THE

LANGUAGE THEORY

OF

AUTOMATH

PROEFSCHRIFT
TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
IN DE TECHNISCHE WETENSCHAPPEN
AAN DE TECHNISCHE HOGESCHOOL EINDHOVEN,
OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF. IR. J. ERKELENS,
VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN DEKANEN
IN HET OPENBAAR TE VERDEDIGEN
OP VRIJDAG 15 FEBRUARI 1980 TE 16.00 UUR
DOOR

DIEDERIK TON VAN DAALEN

GEBOREN TE BERGEYK

DRUK WIBRO HELMOND

Dit proefschrift is goedgekeurd

door de promotoren

Prof.dr. N.G. de Bruijn

en

Prof.dr. W. Peremans

ACKNOWLEDGEMENTS

To all my former colleagues in the Automath project, for the fine cooperation. Especially to Bert Jutting, Rob Nederpelt and Roel de Vrijer, for their help during the last stage of the preparation of the manuscript.

To Marlène Beunis, for her hospitality.

To Lieke Janson, Janna Blotwijk and Adele Hendriks, for the typing, and to Franka van Neerven, for her kind assistance.

To Prof.Ir. W. Baarda, who so generously offered the facilities to finish the thesis.

To the Netherlands Organization for the Advancement of Pure Science (Z.W.O.), for financial support during my work in the Automath project.

CONTENTS

I INTRODUCTION AND SUMMARY

This thesis gives an account of the author's *language theoretical* studies on the *Automath languages*, during his work in the project *Mathematical Language AUTOMATH* (under supervision of Prof. De Bruijn) at the Eindhoven University of Technology. These studies can be considered as a continuation and completion to previously published work of Nederpelt [51] and De Vrijer [70].*)

Actually, an introduction to the remaining chapters of the thesis is hardly necessary because they are formally self-contained and provided with lengthy introductions themselves. However, we like to make some general remarks on the Automath project, hoping to clarify some points which have sometimes given rise to misunderstanding. Most views expressed are common in the Automath project, but some are personal views, not necessarily shared by other workers in the project.

We start with *preliminary remarks*, followed by a *survey* of the Automath project. We discuss the *language theory* and its role in the project. We give an informal introduction to the various *Automath languages* and explain how *mathematical reasoning* can be represented. Finally we *summarize* the contents of this thesis. Occasionally we make a comparison with related logical systems and related enterprises elsewhere. For more information on the subjects of this chapter we refer to De Bruijn [13,20], Jutting [37], Zucker [77] and Van Daalen [27].

I.1 Preliminary remarks

1.1 - Reliability and formal rigour

The Automath project originally arose (around 1966) from the idea that it was desirable to increase the dependability of pieces of mathematics by having them checked by a computer. To this end the mathematics involved was to be formalized in a *mathematical language* allowing computer verification.

First something about this part of the motivation. One might wonder whether greater dependability is desirable at all - and if so, in what parts of mathematics -, and whether formal rigour (as imposed by the

*) Numbers in brackets refer to items on the list of references.

computer) contributes at all to dependability. Critics sometimes argue that correctness of a mathematical text, or of a proof, after all depends on human insight in the situation and understanding of the concepts involved. And, consequently, they sometimes suggest that formal rigour can be opposed to reliability, because the presence of too many formal details may spoil the understanding.

There is, generally, some point in this criticism, but all the same, many mathematicians sometimes produce faulty proofs and, even, false theorems. This just means that they have been cheated by their intuition. Such mistakes cannot be said to be *caused by* lack of rigour but, rather, would have been *prevented* by being more rigorous. E.g. by formalizing the subject matter in a well-chosen formalism. In general, the possibility of computer verification plays a minor role here and, as De Bruijn puts it, the computer is just there to set the standards. Serious errors won't survive the process of formalization and will never be fed into the machine. However, after having taken the trouble to produce a "fully formal" proof with possibly lots of technical details it is nice to have a patient computer actually waiting to read it and relish the details. In particular, because on rereading, the details indeed may spoil one's own understanding.

Besides (this is our second point against the criticism), we think that the latter situation can be avoided by using a *good* formalism, which allows a formalization faithful to the informal ideas one had in mind (see also 1.4). *)

It has, of course, never been intended that computer verification might *replace* human understanding, and that formalization might cover all of mathematics. We just note that formalization sometimes can support our understanding and guide our intuition.

1.2 The "data bank" aspect

According to the above criticism one never can rely on results one does not fully understand. Such an orthodox point of view we think unsatisfactory: one sometimes wants to use what might be called "more or less black boxes", e.g. one sometimes wants to believe a theorem without knowing, or without quite understanding, its proof (e.g. one does not understand the proof *any more*).

---

*) Numbers not in brackets refer to sections in the present volume; if not starting with a Roman numeral they indicate sections inside the current chapter.

Here we touch a certain "data bank" aspect (as opposed to the checking aspect) of such a formalization project: the codification and storing of a large amount of dependable and unambigous mathematics.

1.3 The experimental character of the project

Thus far about the original motivation. The present author likes to consider the Automath project as an *experiment* in order to answer the question: can we develop formalisms (mathematical languages), in which mathematical texts *actually* can be formulated in such a way that mechanical verification (by a computer) is *actually* possible. Apart from the emphasis on computer verification there is another difference as compared with earlier formalization projects: it is required that both *writing* (i.e. translating mathematics into Automath) and *checking* are *practically feasible* (and it would be nice if it were *readable* too), and that the formalism is kind of *universal*, i.e. suitable for large parts of mathematics.

1.4 The correspondence with ordinary reasoning

In Automath it is attempted to achieve the feasibility of the writing stage by keeping as close as possible to ordinary informal mathematical reasoning, and to existing good mathematical habits. This then was to result in the possibility of a fully formal proof not blurring the understanding - compare a well-structured computer program -.

Keeping close to ordinary reasoning also serves the feasibility of the checking process: in principle we do not expect more from the machine than we would expect from a human checker - though of course we expect the machine to be much faster and more accurate than a human -. The feasibility of the checking requires that *all* of the reasoning is formalized in the language, whereas usual logical systems generally formalize only part of it and leave the rest to informal metalanguage. In particular we mention the handling of *proofs*, the handling of *variables* and the handling of *abbreviations* (i.e. the introduction of new *defined constants*, see 4.3).

1.5 The didactical aspect

A side effect of the analysis of mathematical reasoning needed for the development of a formalism meeting the above specifications might be a better insight into ways of presenting and teaching mathematics. This, *didactical*, aspect of Automath (beside the aforementioned *checking* and *storing* aspects) proves indeed to be important: Nederpelt and De Bruijn have used Automath-like systems to explain first-year mathematical students and mathematics teachers-to-be some principles of mathematical discipline. Research in this direction now falls under the WOT project ("Wiskundige Omgangs Taal", this is Dutch for: mathematical vernacular), which is going on in Eindhoven. One tries to codify elements of natural mathematical reasoning into a rather precise language which is inspired by Automath but does not particularly aim at computer verification.

1.6 The possible foundational contribution

From the modest statement of the aims of Automath, above, it will be clear that Automath has *no* strong *foundational claim* - in the usual logical sense - or philosophical position to defend like some of its forerunners. But if one wants to hear such a claim it might be the following one: that it is possible to present large parts of ordinary mathematics in Automath in a *natural* way. In particular that large parts of even *classical* reasoning fit quite well in the "*minimal logic*" of Automath (see 5.10) and that large parts of classical mathematics can be founded on the *typed λ-calculus* frame work of Automath (see 5.3) rather than on axiomatic set theory. (In fact this claim is a sine qua non to the Automath project.)

Besides, the original, simple wish to increase the reliability of mathematics can, from a practical point of view, also be considered as a foundational contribution.

1.7 The nature of Automath

A more ambitious, less carefull phrasing of the aim of Automath, viz. the development of a language in which *all* mathematics can be

expressed so meticulously that *syntactical* correctness would entail *mathematical* correctness, has sometimes given rise to confusion. Logicians then argued that such an enterprise was doomed to failure, firstly, because it would contradict the *incompleteness* theorems and, secondly, because it would contradict the *undecidability*: the computer certainly would not be able to check for correctness (to *decide*, as one says) any substantial part of mathematics.

We will explain that such criticism is hardly to the point. The *basic system* of Automath just covers a tiny part of mathematics, so to say *minimal predicate logic*. The Automath user himself has to add to this basic system all the axioms and constants necessary for his specific area of interest, and he has to supply more axioms and constants whenever he wants to increase the expressive power of his language or the strength of his theory. Further, the computer is certainly not supposed to decide the *truth* of the axioms, it is even not supposed to decide *derivability* from the axioms, but just verifies *derivations* (i.e. *proofs*).

1.8 Some proof checking systems

In the Automath project the computer is not expected to check (e.g. to prove) *theorems* but, rather, is expected to check whether something is a *proof* and whether it *proves* a certain theorem. Thus, the project can be compared with two other major *proof-checking* projects: the FOL (*First Order Logic*) project of Weyhrauch c.s. in Stanford [21, 73], and the LCF (*Logic Of Computable Functions*) of Milner c.s. in Edinburgh [32].

FOL is based on classical first order logic, in *natural deduction* style, and is intended to be universal like Automath. However, according to Bulnes [21], the system (still) has some difficulties in asping with *sorts* (or *types*) which seems to make the system less appropriate for parts of mathematics *not* based on classical set theory.

The kernel of LCF is a system called PPλ (*polymorphic predicate λ-calculus*) a system of typed λ-calculus plus fixed point induction plus logic, also in natural deduction style, based on Scott's work in the theory of computations. It is especially intended for problems concerning algorithms and programming languages.

In principle, these two systems are not more *interactive* than Automath, since in Automath as well line after line can be fed into the machine, thus incrementally constructing pieces of correct mathematics. However, recently both systems have been enriched by a strong heuristic mechanism allowing socalled *top-down* proof (i.e. working from the results backwards to the assumptions). In fact, by these mechanics, called GOAL (for FOL) and ML (for LCF) respectively, a kind of clever mixture between a proof-checker and a theorem-prover has been created (in fact the "top-down tacticals" are just *a part* of ML, which also contains some other useful mechanisms).

The basic elements of Automath just include what may be called "constructive reasoning", as borrowed from ordinary, informal, sound mathematical practice. Of these we mention the "linear" natural deduction system (see 4.5,p.23)used in the construction of both proofs and objects, the facility to abbreviate expressions by a new name (with parameters) at any desired moment (see 4.3, and the introduction to Ch. III), and the *suppression mechanism* for "fixed" parameters (see, e.g., [27 , sec. 2.15]). A consequence of the logical weakness of the basic system is the required universality: the Automath user is even free in the use of his *logical* axioms.

1.9 Proof checking vs. theorem proving

When constructing a proof-checking or theorem-proving system one has to decide how to devide the total amount of work between the human writer and the machine. In general it is assumed that easier writing makes more difficult checking and vice-versa. A distinctive principle of Automath languages always has been that the computer *actually* must be able to cope with its task. So, *at least*, the system the machine is supposed to decide must be formally *decidable*. In fact we want *feasible* decidability (cf. 2.10). On the other hand it is required that the writer's burden is as light as possible.

A nice point is that, in contrast with the above stated general view, easier writing sometimes makes checking easier too. Viz. if the system allows the writer to omit parts of the argumentation these parts, of course, do not need to be checked! But, on the other hand, a certain redundancy will help the machine to detect the, almost inevitable, minor errors at an early stage.

In view of feasible decidability general theorem proving is out of the question. But it *is* in the spirit of the Automath project to successively extend an existing, working, verification system with new tools that handle additional, feasible tasks. In such a way one might turn one's proof-checking system into a partial truth-checking (i.e. theorem proving) system, notably in well-defined restricted domains. Put differently, the machine might be allowed sometimes to *calculate* facts, rather than proving them. Although, if one would allow the user of the system to program such attached mechanisms himself, it would be preferable, if also a proof would be generated and checked (cf. 2.3).

