmost reduction must have; see the Answers for details.)

The following three lemmas will be needed in the next section. The first is a special case of the third but is stated separately because it is needed in the proof of the third.

2B4 First Substitution Lemma for Deductions *Let* $\Gamma \vdash_\lambda M:\tau$ *and let* $[y/x]\Gamma$ *be the result of substituting* y *for a term-variable* x *in* Γ. *If either of the following holds:*

(i) $y \notin Subjects(\Gamma)$,

(ii) y *and* x *receive the same type in* Γ,

then

$$[y/x]\Gamma \vdash_\lambda ([y/x]M) : \tau.$$

Proof First, in both cases (i) and (ii) $[y/x]\Gamma$ satisfies the consistency condition for contexts. Next, by 2A11 there is a deduction of

$$\Gamma^- \mapsto M : \tau$$

for some $\Gamma^- \subseteq \Gamma$ with $Subjects\ (\Gamma^-) = FV(M)$. Then $[y/x]\Gamma^-$ is consistent. An induction on $|M|$ then shows that

$$[y/x]\Gamma^- \vdash_\lambda ([y/x]M) : \tau.^1$$

Then the weakening lemma (2A9.1) gives the result. □

2B5 α-Invariance Lemma *If* $\Gamma \vdash_\lambda P:\tau$ *and* $P \equiv_\alpha Q$ *then* $\Gamma \vdash_\lambda Q:\tau$.

Proof [Depends on Section 9C] It is enough to prove the result for one change of bound variable, say the replacement of a component $\underline{\lambda x \cdot M}$ of P by $\lambda y \cdot [y/x]M$ with $y \notin FV(M)$. If $\underline{\lambda x \cdot M} \equiv \underline{P}$ the result follows using 2B4. If $\underline{\lambda x \cdot M}$ is a proper part of $\underline{P}$, use 9C5 (a replacement lemma). □

2B6 Second Substitution Lemma for Deductions *Let* Γ_1 *be consistent with* Γ_2 *and let*

$$\Gamma_1, x:\sigma \vdash_\lambda M:\tau, \qquad \Gamma_2 \vdash_\lambda N:\sigma$$

Then

$$\Gamma_1 \cup \Gamma_2 \vdash_\lambda [N/x]M : \tau.$$

Proof Assume $x \in FV(M)$. (If not, the result holds trivially.) By 2B5 we can assume no variable bound in M is free in xN. In this case $[N/x]M$ is simply the result of replacing each free x in M by N with no accompanying changes of bound variables. And by 2A11 we can assume that

$$\begin{aligned} Subjects(\Gamma_1) \cup \{x\} &= FV(M), \\ Subjects(\Gamma_2) &= FV(N). \end{aligned}$$

The result is then proved by a straightforward induction on $|M|$. □

[1] By the way, y might be bound in M. To deal with the case that $M \equiv \lambda y \cdot P$, it is necessary to use the induction hypothesis twice and use the fact that $|[z/y]P| = |P|$.

2C Subject reduction and expansion

Besides avoiding logical paradoxes another main purpose of type-theories is to avoid errors of mis-matching in programming. If a term P has a type τ we can think of P as being in some sense "safe". If P represents a stage in some computation which continues by $\beta\eta$-reducing P, we would like to know that all later stages in the computation are just as safe as P. The following theorem guarantees this.

2C1 Subject-reduction Theorem (Morris 1968 §4D Thm. 1, Seldin 1968 §3D Thm. 2.) *If* $\Gamma \vdash_\lambda P{:}\tau$ *and* $P \triangleright_{\beta\eta} Q$ *then*

$$\Gamma \vdash_\lambda Q{:}\tau.$$

Proof [Depends on lemmas in 9C] First, by 2A11 there exists a deduction Δ of $\Gamma^- \mapsto P{:}\tau$ for some $\Gamma^- \subseteq \Gamma$ with *Subjects* $(\Gamma^-) = FV(P)$. By 1C5.1, $FV(P) \supseteq FV(Q)$. Hence by 9C5 it is enough to prove the theorem when P is a β- or η-redex and Q is its contractum.