In fact, the Automath proof-checking system has always contained such a partial truth-checker, viz. a decision procedure for the formulae (*definitional equations* and *typing formulae*) of the underlying typed λ-calculus (see 4.1).

1.10 Some characteristic features of Automath

We just mention here (but will come back to it) that the parallel natural deduction treatment of objects and proofs, which we think quite natural, and characteristic for Automath, gives rise to a *generalized typed λ-calculus*. By "generalized" we mean that the types are not given beforehand, but are rather constructed along with the terms and can have complicated form (cf. 4.1). In IV.1 there is given a further classification of such systems, into *pure*, *extended* and *arithmetical* systems. The pure systems have the ordinary λ-calculus operations only, the extended ones have additional logical operations, and the arithmetical systems have arithmetic built in in the form of a recursion operation. The pure and extended systems are the subject of this thesis.

The Automath languages AUT-68 and AUT-QE (4.5-4.7) belong to the pure, the language AUT-Pi (in Ch. VIII) belongs to the extended systems and there are *no arithmetical* Automath languages. This is a fundamental choice: the addition of a built-in recursor might give rise to definitional equations which are not feasibly decidable and, besides, we don't think that the presence of a recursion would make the representation of ordinary mathematical reasoning any easier. Consequently, the natural number structure is not built in, but has to be introduced axiomatically, just like any other mathematical structure. Needless to say that the

Church (or, any other) representation of numbers in λ-calculus does not come in.

1.11 Propositions as types

The parallelism between objects and proofs, types and propositions, definitional equality and proof theoretic conversion, for short: the *propositions-as-types* notion of construction, was first hinted at by Curry and Feys [25]. Later on it was developed further by Howard [34] and employed by him and other logicians (Scott[62], Prawitz [60], Martin-Löf [45], Girard [31]) in founding a theory of constructions, in proof theory, and in constructing an intuitionistic theory of types. In the meantime, it was independently discovered by De Bruijn (he also inspired Scott [62]) and used in the Automath project.

I.2 A survey of the Automath project

2.1 The AUT-QE stage

The experimental, practical character of the project clearly required: (i) the development of appropriate languages, (ii) the construction of programs for verifying these languages, (iii) the actual writing and checking of large pieces of mathematics.

There exists not just one Automath language, but a whole family of Automath languages. The first language (around 1968) which had the characteristic typed λ-calculus structure was AUT-68. Before 1968 there were just some sub-languages: LSP (see Ch. III) which codified the abbreviation device, PAL which already had type structure but still lacked λ-calculus (see [11]). Experience with AUT-68 led almost immediately to the construction of AUT-QE, which proved to be quite suitable for the then adopted propositions-as-types style of writing mathematics.

So the first language around which the project was centered was AUT-QE. De Bruijn's sketch of a verifying program was elaborated and implemented by Zandleven [75]. Jutting translated Landau's "Grundlagen der Analysis", and his translation was completely checked by the verifying program. This enterprise has been extensively documented in [37]. The Chapters V, VI of this thesis are mainly devoted to AUT-QE.

2.2 The AUT-Pi stage

It was always foreseen that, on the basis of the experience with AUT-QE, higher-level, easier-to-write, so called *super-languages* were to be developed, possibly for "special purposes", i.e. specific areas of mathematics. The second language playing a central role in the project was AUT-Pi, developed by Zucker.

This is indeed a kind of super-language extending AUT-QE in two respects. Firstly, the mathematical basis of AUT-Pi is somewhat stronger (it is an extended system, i.e. there is slightly more logic built in). This answered, e.g., in combination with the principle of *irrelevance of proofs* (see 5.2, and [20]) Jutting's need for easier embedding and "exbedding" facilities (see [37]). Secondly it contains some handy "syntactical features" which make life for the Automath user somewhat more comfortable. We mention the SYNT-facility for syntactical operations on expressions (which, i.a., allows to omit redundant parameters (but see 1.3)), and the presence of *strings* and *telescopes*. More about this can be found in [37, 77].

However, the use of these syntactical mechanisms is *not restricted* to AUT-Pi, they can as well be added to AUT-68 and AUT-QE. This seems to be particularly worthwile, because the strings-and-telescopes in some sense *duplicate* the pairs-and-products of AUT-Pi (see VIII.1.5).

Zucker (assisted by A. Kornaat) employed the new language for a modern, thoroughly classical (in the sense of "classical logic") treatise on the principles of real analysis, thus contributing to the foundational claim mentioned above. A survey of the AUT-Pi part of the project is to be found in [77].

A new verifying program was designed by Zandleven, developed by him and Kornaat, and is now being finished by Jutting. Apart from the fact that this new verifying program accepts AUT-Pi as well as the older languages, it also contains improved facilities for handling *bound variables* (see 3.4) and for *storage manipulation*. The latter proved necessary because with the first verificator, which left the handling of the extensive storage requirements to the computer system, working in interactive mode turned out to be cumbersome.

Apart from the two major Automath texts produced by Jutting, Zucker and Kornaat there have been formalized many smaller pieces of mathematics

into Automath by a variety of authors, mostly students. In Bulnes it has been suggested that the size and scope of the proof checking projects performed in FOL were comparable with size and scope of e.g. Jutting's opus. The present author disagrees: The amount of material handled in FOL is in no way comparable to what has been done in Automath.

2.3 The multi-level approach

The words "higher-level languages" suggest a separation between an object language, and a formal super-language which provides easier writing. Texts in the latter language may then be mechanically translated into object-language, which in turn is to be verified by the machine. In AUT-Pi, contrarily, there is, in principle, no such separation of levels: all the additional features are incorporated into the language. We write "in principle", because the SYNT-facility is indeed somewhat related to this multi-level approach.

There have also been certain proposals actually directed towards this multi-level framework. E.g. Wieringa (now working on the application of Automath to programming language theory), has once constructed a system that answers simple arithmetical questions (n * m = ?) and provides the resulting equation with a proof in AUT-QE. This AUT-QE proof turns out to be correct, of course! Similarly, there has been constructed a mechanism that decides propositional formulas and provides the true ones with an AUT-QE proof [53,74]. Compare also the discussion in 1.8 about partial theorem-proving mechanisms.

In FOL and LCF partial theorem provers and multi-level approach are present too. We mention the FOL procedure MONADIC, which decides formulas of monadic predicate calculus, and the ATTACH facility, allowing the machine to establish combinatorial facts by actual calculation. As for LCF, the meta-language ML is presented as a kind of programming language for manipulating the objects of the PPλ system.

2.4 The theoretical aspects

Of course the development task in the project, viz. of developing languages and verifying programs, and of writing mathematics in Automath,

also gave rise to theoretical studies. Here we distinguish:

(1) language theoretical studies,

(2) studies concerning the way mathematics is formalized in Automath.

This thesis deals with the language theory (1), which we define as the theory of the underlying typed λ-calculus of the Automath languages. Object of study is the syntactical structure consisting of the Automath *expressions*, provided with the relations *reduction*, *definitional equality* and the *typing relation* (or *typing function*). See

As regards (2), we mention some typical logical questions: what do we gain and loose by such formalizations, and: what is the relation between the Automath formalization and, say, some standard formulation of a piece of mathematics. Such questions are interesting, mostly because of the unconventional way in which mathematics is formulated in Automath. In particular, the fact that the *proofs* explicitly enter the Automath formalization is important. E.g. it allows detailed analysis of proofs, and of reasoning, and it gives rise to, as we say, *generalized logic* (see 5.10,[20] or [77]).

Then the studies (2) can, i.a., indicate what Automath language is suitable for what kind of mathematics. Roughly speaking, we might say that (2) concerns *semantical* questions, in contrast with the basically *syntactic* questions of the language theory, treated below.

2.5 What is language theory?

The results of the language theory are important for the construction of the verifying program and for proving its correctness. Further they serve as a foundation for the study of mathematics in Automath, i.e. the studies (2) mentioned above. E.g. the consistency of the underlying typed λ-calculus (as provided by *Church-Rosser theorems* and the like, see below) is clearly a prerequisite for the consistency of mathematics formalized in Automath.

Nevertheless, the language theory concerns the expressions and formulas as mere *syntactical* constructs, thus abstracting from possible mathematical content. Hence, the language theory also abstracts from particular sets of constants and axioms (socalled *books*) belonging to a particular piece of mathematics.

We take the point of view that the *languages* of the Automath family are characterized by their set of *correct* (i.e. well-formed according to the rules and restrictions of the various languages) *books*, *formulas* and *expressions*, rather than by a certain specific definition, i.e. a specific set of rules. Two definitions are said to be *equivalent* if they define the same language. One language is said to be an *extension* of another language if its set of correct expressions, books etc. contains the set of correct expressions, books etc. of the other one.

2.6 The aims of the language theory

Now we mention some typical theoretical aims. On the one hand, the design and comparison of language *definitions*, in particular the comparison of socalled E-*definitions*, which *generate* the language in question by a set of production rules, with the *algorithmic definitions* which describe the language by giving its *verifying* program.

On the other hand there is the comparison of the distinct languages, leading to *conservativity* and *unessential-* or *definitional extension* results (see V.3.3 for the terminology).

Last but not least we mention the *decidability* of the Automath languages, which is, in principle, essential for the aim of the project, mechanical proof-checking. The latter goal (to prove the decidability) consists of: (1) indicating a decision procedure, (2) proving its equivalence with a given language definition (these parts can be skipped if the language in question is given by a definition of the algorithmic type), (3) proving the termination of the indicated procedure.

2.7 Three desirable properties

The main tool of the language theory is the detailed study of the socalled *reduction relations* involved. Roughly speaking, reduction of expressions amounts to step by step *evaluating*, step by step transforming the expression (cf. 4.3), until possibly an *irreducible* (or: *normal*) expression is reached. *Definitional equality* is the equivalence relation generated by reduction (the precise definitions are in II.3-4).

Now three important desirable properties of the systems, in connection with reduction and definitional equality, are: (1) *normalization*

and *strong normalization*, (2) the *closure property*, (3) the *Church-Rosser property*.

Normalization states that all the correct expressions indeed reduce into a normal expression, i.e. there is a *reduction sequence*, a sequence of expressions produced by successive evaluation steps (*reduction steps*), ending in an irreducible expression. *Strong normalization* says that *all* the reduction sequences of correct expressions terminate. The *closure property* (this term is due to Nederpelt) says that correct expressions remain correct under reduction. Finally the *Church-Rosser theorem* (a corollary of the *Church-Rosser property*) states that two definitionally equal expressions have a *common reduct*, i.e. an expression to which they both reduce. For precise definitions see II.5.

2.8 Formal vs. feasible decidability

A typical application of Church-Rosser theorem and normalization is the *decidability* of the definitional equality on the set of correct expressions. First, by the Church-Rosser theorem we have socalled *uniqueness of normal forms*: An expression has at most one normal reduct. So by combining this with normalization we can define *the normal form* of an expression. Then, thanks to these properties, two expressions are definitionally equal iff they have the same normal form. These can be effectively computed, thus yielding decidability (of definitional equality, from which the decidability of the typing relation follows).

However, computing normal forms is not a very practical way of deciding definitional equality, because normal forms can be very long and complicated expressions, and the reduction sequences leading to them often require many reduction steps. A more practical decision procedure rather relies on strong normalization. Namely, when confronted with two expressions A and B we can try to successively apply well-chosen reduction steps on either A or B until we possibly arrive in a common reduct (thus establishing definitional equality) or we arrive in reducts A' (of A) and B' (of B) which can be recognized not to be definitionally equal. Strong normalization warrants that this process anyhow terminates, no matter what *reduction strategy* has been chosen. Although, in the worst case it might end in normal forms A' and B', in particular this might happen if A and B are not definitionally equal.