Case 1: $P \equiv (\lambda x M)N$, $Q \equiv [N/x]M$. If $x \in FV(M)$ then by the subject-construction theorem (2B2) the lower steps of Δ must have form

$$\dfrac{\dfrac{\Gamma_1, x{:}\sigma \mapsto M{:}\tau}{\Gamma_1 \mapsto (\lambda x{\cdot}M){:}(\sigma{\to}\tau)}(\to\mathrm{I})_{\mathrm{main}} \qquad \Gamma_2 \mapsto N{:}\sigma}{\Gamma_1 \cup \Gamma_2 \mapsto ((\lambda x{\cdot}M)N){:}\tau}(\to\mathrm{E})$$

where $\Gamma_1 \cup \Gamma_2 = \Gamma^-$. Then 2B6 applied to the deductions for M and N gives

$$\Gamma_1 \cup \Gamma_2 \vdash [N/x]M{:}\tau$$

If $x \notin FV(M)$ the proof is similar.

Case 2: $P \equiv \lambda x{\cdot}Mx$, $Q \equiv M$, $x \notin FV(M)$. Then $\tau \equiv \rho{\to}\sigma$ for some ρ and σ, and by 2B2 the last steps in Δ must have form

$$\dfrac{\dfrac{\Gamma^- \mapsto M{:}\rho{\to}\sigma \qquad x{:}\rho \mapsto x{:}\rho}{\Gamma^-, x{:}\rho \mapsto Mx{:}\sigma}(\to\mathrm{E})}{\Gamma^- \mapsto (\lambda x{\cdot}Mx){:}\rho{\to}\sigma.}(\to\mathrm{I})_{\mathrm{main}}$$

Thus Δ contains a deduction of $\Gamma^- \mapsto M{:}\tau$ as required. □

The subject-reduction theorem has a partial analogue for expansion as follows.

2C2 Subject-expansion Theorem *If* $\Gamma \vdash_\lambda Q{:}\tau$ *and* $P \triangleright_\beta Q$ *by non-duplicating and non-cancelling contractions, then*

$$\Gamma \vdash_\lambda P{:}\tau.$$

Proof Exercise. This theorem is a special case of Curry et al. 1972 p.315, §14D Thm. 3 (= Seldin 1968 §3D Thm. 3). □

2C2.1 *Corollary* *If P is a closed BCI λ-term and $P \triangleright_\beta Q$ then*

$$\Gamma \vdash_\lambda P{:}\tau \iff \Gamma \vdash_\lambda Q{:}\tau$$

Proof For "$\Leftarrow$" use 1D6 and 2C2; for "$\Rightarrow$" use 2C1. □

The subject-expansion theorem can be extended to some cancelling contractions under suitable restrictions. (For example see Curry et al. 1972 §14D Thm. 3 or Hindley 1989 Thm. 3.3.) But it cannot be extended to arbitrary contractions, as the following examples show.

2C2.2 *Example* *$P \triangleright_{1\beta} Q$ by a cancelling contraction and Q has a type but P has no type*:

$$P \equiv (\lambda uv{\cdot}v)(\lambda x{\cdot}xx), \qquad Q \equiv \lambda v{\cdot}v.$$

We have $\vdash_\lambda Q{:}a{\to}a$ by 2A8.3. But no TA_λ-deduction has a conclusion with form $\mapsto P{:}\tau$. Because such a deduction would have to contain a deduction of $\mapsto (\lambda x{\cdot}xx){:}\sigma$ for some σ and this is impossible by 2A8.6.

2C2.3 *Example* *$P \triangleright_{1\beta} Q$ by a duplicating contraction and Q has a type but P has none*:

$$P \equiv (\lambda x{\cdot}xx)\mathbf{I}, \qquad Q \equiv \mathbf{II}.$$

We have $\vdash_\lambda Q{:}a{\to}a$ by 2A8.5. But P has no type because $\lambda x{\cdot}xx$ has none (by 2A8.6).

2C2.4 *Example* *$P \triangleright_{1\beta} Q$ by a cancellation, P and Q both have types, but Q has more types than P*:

$$P \equiv \lambda xyz{\cdot}(\lambda u{\cdot}y)(xz), \qquad Q \equiv \lambda xyz{\cdot}y.$$

It is easy to prove that

$$\vdash_\lambda P{:}(c{\to}d){\to}b{\to}c{\to}b, \qquad \vdash_\lambda Q{:}a{\to}b{\to}c{\to}b;$$

and an application of the principal-type algorithm (3E1) will show that the types possessed by P are exactly the substitution-instances of the one shown above, and similarly for Q. Hence P cannot have the type displayed for Q. (Roughly speaking, the underlying reason is that x has a function position in P and must therefore be assumed to have a function-type $c{\to}d$; since x does not occur at all in Q the type of Q has no such limitation.)