Since reducing to normal forms is simply not acceptable in feasible verification procedures, the importance of the formal decidability result and of the *completeness* of the indicated more practical decision procedure must not be overemphasized (as observed by De Vrijer in [79]) - though these facts are, of course, important for a good understanding of the procedure -. In practice, in the Automath project, the action of the verifier can be explicitly bounded by giving a suitable upper limit to the amount of work (e.g. number of steps) it is allowed to perform when trying to establish a definitional equation. If, within this bound no common reduct is reached the equality of the two expressions is provisionally refused and the verifier will ask for further information. This, we think, is in full accordance with the fact that, in principle, the verifier is not expected to do more than a human checker. For more comment on actual verification see III.6, V.4.4 and VIII.6.

Strong normalization has, apart from this, more or less practical application, some theoretically useful consequences. E.g. it simplifies the Church-Rosser proof in any case, and it seems indispensable for the case where *surjective pairing* is present. Besides, *certain* proofs of closure (for Nederpelt's λ) depend on strong normalization (in fact on an even stronger termination property, the *big tree theorem*).
Cf. VII.1.2, VII.3, VII.5.

2.9 The consequences of closure

As an application of closure it is sometimes mentioned that it saves time for the verifier. Namely that the verifier does not need to check for correctness again and again when reducing an expression.

More specific, the *combination* of closure and Church-Rosser is important in the verification procedure. First, the Church-Rosser theorem says that definitional equality (via *any sequence* of correct expressions) can be replaced by definitional equality established via a *common reduct*. Secondly the closure property states that the latter equality passes through correct expressions only.

Besides, closure is connected with many other interesting properties, which are in fact characteristic for the Automath languages, like *preservation of types* (under reduction; this property is elsewhere sometimes called *closure* of the types under reduction), *uniqueness of types*

(this means that proper inclusion of types is impossible), *uniqueness of domains*, and *soundness of* (definitional) *equality* with respect to expression formation and typing relation. See 4.1, 5.4 and V.1.3.

Further, closure is necessary in the βη-Church-Rosser proofs (see VI), for showing the equivalence of various language definitions, and for showing the connections between the various languages.

2.10 The "unstability" of the difficulties of language theory

When proving the nice properties connected with closure one often uses induction on the definition of correctness (for terminology about induction see II.0). This means that the choice of definition, i.e. the order in which the expressions are generated, can be important.

In fact, the present author thinks it surprising *how* important the choice of definition can be in this respect. Example: A proof of closure directly from the algorithmic definition turns out to be rather involved (see VII.3.3), whereas De Vrijer [70] formulated his system λλ-ℓ (essentially AUT-QE+, see 4.9) in such a way that closure was straightforward. (On the other hand, De Vrijer had to prove his *big tree theorem* in order to get decidability, whereas decidability for the algorithmic system just follows from normalization).

Similarly, there is much difference between closely related languages, as regards the difficulties they pose in proving their nice properties: Seemingly harmless modifications of the languages - hardly increasing their expressive power - can make some parts of their language theory considerably more difficult. We mention the transition from AUT-68 to AUT-QE, from AUT-QE to AUT-QE+, or the extension from AUT-QE+ (even without type-inclusion) to Nederpelt's system Λ. See sec. 4 for the characteristics of these languages. And there is the addition of the "*extensional*" reductions η, σ and ε (II.3) which essentially complicate the Church-Rosser proof (ε even spoils the property) without contributing much to the expressive power (see e.g.[37, p. 42]). By the way, the phenomenon that hardly impressive modifications can give rise to considerable extra difficulties is itself the raison d'être of a large part of the Automath language theory: Some properties (closure, βη-Church-Rosser) are interesting properties in Automath, but in ordinary typed λ-calculus just trivialities, though the Automath languages can be considered as mere generalizations of the latter system!

Returning to the Automath languages: generally, we have chosen the strategy of first proving the nice properties for a - in this respect - simple system, and then trying to extend these results to more complicated languages. See V.3, VII.6.

I.3 Something on bound variables

3.1 In this thesis we consider expressions *modulo α-conversion* (renaming of bound variables), i.e. our relation of *syntactical identity* $\equiv$ actually stands for α-convertibility (II.2.2). So, in the sequel, we leave the complications concerning the handling of bound variables out of the discussion. This can be accounted for, e.g., by referring to Curry's classical exposition on substitution [25], to Nederpelt's notion of distinctly bound expressions [51], or via the correspondence with one of the proposals to eliminate the names of bound variables altogether (De Bruijn [10], Staples [66]).

3.2 Both these proposals for nameless dummies reflect the idea that a bound variable occurrence is just an open position in an expression, which has to be uniquely linkable to its binding λ. De Bruijn performs this unique linking by replacing such an open position with a positive number, the *reference depth*, viz. the distance to its binding λ. I.e. the number of λ's one encounters scanning the expression from within until one arives at the binding λ (the latter included). E.g. the bound occurrence x in $\lambda xy{\cdot}y(yx)$ has depth 2, the two bound y's have depth 1. Of course the *binding* variables going with a λ can be skipped in this notation. Staples, on the other hand, replaces all such open positions with one and the same standard symbol (one might as well leave them open) and provides the linking information by attaching a list of positions to every λ. These positions are coded in the form of binary strings, with 0 standing for left part and 1 for right part of the expression. E.g. the position x in $\lambda y{\cdot}y(yx)$ is coded 111, and the y's in $y(yx)$ have codes 0,10 respectively.

In other words, in De Bruijn's notation one counts backwards from a bound position to its binding λ, in Staples' notation one counts forwards from a binding λ to the positions it binds. Example: the name-

carrying expression $\lambda xy \cdot y(yx)$ becomes $\lambda(\lambda(1(12)))$ and $\lambda(111)(\lambda(0,10)(x(xx)))$ respectively, where we have taken x for Staples' standard symbol.

3.3 De Bruijn admits that his system is not particularly suitable for (i) easy reading and writing, but claims it to be good for both (ii) metalingual discussion and (iii) mechanical manipulation - what is was invented for, in the context of the Automath project -. In fact, De Bruijn's system is just *the* symbolic representation of the most straightforward computer implementation of λ-expressions.

Staples thinks his system is better than De Bruijn's for purposes (i) and (ii) and does not know about (iii). The present author thinks there is not much difference between the two systems as regards (i) and (ii) (probably De Bruijn's is somewhat better for (i)), but thinks that De Bruijn's is definitely superior for (iii). He thinks further that both systems, when compared to ordinary name-carrying λ-calculus, are better for (ii) - unless, of course, one wants to study α-conversion - but so much inferior for (i) - at least to people accustomed to ordinary notation but probably to others as well - that he has preferred the ordinary approach in this thesis.

3.4 Zandleven has actually used De Bruijn's system in the implementation of Automath, extending it to a system of socalled *postponed substitution:* substitution instructions are incorporated into the syntax of the system, and so, they can be postponed until needed (e.g. for establishing definitional equality). Since the substitution instructions are also coded by means of reference depths, we call the system a system of *iterated references* (documented in [38]). Closely related are De Bruijn's system of *reference transforming mappings* [16] and Wadworth's system of *graph reduction* [72]. Wadsworth's system is not namefree, but he surely hints at namefree implementation. De Bruijn and Wieringa [19,80] have also studied even more general namefree λ-calculuses.

3.5 In a review [63] of De Bruijn's article [10], Seldin suggested that combinatory logic is as good as any other system for nameless representation of bound variables. Since most λ-calculus theories can only partially be represented in combinatory logic (see, e.g., Hindley [33]), and since the usual translations are rather clumsy (though perhaps Turner's

recent proposal [78] might be satisfactory) we think that Seldin's remark is not quite correct. (Lately (Swansea, 1979, oral communication) Seldin seemed to agree with this view himself.)

I.4 The Automath languages

4.1 General language rules

We give a tutorial survey of the characteristics of the several Automath languages. Other introductory references on AUT-68 and AUT-QE are [27,11], for AUT-SL see VII.1 or [51], for AUT-Pi see VIII.1 or [77]. See also the discussion in IV.1.

We have already announced the generalized type-structure of Automath: the types can be complicated expressions themselves (e.g. they can depend on variables), they are constructed along with the terms and hence, the typestructure cannot be given beforehand - as is usual in ordinary typed λ-calculus -.

So the type-assignment is itself part of the system and does not belong to metalanguage. Consequently the system has besides formulas

$$A \; Q \; B$$

expressing the *definitional equality* of the expressions A and B, also formulas

$$A \; E \; B$$

standing for A *has type* B. An alternative notation for Q is $\stackrel{D}{=}$ or just = (e.g. in[11, 37,70]), for $A \; E \; B$ one sometimes writes $A : B$ (in [20,77]). In fact, in accordance with the implicit character of definitional equality (see below), the Q-formulas are not written down, when actually *using* the Automath system, but are just introduced in the language theory for formal purposes.

All Automath languages have the *right hand equality rule* (or rule of *type conversion*)

$$A \; E \; B, \; B \; Q \; C \Rightarrow A \; E \; C$$

Most languages also have the *left hand equality rule* LQ

$$A \mathsf{E} C,\ A \mathsf{Q} B \Rightarrow B \mathsf{E} C$$

as a *derived* rule (contrarily to the right hand rule, which is part of the language definition). Further, most languages satisfy *uniqueness of types*

$$A \mathsf{E} B,\ A \mathsf{E} C \Rightarrow B \mathsf{Q} C$$

i.e. the "converse" of type conversion. In such languages there can be defined an operation typ, such that, for all correct A,

$$A \mathsf{E} \mathrm{typ}(A), \quad \text{and}$$

$$A \mathsf{E} B \Rightarrow B \mathsf{Q} \mathrm{typ}(A)$$

(this explains why the decidability of Q entails the decidability of E).

The expressions are formed from variables x, y etc. and constant-expressions $c(A_1,\cdots,A_k)$ by the operations of *λ-abstraction* and *application* (in the socalled *pure* languages AUT-68, AUT-QE, AUT-SL) and possibly other operations (in the *extended* system AUT-Pi). Expressions formed according to the rules and the restrictions (in particular the type restrictions) of the various languages are said to be the *correct* expressions of those languages, in contrast with the (general) expressions just resulting from unrestricted use of the formation operations.

4.2 Abstraction and application

The operation *λ-abstraction* leads to *abstraction-expressions* $[x{:}A]B$. Generally such an expression can be interpreted as the *function* $\lambda_{x:A}{\cdot}B$, with domain A and producing values $B[\![D]\!]$ when applied to arguments $D \mathsf{E} A$. Here the postfix $[\![D]\!]$ belongs to the metalanguage; it is short for $[\![x/D]\!]$, i.e. substitution of D for the variable x.

The *application* operation constructs the *application expression* $\{A\}B$. This expression must be interpreted as the result of applying the *function* B to the *argument* A, i.e. the object usually denoted $B(A)$ or BA. The choice of putting the argument in front, between brackets, combines nicely with the notational habit of putting the binding variable $x{:}A$ in front too, between a different kind of brackets, and is generally

preferred in the Automath project. Of course, people grown up with the usual λ-calculus conventions find it difficult to get used to such a new notation. (Admittedly, it would have been consistent with our notation for application to put the substitution operator in front too. However we do not find this too important because substitution just belongs to metalanguage.)

4.3 Reduction and definitional equality

The definitional equality is a restricted form of equality, just covering certain identifications which in ordinary mathematics are understood without any explicit justification. It is defined in a combinatorial, syntactical way, viz. as the equivalence relation generated by socalled *reduction steps*. Each reduction step replaces a part of an expression, a *redex*, by another expression, a socalled *contractum*. This is the usual terminology in λ-calculus, where definitional equality is often called *convertibility*. In order that the so-defined relation is acceptable as definitional equality, it must clearly be required that redex and contractum are intuitively equal. Our notation for reduction is ≥. The reductions associated with abstraction and application are β- and η-reduction:

β-reduction: $\{A\}[x{:}B]C \geq C[\![A]\!]$

η-reduction: $[x{:}B]\{x\}C \geq C$ if C does not depend on x.