2C2.5 *Example* *$P \triangleright_{1\beta} Q$ by a duplication, P and Q both have types, but Q has more types than P*:

$$P \equiv (\lambda vxyz{\cdot}v(y(vxz)))\mathbf{I}, \qquad Q \equiv \lambda xyz{\cdot}\mathbf{I}(y(\mathbf{I}xz)).$$

By 2A8.8 we have

$$\vdash_\lambda P{:}(a{\to}b){\to}(b{\to}a{\to}b){\to}a{\to}a{\to}b,$$
$$\vdash_\lambda Q{:}(a{\to}b){\to}(b{\to}c){\to}a{\to}c;$$

and an application of the principal-type algorithm (3E1) will show that P cannot have the type displayed for Q. (The underlying reason is that the two v's in P must receive the same type whereas the two I's in Q are not so limited.)

2C2.6 *Example* P η-*contracts to* Q, P *and* Q *both have types, but* Q *has more types than* P:

$$P \equiv \lambda xy{\cdot}xy, \qquad Q \equiv \lambda y{\cdot}y.$$

It is easy to see that

$$\vdash_\lambda P{:}(a{\to}b){\to}a{\to}b, \qquad \vdash_\lambda Q{:}a{\to}a,$$

and that a TA_λ-deduction of $\mapsto P{:}a{\to}a$ is impossible (because x is in a function position in P).

2C3 Definition (Types(M)) If M is closed, define ***Types(M)*** to be the set of all τ such that $\vdash_\lambda M{:}\tau$.

We shall see in Chapter 3 that if *Types(M)* is not empty its members are exactly the substitution-instances of one type, the *principal type* of M; hence *Types(M)* is either empty or infinite.

2C3.1 *Lemma Let* P *be closed. Then*

(i) $P \rhd_\beta Q \implies Types(P) \subseteq Types(Q)$,
(ii) *if* $P \rhd_\beta Q$ *by a non-cancelling and non-duplicating reduction, then*

$$Types(P) = Types(Q).$$

Proof By 2C1 and 2C2. □

2C3.2 *Note* (Conversion-invariance) Examples 2C2.2–2C2.6 show that we do not always have

$$M =_\beta N \implies Types(M) = Types(N).$$

Even worse, we shall see an example in 7A2.1 where $M =_\beta N$ but

$$Types(M) \cap Types(N) = \emptyset.$$

Thus *Types(M)* is very definitely not invariant under conversion.

From a theoretical point of view this seems unsatisfactory. In fact, continuing the Church-versus-Curry discussion from 2A3, it must be admitted that conversion-sensitivity of *Types(M)* is the main disadvantage of a Curry-style type system. In a Church system the type of M does not change with conversion (because all terms are typed, including β-redexes, and this fact restricts the β-reduction rule and prevents type-changes). But when we move to a Curry-style system to get the extra expressive power provided by its polymorphism, we do not get it for free, and the price we pay is that *Types(M)* can change with conversion.

However, Chapter 4 will describe the effect of adding a new rule to TA_λ to overcome this defect, and it will give theoretical evidence to suggest that perhaps the price is not so high after all.

In practice too the conversion-sensitivity of *Types(M)* has turned out to be a very small problem. Indeed, if one views an assignment $M:\sigma\to\tau$ as saying that the application of M to every term with type σ is "safe" in some sense, then the most important practical property of a type-system is the subject-reduction theorem, which says that if M has type $\sigma\to\tau$ it will not lose this safety-feature during a reduction. If *Types(M)* happens to increase as M is reduced this is not a drawback but simply means that M is becoming safer. In particular, practical programming languages like ML and its relatives operate very successfully without conversion-invariance.

2D The typable terms

The system TA_λ divides the λ-terms in a natural way into two complementary classes: those which can receive types, such as $\lambda xyz\cdot x(yz)$, and those which cannot, such as $\lambda x\cdot xx$. The former may be regarded as "safe" in the sense that if a term has a type we know there is a way of assigning types to all its components that avoids mis-matches of types. The following is a precise definition of this class.