There is also associated a reduction (called δ-reduction) to the expressions $d(A_1,\cdots,A_k)$ where d is a *defined constant*. For such defined constants *defining axioms* (*abbreviations*, with parameters)

$$d(x_1,\cdots,x_k) := D[\![x_1,\cdots,x_k]\!]$$

are given. Here the postfix $[\![x_1,\cdots,x_k]\!]$ is to indicate that D may depend on the variables shown.

The δ-reduction reads

$$d(A_1,\cdots,A_k) \geq D[\![A_1,\cdots,A_k]\!]$$

where $[\![A_1,\cdots,A_k]\!]$ stands for $[\![x_1,\cdots,x_k/A_1,\cdots,A_k]\!]$, the simultaneous

substitution of $A_1,\cdots,A_k$ for $x_1,\cdots,x_k$. Our δ-reduction is distinct from other δ-reductions in the literature (cf. II.3.2.4).

The equality generated by β, η and δ indeed corresponds to the intuitive interpretation of abstraction and application, and to the idea of abbreviation. However, certain restrictions have to be fulfilled. In particular, η-equality is only acceptable if the C (in the η-redex, above) is also a function, with domain B. Since in the general, unrestricted expressions such provisions are not necessarily satisfied, we define Q between correct expressions A and B only, and also require that the expressions "in between" A and B (i.e. via which the conversion from A to B can be established) are correct as well. For precise definitions of reduction and equality see II.3-4, for Q see V.2. For the additional operations (with associated reductions) of AUT-Pi see VIII.1.

4.4 Type assignment

Type assignment takes place together with expression formation. The variables get a type by *assumption* (of the form x E A). Formulas are derived and expressions are constructed in natural deduction style, i.e. relative to a set (in our case: a string) of assumptions, called the *context* of the formula, resp. the expression. Such a context has the form

$$x_1 \text{ E } B_1, x_2 \text{ E } B_2[\![x_1]\!], \cdots, x_k \text{ E } B_k[\![x_1,\cdots,x_{k-1}]\!]$$

where all the x_i are distinct. (This notion of context is only vaguely related to the notion of context nowadays used in λ-calculus theory.) If ξ is a context we sometimes write

$$\xi\vdash A,\ \xi\vdash A \text{ E } B,\ \xi\vdash A \text{ Q } B$$

to indicate that an expression or formula is correct, resp. derivable, with respect to ξ. Here ξ contains so to say the type declarations of the variables on which A (resp. A E B, A Q B) depends.

The constant expressions obtain a type by *instantiating of* (i.e. *substitution in*) a *scheme*. A scheme consists of an axiomatic type assignment with parameters

$$c(x_1,\ldots,x_k) \text{ E } C[\![x_1,\ldots,x_k]\!]$$

relative to a context

$$x_1 \mathsf{E} B_1,\ x_2 \mathsf{E} B_2[\![x_1]\!],\ \cdots,\ x_k \mathsf{E} B_k[\![x_1,\cdots,x_{k-1}]\!].$$

Only such instantiations $c(A_1,\cdots,A_k)$ are admitted, where the A_i meet the type requirements of the context, i.e.

$$A_1 \mathsf{E} B_1,\ A_2 \mathsf{E} B_2[\![A_1]\!],\ \cdots,\ A_k \mathsf{E} B_k[\![A_1,\cdots,A_{k-1}]\!].$$

Then the type assignment to the constant expression becomes

$$c(A_1,\cdots,A_k) \mathsf{E} C[\![A_1,\cdots,A_k]\!].$$

A list of constant schemes is called a *book* and the constants c are called *book constants* (to distinguish them from the language constants). There are two kinds of constants, viz. *primitive* constants, having a type-assignment only, and *defined* constants, having a defining axiom (as mentioned in 4.3) *and* a corresponding type-assignment (see below). All constants in the book are distinct so each book constant has a unique type-assignment (resp. unique defining axiom). If d has defining axiom $d(x_1,\cdots,x_k) := D$ and typing $d(x_1,\cdots,x_k) \mathsf{E} C$ then, for the sake of the intuitive interpretation, it must be required that $D \mathsf{E} C$ w.r.t. the context of the scheme. This is the *compatibility condition* of def and typ. For more precise definitions see IV.3.2,IV.3.3, V.2.1.

4.5 The rules of AUT-68

As for the application and abstraction rules, we first describe the simplest language, now named AUT-68. This language has three kinds of expressions: *terms* (also called expressions of *degree* 3, or: 3-expressions), *types* (with degree 2, or: 2-expressions) and a single *untyped* constant type (also denoted τ, and called a *supertype* or 1-expression, of degree 1). Languages with expressions of degree 1, 2 and 3 only are said to be *regular*.

The 1-expressions generally serve as types for the 2-expressions, but do not have a type themselves. Notice that the word "type" is used ambiguously here, viz. to name the 2-expressions *and* in the sense of: "being the type of". Typically, the types are the types of the terms and (in AUT-68) type is the type of the types.

So, in AUT-68 there are two cases $A \in B$: either A is a term and B is a type, or A is a type and $B \equiv \mathsf{type}$ ($\equiv$ means syntactical identity). In terms of degrees: if $A \in B$, B has degree i then A has degree i+1. This property holds generally, also in the *irregular* languages, like AUT-SL, where expressions of all positive degrees are admitted.

Now we give the term formation rules for AUT-68. First notice that all variables have a type, so must be a type variable (of degree 2) or a term variable (of degree 3). The abstraction rule reads: if from an assumption $x \in A$, and possibly other assumptions not depending on x, it can be derived that $B \in C$, where x is a *term variable* and B is a term, then one can conclude that $[x{:}A]B \in [x{:}A]C$ and *discharge* the assumption $x \in A$. In natural deduction notation

term abstraction rule

$$\begin{array}{c} [x \in A] \\ \vdots \\ B \in C \end{array} \qquad \text{degree}(x) = \text{degree}(B) = 3$$

$$[x{:}A]B \in [x{:}A]C$$

Actually, in Automath only the last assumption in the context is allowed to be discharged. The remaining assumptions clearly satisfy the above mentioned restriction (of not depending on x). We refer to the fact that the context is a string rather a set (and consequently, that the assumptions can be removed according to the last-in first-out principle) by speaking of the *linear* natural deduction character of Automath. In the notation of this thesis the rule becomes:

$$\vdash^2 A,\ (x \in A \vdash^3 B \in C) \Rightarrow [x{:}A]B \in [x{:}A]C$$

with $\vdash$ standing for correctness, resp. derivability, with the superscripts indicating the degrees (for the precise conventions see V.2.1.1).

In order to guarantee that the type of correct expressions are correct too, there must be an abstraction rule for types as well. This one reads

*

type abstraction rule AUT-68

$$\frac{\begin{array}{c}[x \mathrel{E} A] \\ \vdots \\ C \mathrel{E} \text{type}\end{array}}{[x{:}A]C \mathrel{E} \text{type}} \qquad \text{degree}(x) = 3$$

In our notation

$$\vdash^2 A,\ (x \mathrel{E} A \vdash C \mathrel{E} \text{type}) \Rightarrow [x{:}A]C \mathrel{E} \text{type}$$

Then there is the application rule for AUT-68:

application rule AUT-68

$$\frac{D \mathrel{E} A \qquad B \mathrel{E} [x{:}A]C}{\{D\}B \mathrel{E} C[\![D]\!]}$$

4.6 Interpretation

Now something about interpretation. With the 3-expressions $[x{:}A]B$ and $\{D\}B$ constructed above there is no problem: $[x{:}A]B$ is the function $\lambda_{x:A}{\cdot}B$, $\{D\}B$ is the result of applying function B to argument D. But consider the 2-expression $[x{:}A]C$ occurring in the rules above. Under the most convenient interpretation, maintaining that a type is a kind of *set* or *class*, and that the E-relation is a kind of element relation, $[x{:}A]C$ must stand for the object usually denoted $\prod_{x:A} C$ or $\Pi(\lambda_{x:A}{\cdot}C)$. I.e. the cartesian product of all the $C[\![D]\!]$, for $D \mathrel{E} A$. In case C does not depend on x, this product reduces to the function space $A \to C$ which in type theory would be denoted (AC) or the like. In other words, $[x{:}A]C$ is the "set" (class, aggregate) consisting of all the functions B with domain A which, when applied to arguments D in A, produce values belonging to $C[\![D]\!]$. This is precisely what the *appl* rule says. So in this interpretation the *abstractor* $[x{:}A]$ has two different meanings: when used with a term it gives a function, when used with a type it gives a kind of set. Or, we can say that $[x{:}A]$ has just one meaning, viz. $\lambda x{:}A$, but that the Π has been omitted, for brevity, in a situation where no confusion is reasonably possible. This is the standard interpretation corresponding with the notation in related typed λ-calculus systems and in AUT-Pi (see VIII.1).

However there is a second, alternative, interpretation, too. It is not necessary to stick to the idea that types are sets and that E is a kind of element relation. Namely, we can very well interpret $[x{:}A]C$ as the function $\lambda_{x:A}{\cdot}C$, if only we accept that a *function* can act a *type*. Then, the term *abstr* rule says (i.a.) that the type of a function is again a function, with the same domain, and, conversely, the appl rule says (i.a.) that the functions of degree 3 are *characterized* by having a function for their type, from which their domain can be read off. In this interpretation the conclusion of the term abstraction rule $([x{:}A]B \ \mathsf{E} \ [x{:}A]C)$ just mean $\forall_{D \ \mathsf{E} \ A} \ (B[\![D]\!] \ \mathsf{E} \ C[\![D]\!])$, i.e. the rule abstracts the formula $B \ \mathsf{E} \ C$ rather than the expressions involved. In algebraic terms: the rule can be considered as a *distribution rule* of the abstractor $[x{:}A]$ w.r.t. the E-relation.

This, second, interpretation has given rise to several extensions of the language, viz. to AUT-QE, to socalled +-languages (AUT-68+ and AUT-QE+), and even to AUT-SL (i.e. Nederpelt's Λ).

4.7 AUT-QE

First the extension to AUT-QE. Since we interpret the 2-expression $[x{:}A]C$ as a (type valued) function, and since we want a uniform method of type assignment for both term valued and type valued functions, we drop the restriction to B of degree 3 in the term abstraction rule of AUT-68, thus getting the

general abstraction rule: $\vdash^2 A, \ (x \ \mathsf{E} \ A \vdash B(\mathsf{E} \ C)) \Rightarrow \vdash [x{:}A]B(\mathsf{E} \ [x{:}A]C)$

So the degree restriction for the variable x is maintained. In the new rule there is included (skip the two E-parts between parentheses) the abstraction rule for 1-expressions, to guarantee that the types of correct expressions are correct again:

$$\vdash^2 A, \ (x \ \mathsf{E} \ A \vdash^1 B) \Rightarrow \vdash^1 [x{:}A]B$$

So in AUT-QE there are other supertypes than just type, of the form

$$[x_1{:}A_1]\cdots[x_k{:}A_k]\mathsf{type}.$$

These expressions have originally been named *quasi-expressions*, whence the name of the language AUT-QE.

The application rule of AUT-68 is maintained in AUT-QE:

application rule I $\quad D \in A,\ B \in [x{:}A]C \Rightarrow \{D\}B \in C[\![D]\!]$

but is more general here, because it can be used with B of degree 3 *and* 2 now (in AUT-68 only with B of degree 3). Besides, AUT-QE has, in accordance with the proposed interpretation, another appl rule:

application rule II $\quad E \in A,\ B \in C \in [x{:}A]D \Rightarrow \{E\}B \in \{E\}C$

Namely, $[x{:}A]D$ is a function with domain A, so C is a function with domain A, so B is a function with domain A and can be applied to the argument $E \in A$. (In fact, this rule can be derived from appl rule I by η-equality, which confirms the agreement with the interpretation.)

Just like a degree 2 abstr expression of AUT-68 allows different interpretations, viz. as a set or as a function, a degree 1 abstr expression of AUT-QE has such different interpretations too. Under the first interpretation the expression $[x_1{:}A_1]\cdots[x_k{:}A_k]\mathrm{type}$ stands for the object

$$\prod_{x_1:A_1}\Big(\prod_{x_2:A_2}\big(\cdots\big(\prod_{x_k:A_k}\mathrm{type}\big)\cdots\big)\Big)$$

This corresponds with the notation of AUT-Pi, see VIII.1. Under the second interpretation it stands for the object

$$\lambda x_1{:}A_1.\lambda x_2{:}A_2.\cdots\lambda x_k{:}A_k.\mathrm{type}$$

4.8 Type inclusion

Now let $x \in A \vdash C \in \mathrm{type}$. Two rules of type assignment are applicable, viz. the type abstr rule of AUT-68 and the general abstr rule, giving rise to

$$[x{:}A]C \in \mathrm{type},\ \text{resp.}\ [x{:}A]C \in [x{:}A]\mathrm{type}$$

Generally a 2-expression $[x_1{:}A_1]\cdots[x_k{:}A_k]C$ of AUT-QE has as its possible types

$$\mathsf{type},\ [x_1{:}A_1]\mathsf{type},\ [x_1{:}A_1][x_2{:}A_2]\mathsf{type}\ \text{etc.}$$

$$\text{up to, at least } [x_1{:}A_1]\cdots[x_k{:}A_k]\mathsf{type}\ .$$

This ambiguity of types, which is typical for AUT-QE, is usually implemented by adding a *rule of type inclusion*

$$B\ \mathsf{E}\ [x_1{:}A_1]\cdots[x_k{:}A_k][y{:}C]\mathsf{type} \Rightarrow B\ \mathsf{E}\ [x_1{:}A_1]\cdots[x_k{:}A_k]\mathsf{type}$$

and dropping the type abstraction rule of AUT-68, which now becomes a derived rule. In fact, the type inclusion rule is somewhat stronger than the type abstraction rule of AUT-68 (or, similarly, the product rule of AUT-Pi). See VIII.1.5 and VIII.6.1.

Clearly the property of uniqueness of types

$$A\ \mathsf{E}\ B,\ A\ \mathsf{E}\ C \Rightarrow B\ \mathsf{Q}\ C$$

is, for 2-expressions A, not valid any more in AUT-QE. This is, however, the *only* case of proper type-inclusion in Automath languages. We introduce $\sqsubset$ to denote type-inclusion, i.e.

$$B \sqsubset C :\Leftrightarrow \forall_A(A\ \mathsf{E}\ B \Rightarrow A\ \mathsf{E}\ C).$$

For the precise definition see V.2.13 or V.3.2. The possible types of a 2-expression appear to be *linearly ordered* under $\sqsubset$, so

$$A\ \mathsf{E}\ B,\ A\ \mathsf{E}\ C \Rightarrow B \sqsubset C \quad \text{or} \quad C \sqsubset B$$

and it is still possible to define a *canonical type* which is minimal, w.r.t. $\sqsubset$, among the possible types (and hence gives maximal information), i.e. such that

$$A\ \mathsf{E}\ B \Rightarrow A\ \mathsf{E}\ \mathsf{typ}(A) \sqsubset B.$$

4.9 +-languages

Now the extension to +-languages. Recall that in AUT-68 there were abstr expressions of degree 3 and 2, but appl expressions of degree 3 only. We say the *value degrees* are 2 and 3, and the *function degree* is 3. Here we use the terminology of V.2.7: B is called the *value part* of $[x{:}A]B$ and the function part of $\{A\}B$. Similarly AUT-QE has value

degrees 1, 2 and 3 and function degrees 2 and 3. Such languages, where the minimal value degree is not a function degree are named *non-+-languages*.

However, if the abstraction expressions of minimal value degree are functions, it is reasonable to have an appl rule for them too:

appl rule +-languages $\quad D \mathrel{E} A,\ B\ Q\ [x{:}A]C \Rightarrow \vdash\{D\}B$

In particular, if $D \mathrel{E} A$, $\vdash[x{:}A]C$ then $\vdash\{D\}[x{:}A]C$. Indeed, by adding the above rule for B of degree 2 to AUT-68 we arive at the +-language AUT-68+. And by adding it to AUT-QE for B of degree 1 we arive at AUT-QE+ (which is essentially $\lambda\lambda$-ℓ, the *legitimate* fragment of De Vrijer's $\lambda\lambda$ [70]). In principle, the new rule is a derived rule for E not having minimal value degree. The words "in principle" here refer to certain problems with type inclusion and defined constants, explained at length in V.1.7, V.3.3 and V.4.2.

It will be shown (V.3.3 , V.3.4) that a +-language is an *un-essential* (and even, *definitional*) *extension* of the corresponding non-+-language (see V.3.3):

$$\vdash_{+} A \Rightarrow \exists_{A'}(\vdash A' \ \&\ A\ Q_{+}\ A')$$

i.e. to each A in the +-system there corresponds a definitionally equal A' correct in the smaller system.

In all the languages now defined, the rule

general application rule $\quad B \mathrel{E} C,\ \vdash\{A\}C \Rightarrow \{A\}B \mathrel{E} \{A\}C$

is a derived rule. Alternatively, this rule can be adopted in the language definition, either with the application rule I (in the non-+-languages), or with the application rule for +-languages, to generate all the appl expressions of the various languages. The nice point about the general application rule is that it (similar to the general abstraction rule) can be considered as a kind of distribution rule, viz. of the applicator $\{A\}$ w.r.t. the E-relation.

Though in AUT-QE+ we have achieved a fairly uniform treatment of expressions of all degrees, we still have maintained the restriction that only abstractors $[x{:}A]$ with degree(x) = 3, degree(A) = 2 are

formed. In other words, only term variables are *quantified*. So there is no quantification over type variables and we say that our systems are *first-order* (this term refers to the fact that in the propositions-as-types interpretation quantification over types gives rise to *higher-order logic*). Consequently only applicators $\{A\}$ with degree(A) = 3 are admitted. We say that the only *domain degree* is 2, and the only *argument degree* is 3 (A is said to be the *domain part* of $[x{:}A]B$ and the *argument part* of $\{A\}B$). Apparently there is a certain duplication in having both instantiation and application in the system. However, because of the aforementioned application restriction instantiation cannot be missed: substitution of 2-expressions (for type-variables) cannot be performed by means of application so has to take place by means of instantiation. (See also 5.6)

4.10 AUT-SL

Now we explain how AUT-SL (i.e. Nederpelt's Λ) can also be considered a result of our extended interpretation of the E-symbol. Namely, now that we have accepted that *functions* can be *inhabitable*, i.e. can be the type of other expressions, there seems to be no principal objection against allowing *each* expression to be inhabitable. This is indeed the most striking characteristic of Λ: there are expressions of all positive degrees admitted, so Λ is *irregular* (sec. 4.5). (Here is an analogy with the *language* of set theory where a priori no term is excluded from being inhabitable, i.e. from being a set).

Further, in Λ all degrees are domain degrees, so all degrees but 1 are argument degrees, so instantiation can be missed and, indeed, has been dropped. Still, we shall *not* call Λ a *higher-order* language (IV.1.5.3, VII.1) because any form of type inclusion has been omitted. So, AUT-68 and AUT-QE which are based on type-inclusion, are not included in Λ, and uniqueness of types holds in Λ. For more information about the background of Λ see VII.1.

The definition of Λ either must contain the general application rule, above, or for B of degree k, k $\geq$ 2,

$$D \mathbin{E} A,\ B \equiv C_k \mathbin{E} \cdots \mathbin{E} C_1 \equiv [x{:}A]E \ \Rightarrow \{D\}B \mathbin{E} \{D\}C_{k-1}$$

In fact, Nederpelt gives an *algorithmic definition* of Λ, in terms of a type function typ, and in terms of unrestricted reduction $\geq$, instead of a socalled E-*definition* in terms of E- and Q-formulas, such as the definitions given above. For a discussion of algorithmic definition vs. E-definition see V.1.2 and for the equivalence of both definitions see V.4.

Because of the simple form of the general abstraction and application rule, the function typ has a very simple definition too, in particular

$$\mathrm{typ}(\{A\}B) :\equiv \{A\}\mathrm{typ}(B),\ \mathrm{typ}([x{:}A]B) :\equiv [x{:}A]\mathrm{typ}(B)$$

Nederpelt gives a socalled *application condition* which in our notation, for B of degree k would read

$$D \mathrel{E} A,\ \mathrm{typ}^{k-1}(B)\ Q\ [x{:}A]E \Rightarrow \vdash\{D\}B$$

(where typ^{k-1} stands for k-1 successive applications of the function typ), completely in accordance with our application rule for B of degree k, above. By the way, we write, like Nederpelt, typ^* for the typ^{k-1} of expressions of degree k.

The language Λ was invented for theoretical purposes. It is interesting because it has a very simple and elegant definition and exhibits some typical Automath features. However, because it is in some sense weaker (no type inclusion) than AUT-68 and AUT-QE, results valid for Λ cannot directly be transferred to these, from a practical point of view, more important languages. In particular, the *"strict" normability* of Λ (proved by Nederpelt) is easier to prove than the *"weak" normability* of AUT-QE (see IV. 3-4) because of the *weak second order* aspect AUT-68 and AUT-QE. See IV.1.5 See also VIII.4.2.2 for an interesting interpretation of these normability results (inspired by Ben-Yelles [6]).

Conversely, the facts that Λ is a +-language, is irregular, and has no abstraction restrictions, pose certain difficulties which in the theory of AUT-68 and AUT-QE can be avoided.

The present author has mainly devoted his language theoretical attention directly towards the languages actually being in use: AUT-68, AUT-QE and AUT-Pi. In this theses we *have* indeed at some places introduced new languages (for technical or expository reasons), but we

have tried to exhibit the precise connections with existing languages. Also, we have devoted a chapter (VII) to Λ, which deserves some interest of its own.

4.11 AUT-Pi

For an informal introduction to AUT-Pi see VIII.1. In AUT-Pi the standard mathematical distinction between types (being inhabitable) and functions (not being so) is made by putting in Π's at the proper places (whence the name AUT-Pi). In VIII.6 the difference has been indicated between the rule for inserting Π's (the *product rule*) and the rule of type-inclusion of AUT-QE.

4.12 Two higher-order languages

For completeness reasons we mention two proposals for higher order languages. First, De Bruijn once proposed a language AUT-4 [14], where the proofs come in as degree 4 expressions (whence AUT-4), instead of, as usual (5.9, 5.2), as degree 3 expressions. AUT-4 would have provided an application of the higher degrees of irregular languages, but has never been used or implemented. Secondly, the author has introduced a language (let us name it AUT-2) which has expressions of degree 1 and 2 only, with unrestricted type-inclusion rule (sec. 4.8) and without abstraction restrictions. This language proved to be essentially identical to a system of type-assignment to λ-calculus terms invented by Dezani and Coppo [22,23] for quite different purposes. These two languages are *not* discussed in this thesis. It seems that (strong) normalization for AUT-4 can only be proved by Girard-like methods [30, 31], whereas for AUT-2 we have a strong normalization proof in the style of this thesis.