2D1 Definition A term M is called ***(TA_λ-) typable*** or ***stratified*** iff there exist Γ and τ such that

$$\Gamma \vdash_\lambda M:\tau.$$

2D2 Lemma *The class of all* TA_λ*-typable terms is closed under the following operations:*

(i) *taking subterms (i.e. all subterms of a typable term are typable);*
(ii) *$\beta\eta$-reduction;*
(iii) *non-cancelling and non-duplicating β-expansion;*
(iv) *λ-abstraction (i.e. if M is typable so is $\lambda x\cdot M$).*

Proof (i) by 2B2. (ii) by 2C1. (iii) by 2C2. (iv) by rule (→I). □

2D3 Theorem *The class of all* TA_λ*-typable terms is decidable; that is, there is an algorithm which decides whether a given term is typable in* TA_λ.

Proof The principal-type algorithm (3E1) will be a suitable decision-procedure. □

2D4 Remark (Normalization) A property that nearly every type-theory in the literature possesses is the ***weak normalization (WN)*** property, which says that every typable term can be reduced to a normal form. Many type-theories also have the ***strong normalization (SN)*** property, which says that all reductions of a typable term

are finite. Both WN and SN can be regarded as safety-features of the type-theory in question: if reductions are viewed as imitating the process of computing values, WN says that a computation can always be continued to a result if we wish and SN says that all computations terminate.

The next theorems state precisely the position for TA_λ.

2D5 Weak Normalization (WN) Theorem (Turing 1942, Curry and Feys 1958, etc.) *Every TA_λ-typable term has both a β-nf and a $\beta\eta$-nf.*

Proof See 5C1 and 5C1.1 for a proof (from Turing 1942), and 5C1.2 for historical notes. □

2D5.1 *Example* By 2D5 the fixed-point combinator **Y** in 1A10.1 is not typable in TA_λ; because by 1B9.1, **Y** has no β-nf.

2D6 Strong Normalization (SN) Theorem (Sanchis 1967, Diller 1968, etc.) *If M is a TA_λ-typable term, every $\beta\eta$-reduction that starts at M is finite.*

Proof There are many proofs in the literature besides those of Sanchis and Diller; for example HS 86 Appendix 2 contains an accessible one for β in Thm. A2.3 and one for $\beta\eta$ in Thm. A2.4. For references to some others see 5C2.2. □

2D6.1 *Note* Since SN implies WN there is no real need for a separate treatment of WN. But the Turing proof of WN in 5C1 is both simpler and older than any proof of SN. Further, most applications of normalization turn out to be of WN rather than SN. The following are a couple of such applications.

2D7 Theorem *There is a decision-procedure for β-equality of TA_λ-typable terms; i.e. an algorithm which, given any typable terms P and Q, will decide whether $P =_\beta Q$. Similarly for $\beta\eta$-equality.*

Proof Reduce P and Q to their β-nf's (which exist by WN, and can be found using leftmost reductions, by 1B9) and see whether they differ. □

2D7.1 *Note* The complexity of the above decision-procedure can be measured in terms of the Grzegorczyk hierarchy $\mathscr{E}^0$, $\mathscr{E}^1$, $\mathscr{E}^2$, ... of sets of primitive recursive functions (Grzegorczyk 1953 p.29): in fact Statman 1979b pp. 73–75 points out that the procedure can be programmed to operate on a Turing machine in $\mathscr{E}^4$ time but no decision-procedure for β-equality of typable terms can be made to operate in $\mathscr{E}^3$ time. (The members of $\mathscr{E}^3$ are known as *elementary functions*.)

2D8 Theorem *Every $BCK\lambda$-term (as defined in 1D2) is typable.*

Proof Hindley 1989 Thm. 4.1, depending on WN. □

2D8.1 *Note* The BCKλ-terms are terms without multiple occurrences of variables (except possibly for binding occurrences), so the above theorem connects untypability with multiple occurrences of variables. On the other hand not every term with multiple occurrences is untypable; consider $\mathbf{S} \equiv \lambda xyz{\cdot}xz(yz)$ in 2A8.7(iii) for example.