I.5 Mathematics in Automath

5.1 Survey of this section

Because of the presence of a type (type) of types, the presence of type-variables and the generalized type-structure, people often tend to overestimate the expressive power of (i.e. what can be said in) the Automath languages. Here we refer to the expressive power of the languages *as such*, i.e. to what can be said *directly* in the *basic system*, without any constants added. (Because, *with* additional constants, as we shall see, almost anything can be expressed, just like in the language of first order predicate logic.)

Below we sketch what has become the standard development of mathematics in Automath. The emphasis will be on the inherent limitations of Automath. Occasionally we make a comparison with closely related systems: Seldin's system of *generalized functionality* [64], Scott's system of *constructive validity* [62] and Martin-Löf's systems of *intuitionistic type theory* [45,46], and Girard's systems for *analysis* [31]. Throughout we comment on the typical Automath features.

5.2 The t-part and the p-part of Automath

Let us, for the sake of the exposition, divide mathematics in two parts: one part, let us say the *object part*, dealing with the *construction* of mathematical *objects* (resp. types), and one part, the *logical part*, for reasoning *about* these objects. Our framework of Automath languages, above, is formulated in terms of objects and types, rather than in logical terms: there are, indeed, Q- and E-formulas expressing facts *about* the objects, but they just play an auxiliary role, viz. to control the construction of the *correct* (sec. 2.6) objects.

Following [37,77] we name the fragment of Automath that deals with the object part the *t*-fragment (for *t*erms, *t*ypes and *t*ype-valued functions), and the fragment of Automath representing the logical part the *p*-fragment (for *p*roofs, *p*ropositions, *p*redicates). Degree i (sec. 4.5) expressions of the t-fragment and the p-fragment are said to be i-t-expressions and i-p-expressions respectively.

So, whereas the preceding sections suggest how the t-fragment can be developed (3t-expressions for objects, 2t-expressions for types), it is a priori not clear how the p-fragment will express the logical part. Essential is that the E-formula A E B, of the p-fragment, with A a 3p-expression and B a 2p-expression, is interpreted as *expressing the truth of the proposition* B (i.e. as expressing B itself). So, a proposition is true (asserted) if "we have something in it", i.e. if we have a (3p-) expression having the proposition for its type.

There are several ways of interpreting the *realizer* A (we borrow this term from Pottinger [58] who borrowed it from Helman), i.e. the expression we have in the proposition B: as an abstract proof construction proving B, as a symbolic translation of a natural deduction proof figure (with B as its end formula), or as just some indication (some reference to the fact) that B holds. If we are interested in constructive foundations the first interpretation is appropriate. If we want to study proof figures (e.g. in view of normalization properties) the second interpretation is the best one. If we just want classical logic the third point of view seems to be right, and it also seems justified to identify (in the sense of definitional equality) all the realizers of one and the same proposition. This identification principle is called *irrelevance of proofs* [77,37,20].

We will explain that the *propositions-as-types* way, as sketched above, of fitting the logical part of mathematics into a typed λ-calculus framework arises quite naturally from the idea of mechanical proof-checking (and, on the other hand, that it is the *only* way of expressing actual reasoning in terms of the E- and Q-formulas).

5.3 The t-fragment

Generally speaking, the systems introduced in sec. 4 are as yet still *empty* because we have not introduced any *constants*. Here we adopt the common point of view that the meaningful objects (resp. types) of a theory correspond to its closed expressions (i.e. those not depending on variables). One way to construct closed terms is from constants, another way is by binding the variables in an expression, i.e. by λ-abstraction. Since in most Automath languages abstraction over type-variables is forbidden we need at least one *primitive type-constant*

before we can start generating closed expressions. (Here Λ is an exception: In Λ the basic constant τ (this is just an alternative notation for type) can be used as a ground type and we can directly start constructing functions of type $\tau \to \tau$ etc.)

In the Automath project it has sometimes been stated, that there is no essential difference between a constant without parameters - i.e. introduced in an empty context - and a variable. This is formally right: a constant can be conceived as a variable one does not want to get rid of, and for which no substitution is possible. Conceptually however, it seems better to maintain the distinction.

We just sketch very briefly how the typed λ-calculus framework of Automath can be used to construct the objects (numbers, functions, functionals) forming the universe of discourse of ordinary mathematics (say, analysis). One first introduces some *primitive type constants* (2t-expressions) for the *ground types*, the *natural* and the *real numbers*, say, by stating as an *axiom* (i.e. an axiom scheme in an empty context): nt E type, rl E type. (Of course, if one knows a bit more one can also *define* the reals in terms of the natural numbers, but that does not concern us here.) Secondly, one introduces some *primitive term constants* (3t-expressions) for generating the objects of these types. E.g. in order to construct the natural numbers one states axioms one E nt, sucfun E nt → nt (the successor function, which can alternatively be introduced by a scheme, see below). From these constants we get the natural numbers, which we can give a new name by introducing *defining constants*: two := {one}sucfun, three := {two}sucfun(Q{{one}sucfun}sucfun) etc. If one likes, one can also introduce primitive constants plusfun E nt → (nt → nt) and timesfun E nt → (nt → nt) for *plus* and *times* on the naturals. Additional (equality) axioms will be needed to fix the properties of the thus constructed objects, but these rather belong to the logical part. Similarly, constants can be introduced (with the additional axioms) to generate the objects of type rl.

By λ-abstraction closed expressions of *higher type* are constructed. These higher types themselves (we already used some of them) are also constructed by λ-abstraction (in AUT-QE etc.) or by λ-abstraction and product formation (in AUT-Pi). E.g. we get nt → rl, the type of real number sequences, (rl → rl) → rl the type of real functionals etc. We see that up to now there seems to be no possibility to introduce

non-trivial type-valued functions: the higher types shown are just (products of) the *constant* type-valued functions
$[x{:}\mathsf{nt}]\mathsf{nt}$, $[x{:}\mathsf{nt}](\mathsf{nt} \to \mathsf{nt})$ etc.

In fact, the type-valued functions do not become essential before we arrive at the p-part. However, we give an example of a typical type-valued function in the t-part (see [37]): In the context x E nt we can introduce the primitive 2t-constant 1to(x) intended to contain the natural numbers up to x, as follows

$$x \text{ E } \mathsf{nt} \vdash \mathsf{1to}(x) \text{ E type}$$

(This cannot become an actual subtype of nt (cf. 5.4), injection functions and equality axioms will be needed.) From this scheme we can construct the non-trivial type-valued function $[x{:}\mathsf{nt}]\mathsf{1to}(x)$ (a 2t-expression). It depends of course on the additional axioms what objects will belong to this type.

It is an interesting question what higher type objects (functions and functionals) can actually be defined by mere λ-abstraction (either from object constants, or just from variables): of course we have *constant* functions and *selectors* $\lambda x_1 \cdots x_n . x_j$, and we can define composition of functions, but what else? For an answer see Plotkin [54].

5.4 Some comment on the t-part

From the examples, above, several characteristic features and limitations of Automath become clear. First, that the whole development is based on typed λ-calculus rather than on set theory. More about this in the next section. Then a point on *defined constants*: from our present point of view (What objects are actually constructed?) they are irrelevant, because they just serve as new names for objects already present. From a practical point of view, however, they form an indispensable feature of Automath.

Another characteristic facility of most Automath languages is that a function can be introduced in two ways, viz. either as a single higher type constant or, by a scheme, as a constant depending on parameters (in this case the constant rather stands for the function *value*). Above, sucfun, plusfun and timesfun were introduced by the first method. Alternatively, one might introduce suc, plus and times by an *axiomatic typing*

scheme, i.e. depending on variables of type nt:

x E nt⊢suc(x) E nt

x E nt, y E nt⊢plus(x,y) E nt etc.

That these mechanisms really form a duplication is shown by the fact that they can be defined in terms of each other, e.g.

sucfun := [x:nt]suc(x), resp.

x E nt⊢suc(x) := {x}sucfun etc.

More about schemes can be found in section 5.6.

Now we arrive at some mutually related characteristic limitations of the Automath languages (further elaborated in 5.7). First that hardly any mathematical structure is given beforehand: *even* the natural numbers have to be introduced by a series of constants and axioms (this point we have mentioned before).

Secondly that a type must be present before it can be postulated to be inhabited, i.e. a type must be introduced *before* the objects of that type. This contrasts with the common ideas about the *set theoretic hierarchy* where sets cannot be constructed unless their elements are given (and *grasped*, as one says). In fact, this distriction between types and sets suggests that, after all, the ground types must be understood as syntactic linguistic categories rather than as actual mathematical objects themselves (compare [46]). Then, the higher types can be understood in terms of the ground types.

A third limitation of Automath (related to the second one, though) is the *uniqueness of types*. In the above development one might think it *handy* if the number one of type nt would be of type rl as well and, more general, if nt would be an actual subtype of rl (in the sense of ⊏, see 4.8). Such *proper inclusion of types* is *not expressible* in Automath, and *non-trivial intersections* of types are *not present* either. (Whether the identification of the natural number one with the corresponding real number would be *justified* is another question. See De Bruijn [12].)

5.5 The typed λ-calculus framework

This section tries to support the choice of basing Automath on the concept of *function* rather than on the concept of *set*. The first point is, that in *almost* any interesting part of mathematics some form of *abstraction* is needed, either as *λ-abstraction*, or as a *comprehension axiom*. (The alternative to abstraction is a development in the style of combinatory logic, as in von Neumann-Bernays-Gödel set theory.) As stipulated by De Bruijn [10], λ can be considered as *the, neutral* binding operator, not to be explained in more primitive terms. E.g.the comprehension set $\{x|A\}$ can be defined in terms of λ by, say, $setof(\lambda x.A)$.

The second point is, that the *primitive* concept of function is *basic* in ordinary mathematics (analysis, say). It is, of course, well-known that the graph of a function can be *coded* (*implemented*, say) as a set - and we don't deny that the graph concept itself can be clarifying -, but in ordinary mathematics there is usually no point in this *implementation*. In fact it just shows the well-definedness of the function concept (i.e. of a function on a given domain) in terms of the commonly accepted formal development of axiomatic set theory - which for a practical mathematician is hardly doubtful and probably uninteresting-. Compare [12]. Similarly the possibility of implementing other familiar concepts (the natural numbers, the reals, the complex numbers) in axiomatic set theory, or in any other form, is usually of no practical importance.

By basing one's function concept on βη-λ-calculus one gets the possibility of making *explicit definitions* of functions (by λ-abstraction), and of making those identifications (by definitional equality) that follow from these explicit definitions. Clearly, the graph concept of functions gives more, viz. *extensionality*, whereas βη-equality just pins down the function *intensionally*, i.e. as a *rule*. Additional equality axioms (not for definitional, but for *book equality*) are needed for extensionality. We stress that η just gives a very weak form of extensionality. According to Scott, the η-equality $\lambda x.fx = f$ (in ordinary λ-calculus notation) must not be understood as extensionality but rather as stating that f is a function. So, in a typed setting η seems to be anyhow justified: the mere correctness of $[x:\alpha]\{x\}f$ (in Automath notation) warrants that f is a function. However, η-equality presupposes uniqueness of types!

Above we have taken for granted that the appropriate practical function concept is a *typed* one. Indeed, free, untyped λ-calculus is a farreaching,a priori just formal, extension of this concept (compare, e.g., the notations for limits and formal series, in analysis). It is an extension useful for studying computations but which does not seem very well applicable to "ordinary" mathematics. Compare LCF, being intended for the former purpose and actually based on the *polymorphic typed* λ-calculus PPλ, where the type conventions are not quite as strict as in ordinary typed λ-calculus.

We note that these two restrictions of the definitional equality (that it just covers *intensional* equality, between *ordinary* typed λ-calculus objects) are essential for its being decidable (in contrast with, e.g., the convertibility in PPλ).

5.6 Axioms vs. schemes, abstraction vs. abbreviation

In 5.4 we saw that there are two possibilities to introduce primitive constants for the construction of functions, either at low type level (example: suc) in a *scheme*, or in a higher type by an *axiom* (example: sucfun). The difference between the two approaches is that from a *scheme* objects are constructed by *instantiation* (example: suc(one)), and from the corresponding higher type axiom by *application* (example: {one}sucfun). In most logical formalisms the distinction between instantiation and application cannot be stated in such an explicit form, since their instantiation mechanisms belong to metalanguage.

Similarly there are in Automath (usually) two possibilities for making *explicit definitions* of functions: by *λ-abstraction* and by a *definitional axiom scheme*.These definitions are respectively eliminated by application plus β-reduction and instantiation plus δ-reduction (this duplication is eliminated in Nederpelt's Λ).

Apart from the fact that writing schemes allows a form of (*substitutional*) *quantification* of variables not quantifiable by λ (viz. type variables), it also allows quantification of more variables at a time. However, as one knows, this simultaneous quantification can be simulated by successively quantifying one variable at a time.

So, roughly speaking, what can be done by schemes can also be done

by λ-abstraction. In some sense schemes are simpler than abstraction: higher type objects are avoided. Indeed, in the Automath project a schematic introduction of constants (i.e. SUC instead of SUCFUN etc.) would generally be preferred. And, rather than asking how instantiation can be dismissed in favour of application, one should ask what abstraction, application and higher type objects actually contribute. We think that λ-calculus only comes in when one wants to *express* nested quantifications (either *substitutional* or by *λ-abstraction*) such as, e.g., needed when quantifying over functions or defining functionals. Example: the proposition CONT(f) expressing the continuity of f depends on the higher type variable f. If one wants to *use* this proposition (by instantiation), higher type objects (like $[x{:}r]F$) must be substituted. De Bruijn has, accordingly, conjectured that up to 18th century mathematics is expressible without λ-calculus and, hence, that the primitive Automath language PAL would do for that subject.

5.7 More on the language restrictions (as mentioned in 5.4)

The fact that no arithmetic is built in, distinguishes Automath from systems meant to give a foundation for constructive mathematics. In particular, we want to make a comparison with the system of Scott [62] and Martin-Löf [45] because these two systems have the same generalized type-structure as Automath, and the same way to represent reasoning, viz. a propositions-as-types way.

Scott sketches a general recursive construction mechanism that allows the definition of the natural numbers from a finite set of given ground objects. Martin-Löf's introduction of the natural numbers is more like ours: he introduces *zero* and *successor* but additionally he has *recursion* over the natural numbers built-in in his language.

The main difference between built-in arithmetic and arithmetic introduced axiomatically (as in Automath) is that in the case of built-in arithmetic one gets the equations following from the recursive definition of a function for free, i.e. as *definitional equality*. In Automath one can also introduce a constant intended for primitive recursion but the point is that the additional equality axioms, needed to give such a constant its meaning, concern *book equality*, not definitional equality. This limitation also distinguishes Automath from LCF, where

recursive definitions of functions is indeed possible.

Now we come back to the second and the third limitation: that a type must be present before its inhabitants and, that in Automath uniqueness of types holds. These limitations prevent any inductive construction of a type, in a general sense: both the recursive definition of a type, and, even, the construction of a new type consisting of, e.g., a finite number of previously given objects, are impossible. Such previously given objects *have a type* already and it is simply not possible to state as an axiom (neither as an assumption) that such an object *also* belongs to a different type. In AUT-Pi (and in Scott's and Martin-Löf's system as well) there *is* the possibility to construct *binary disjoint unions* of previously given types but, even there, the objects of the old types cannot be identified with the object of the new types: *injection functions* are needed.

5.8 A comparison with generalized functionality

Uniqueness of types seems a good starting point for a comparison with Seldin's system of *generalized functionality* [64]. This is a generalization of Curry's systems of *basic functionality* [25, 26]. Basic functionality has the usual function types $\alpha \to \beta$ (there denoted $F\alpha\beta$), but generalized functionality has the *generalized type-structure* of Automath and the other two systems, above. Actually we took the word "generalized" from Seldin. The product types denoted above as $[x{:}\alpha]\beta$ or $\Pi([x{:}\alpha]\beta)$ or $\prod_{x:\alpha} \beta$ are in Seldin's system written as $G\alpha(\lambda x.\beta)$. This is, including the introduction and elimination rules for G (i.e. our abstraction rules) all quite similar to the product types of Automath.

However, an important difference is that in Seldin's system the variables do not get a fixed type and consequently, the system rather must be viewed upon as a system of type assignment to (certain) terms of the type free λ-calculus. E.g. the identity I belongs to every type $\alpha \to \alpha$ (where α is a type), whereas in Automath we have different I_α's, denoted $[x{:}\alpha]x$, at every type α. Consequently, a term can indeed belong to different types.

In functionality theory the statement *A has a type B* is denoted *BA* (the *predicate B* applies at the *subject A*, as one says) and is itself an object (*ob*) of the system. In principle, interference of *B* and

A (by reduction, where B acts as a function, with argument A) is not excluded. However, in the *separated* systems, where the equality rules operate on subject and predicate separately, the interference is forbidden and BA is just an alternative notation of our $A \in B$. (Notice that this kind of interference in the case of Automath, where (except in AUT-Pi) $[x{:}A]B$ can be both a function and a type, would be disastrous.) A point of difference between Seldin's system [64] and our systems is that the type formation rather belongs to his meta-language (and is less restricted then ours: he just respects the arity (i.e. number of arguments) of the type valued functions). Seldin proves for his system the *subject reduction theorem* (our *closure* theorem) and the *normal form theorem* (our *normalization theorem*).

The systems of functionality are said to be systems of *illative (combinatory) logic*. The word "illative" now refers to the presence of other basic constants (viz. F and G) than just the *combinators* (or, alternatively, than just λ-abstraction). Originally, Curry rather meant the word "illative" to stand for *inferential*, i.e. also dealing with the logical part (cf. 5.2) of mathematics. In view of the facts, that the Automath languages are quite similar to functionality systems, and that Automath is indeed intended to represent both the object part and the logical part of mathematics, it seems justified to call Automath a system of *illative combinatory logic* (or rather *illative λ-calculus)*.

5.9 The p-fragment

Recall that the logical part of mathematics (the reasoning) is represented in Automath by a *propositions-as-types* method. The standard way of developing propositions-as-types in the p-fragment of Automath is as follows. The propositions enter as special types (2p-expressions of type prop, where prop is another basic constant, a 1p-expression, that behaves just like type).

We saw that a proposition is true if we have a *realizer*, a 3p-expression in it. A proposition B is *assumed* by introducing a *variable* realizing (i.e. of type) B, and a proposition B is stated as an *axiom* (resp. *axiom scheme*) by introducing a *primitive constant* (resp. primitive constant depending on parameters) realizing B. The *implication* $B \Rightarrow C$ is represented by the function type $B \to C$ (in AUT-68- and AUT-QE-notation

$[x{:}B]C$). Introduction- and elimination rules for ⇒ correspond with the abstraction and application rules of Automath.

The standard development of (classical) logic in Automath starts with the introduction of a primitive 2p-constant con E prop, to represent the contradictory proposition, i.e. *falsum*. Clearly con is intended to remain *empty*. So, the *negation* of a proposition α (i.e. α ⇒ *falsum*) can be represented by $[x{:}\alpha]$con, which we abbreviate by non(α). Hence the *double negation* of α becomes non(non(α))(Q $[x{:}[y{:}\alpha]$con]con). Then, for classical logic, a primitive realizer, called dnl, for the *d*ouble *n*egation *l*aw is introduced by a scheme

$$\alpha \text{ E prop}, a \text{ E non(non}(\alpha)) \vdash \text{dnl}(\alpha,x) \text{ E } \alpha$$

We also promised some *book equality axioms* for giving the expressions of the t-part their meaning. To this end a primitive proposition eq, for book equality between objects of the same type, is introduced by a scheme

$$\alpha \text{ E type}, Q \text{ E } \alpha, b \text{ E } \alpha \vdash \text{eq}(\alpha,a,b) \text{ E prop}$$

together with, e.g., primitive realizers for reflexivity (i.e. in eq(α,*a*,*a*)), symmetry (i.e. to infer eq(α,*b*,*a*) from eq(α,*a*,*b*)) etc.

Predicates are special type-valued, viz. *proposition-valued functions*, formed from propositions by λ-abstraction. In constant with the type-valued functions of the t-fragment (cf. 5.3), predicates are usually *non-trivial* type-valued functions. E.g. the property "being equal to one" on type nt is expressed by the predicate $[x{:}\text{nt}]$eq(nt,one,*x*). The (minimal) type (cf. 2.10) of this predicate is nt → prop, in AUT-QE written $[x{:}\text{nt}]$prop and in AUT-Pi written Π([*x*:nt]prop).

These typical 1p-expressions of AUT-QE and AUT-Pi allow the introduction of *predicate variables* and, hence, the formulation of schemes depending on predicate parameters. An important scheme containing a predicate parameter is the axiom scheme for*induction* over the natural numbers.

If *P* is a predicate on type α (having type α → prop) then the product $\prod_{x:\alpha} P(x)$ (in AUT-Pi this is written Π(*P*), in AUT-QE it is just *P* itself) stands for the proposition $\forall_{x:\alpha} P(x)$. Introduction and elimin-

ation rules for ∀ correspond with the abstraction and application rules of Automath.

5.10 Some comment on the p-part

The above examples illustrate why the formulation of schemes with type-variables (and prop- and predicate-variables) are useful. Otherwise we would have needed e.g. separate dnl's for every proposition, separate book-equalities at every type, and a separate induction axiom for each predicate on type nt. And it also becomes evident why abstraction over degree 2 variables is called *higher order quantification*: proposition and predicate variables are 2-variables and abstraction corresponds to universal quantification. See further sec. 5.12.

By using Automath in this propositions-as-types fashion we get an almost ordinary *many sorted first-order predicate logic*, viz. *over* a pure (or extended) typed λ-calculus. It depends mainly on the axioms concerning falsum what kind of logic we get: *minimal logic* (without axioms), *intuitionistic logic* (with absurdity rule), or *classical logic* (as above, with the double negation law, or the like). Additional constants and axioms can be added for the introduction of further mathematical structures (see, e.g. Jutting [37]).

We wrote that Automath is an *almost* ordinary predicate logic, "almost" because there is one unconventional feature: Expressions for proofs (i.e. realizers) can occur inside the expressions for mathematical objects and for propositions, i.e. mathematical objects and propositions can become dependent on the truth of (other) propositions. Example: Let P be a predicate on type α, let $\exists!x.P(x)$ (how this is defined does not matter here). Then the *axiom of individuals* [37], which is usual in the standard development, introduces a constant (a iota-symbol) $\mathrm{ind}(\alpha,P,t)$ together with the appropriate axioms, for the unique object satisfying P; here t realizes $\exists!x.P(x)$. Of course, $\mathrm{ind}(\alpha,P,t_1)$ and $\mathrm{ind}(\alpha,P,t_2)$ are book-equal. However, *irrelevance of proofs* is needed to make these expressions definitionally equal (cf. 5.2).

In this way implications $\alpha \Rightarrow \beta$ (*generalized implications*, as we say) are formed where β cannot be stated unless α holds, and similarly we can get *generalized conjunctions*. Such propositions are said to belong to *generalized logic* (see [20,37,77]).

The propositions-as-types development of sec. 5.9 is not the only one possible. Alternatively, the propositions can be introduced as *ordinary* types (of type type), or as 3-expressions of a new type bool. Since in the first alternative no distinction is made between propositions and ordinary types (in fact there is no p-fragment, only a t-fragment) the realizers enter the discussion as ordinary objects (constructions) too. This seems to be the proper choice if we want to study constructive foundations. Of course, irrelevance of proofs is out of the question here. The second implementation, where the propositions enter as degree 3 expressions, gives rise to higher order logic. In this case the truth of a proposition B is expressed by a formula $t \in B'$, where B' is an ordinary type (the "*proof-type*" of B) associated with the proposition B. This "proof-type" of B (usually denoted TRUE(B), or $\vdash(B)$ or proof(B)) has to be introduced because B itself is not inhabitable (unless we use AUT-4, see 4.12). In Jutting [37] there is also a development in the bool-style.

5.11 On propositions-as-types

In fact, Automath is not just a predicate logic but rather the *proof system* of a predicate logic, because a formula A of the logic is not expressed *directly* but via a statement of the underlying typed λ-calculus, of the form $t \in A$. So it is reasonable to ask for the decidability of the system: proof systems *have* to be decidable. One might wonder, though, why we took such a peculiar proof system, this formulae-as-types kind of formalization.

Our main point is that the formulae-as-types way of implementing a proof system is a straightforward one. The classical notion of formal proof is: a finite sequence of formulae, each of which is either an axiom or follows from the preceding ones by application of an inference rule. This meagre notion of proof is already decidable but useless for our purposes because the decidability is not feasible. For other purposes as well (proof theory) this notion of proof is considered too uninformative.

The first improvement coming to mind is to provide each formula (let us say: *line*) in the sequence with additional information: (1) a *label* (e.g. a mere line number, or a more expressive identification),

for later reference, (2) some *reason*, some *justification* for that line. The information (2) has to indicate: (a) what inference rule is used for establishing that line, (b) on which previous formulas (indicated by their labels) that inference rule has to operate. The axioms in the sequence do not get a justification but just a flag AXIOM, say. Notice that the justification part of a line can also be conceived as an instruction to operate with the indicated inference rule on the indicated preceding lines. If the proof is correct, the formula part of the line will be the result of this operation.

Another, independent, improvement is to allow proofs *from assumptions*, in natural deduction style. In this case additional information must be given with each line to indicate the context in which it is valid (i.e. the assumptions on which it depends).

The proof system we have now arrived at seems to be a natural one for mechanical proof-checking: each line consists of four parts, a *context part*, an *identifier part*, a *justification part* and a *formula part*. Just a slight generalization leads us to Automath. First, we allow the justification part to be a compound *expression* coding *iterated* use of inference rules. This will save a lot of lines in the proof. Secondly we allow each *theorem* from assumptions and depending on propositional or predicate variables to be used in subsequent lines as a new *derived*, inference rule. This gives the system on the flexibility and generality of ordinary mathematical reasoning.

Still one step has to be made: to recognize that what happens in our proof system is completely parallel with what happens in our typed λ-calculus framework. That making assumptions amounts to introducing variables, that stating axioms amounts to introducing primitive constants, and that deriving theorems can be conceived as introducing defined constants. Finally, the abstraction and application rules of the typed λ-calculus amount to the introduction and elimination rules for implication and universal quantification. Then the abbreviation line

$$a \mathrel{E} A,\ y \mathrel{E} B \star d := D \mathrel{E} C$$

(this is the proper book-and-line format, we would rather write $x \mathrel{E} A,\ y \mathrel{E} B \vdash d(x,y) := D \mathrel{E} C$ or the like) can be understood as "from the assumptions A, B the formula C can be derived by using the compound instruction D; this theorem can be referred to as line d".

So, we can explain formulae-as-types as just a practical way of implementing a proof-checking system. Fitting the proof system into typed λ-calculus gives rise to an unusual interpretation of the E-symbol but there is no harm in that (compare 4.6). The third interpretation of realizers (cf. 5.2) seems appropriate to the above explanation: a realizer is a mere indication that its formula holds.

A completely different question is: would there be any more direct way of representing reasoning via the E- and Q-formulas of the underlying typed λ-calculus of Automath? The answer to this question (no) sheds some light on the particular limitations (see 5.7) of Automath. The first point is that the E- and Q-formulas themselves do not allow any reasoning. The only E-*assumptions* we can make are the typing assumptions for variables, and the only E-*axioms* we can make are the typing axioms for the *primitive constants*. The Q-formulas are even more implicit: Q-assumptions are not allowed *at all*, and the only Q-axioms are the abbreviations. (Scott [62] indicates that allowing Q-formulas for assumptions would spoil the decidability). For the rest, E- and Q-formulas just hold or not: if they do not hold they cannot even be stated as an axiom or as an assumption. Consequently they cannot be negated or used in a reasoning ad absurdum. Then, we might look for another trick (different from propositions-as-types) to represent reasoning. One idea might be to introduce a type of truth-values and to see to it that each proposition (or some object associated to it) would be definitionally equal to a truth value. Another idea might be to introduce a type for the true propositions (or objects associated to them) and a type for the false ones(or objects associated to them). Apart from the fact that these proposals simply are not feasible (just try) they would imply that all propositions would become decidable (because E and Q are so) and that is *not* what we want.

5.12 A comparison with higher order systems

We have mentioned before that abstraction over type-variables is not allowed in Automath. In this respect Automath is distinct from both Martin-Löf's system and Girard's systems. Martin-Löf distinguishes *small types* and *large types*. An example of a small type is the type of the natural numbers, examples of large types are: the type V of small types (like our type) *and* the types which represent propositions (in

the propositions-as-types sense). Now variables ranging over small types can be quantified, but quantification over, e.g., propositional variables is still not permitted, so Martin-Löf's system does not have higher order logic.

However, Martin-Löf's system *is* higher-order in our technical sense (see IV.1.5) because, by his built-in recursion mechanism, a type-valued function, T say, can be defined such that e.g. $T(0) = nt$, $T(n+1) = T(n) \to nt$ (where nt is the type of natural numbers). Then the product $\Pi(T)$ consists of functions with values (numbers, functions, functionals) of arbitrary high *complexity* (Seldin would say *rank*). Note that in Automath such functions of *unbounded* functional complexity cannot be defined: crucial in the recursive definition of T is the presence of the function $\lambda y{:}V.(y \to nt)$ (with y a type-variable!) which takes $T(n)$ to $T(n+1)$.

Girard's systems actually contain higher-order logic, because quantification over all type-variables is admitted. E.g. (we use Automath notation) the object $[\alpha{:}\mathrm{type}][x{:}\alpha]x$ of type $[\alpha{:}\mathrm{type}][x{:}\alpha]\alpha$ can be constructed. In fact Girard would write that $\mathrm{DT}\alpha.\lambda x^{\alpha}.x^{\alpha}$ is of type $\Lambda\alpha.(\alpha \to \alpha)$.

I.6 The contents of this thesis

6.1 This thesis has become a comprehensive volume on results and methods in the language theory of Automath: most of the language theoretical questions, as they are stated above, are treated for most of the current Automath languages.

Since many results are quite technical we often, for better accessibility, give a double exposition. First an informal, heuristic one, to explain the ideas, followed by a more rigorous one with some (sometimes many) technical details. If one likes, one can skip the latter.

Most chapters are almost independent and self-contained: they have their own introductions, definitions are repeated etc. For many results some different proofs are given, and some known theorems from [51] and [70] get new proofs.

The discussion is mainly directed towards the Automath languages and the Automath project. However we think that some results may be of more general interest: to λ-calculus and, by the propositions-as-types isomorphism, to proof-theory.

6.2 This thesis (apart from the introduction) can be divided into three parts: (1) a general, preparatory part in a *type-free* setting (Chs. II and III), (2) a part on *pure* (see 1.10) *typed* systems, with application to AUT-68, AUT-QE and AUT-SL (Chs. IV-VII), (3) a part on the *extended* (1.10) language AUT-Pi (Ch. VIII).

Ch. II deals with the preliminary definitions: *expressions, substitution, reductions, definitional equality*. The expressions are already internally decorated with type labels, but a typing relation is not yet defined and, hence, the types do not restrict the expression formation. Various properties are introduced and discussed in a general setting: *normalization* and *strong normalization, closure, Church-Rosser* and *postponement*. The possible interference of the various kinds of reduction is analyzed, in connection with the latter two properties. Finally the important *reduction-under-substitution lemma* of type-free λ-calculus is proved.

It is advised *not to miss* II.0.4.2: we introduce some handy but slightly unusual notational conventions (in particular on tacit *existential* quantification).

Ch. III deals with the isolated study of one specific kind of reduction, viz. δ-*reduction* (see 4.3). A Church-Rosser proof is given, and various ways of proving strong normalization are indicated. Particularly interesting is De Bruijn's strong normalization proof for δ-reduction, which simply calculates the *maximum length* of a reduction sequence.

6.3 Each of the chapters IV, V, VI is devoted to one specific aspect of the pure typed systems: (strong) normalization, closure and Church-Rosser (cf. 2.7) respectively. Ch. IV starts with an introduction on typed λ-calculus systems in general. Like Nederpelt in [51] we use the following strategy to prove (strong) normalization for our languages: first we introduce a general system of *normable* expressions (for short: a normable system), then we prove (*strong*) *normalization* for this system; finally we prove that both AUT-SL (i.e. Λ) and a liberal, comprehensive version of AUT-QE (including all the current versions of AUT-QE and AUT-68) are normable.

There are given three *new proofs* of *strong* β-*normalization* for normable systems. Because the usual pure first-order (see p. 29) typed systems are clearly normable, these proofs are quite generally applicable.

Like Nederpelt's proof of strong normalization in [51], these proofs are not based on a notion of computability.

Ch. IV also contains the precise definitions of *book*, *context* and *degree*, and there *is* defined a *typing relation* (or rather: a *typing function*). However, in the normable expressions the typing restrictions on the expression formation are not fully respected, but only a weak form of them.

6.4 Ch. V gives a framework (the E-*definition*) for generating the *correct* expressions and formulas of the various Automath languages. It mainly concentrates on the *regular* languages (see 4.5) AUT-QE, AUT-68 and their variants.

Then the *closure proofs* are given: first of AUT-QE with βη-reduction (so without δ) then of some more liberal versions AUT-QE+, AUT-QE* with full reduction. Several *unessential-extension* results are presented. Since the closure proofs of βη(δ)-AUT-QE are technically somewhat complicated, we also indicate how, e.g., β-AUT-QE and βηδ-AUT-68 allow a *simpler* closure proof.

In the last section of Ch. V we prove - anticipating the Church-Rosser result of Ch. VI - the *equivalence* of the E-*definition* with the *algorithmic definition* (see 2.6). Quite some attention is paid to the choice of a *typing function* and a *domain function* for the various languages. Finally we make a few remarks on *practical verification* of Automath languages.

6.5 In Ch. VI we prove the *Church-Rosser property* for the pure Automath languages. In particular we solve the βη-*Church-Rosser problem* caused by the presence of the *type-labels* (which are themselves expressions) inside the abstraction expressions in Automath. Nederpelt [51] first indicated this βη-problem and correctly conjectured that βη-Church-Rosser holds in the correct expressions. Except for the βη-case, the Church-Rosser property for pure systems can be proved in the general, unrestricted expressions (as indicated in Ch. II.6).

In fact, we first prove βη-Church-Rosser for a weak form of η-reduction, just sufficient to cover the η-reductions needed in the verification of Jutting's Landau-translation. Afterwards we tackle full η-reduction.

Resuming, Chs. IV-VI show *that the pure Automath languages satisfy the three desirable properties* (cf. 2.7).

6.6 Ch. VII deals exclusively with the language theory of Nederpelt's Λ (or: AUT-SL). Here our point of departure (in contrast with Ch. V) is the *algorithmic definition*. We introduce the socalled *degree-norm correct* expressions. We show that *closure* and *Church-Rosser* can directly be proved from the algorithmic definition, with the help of the *big tree theorem*. We give two new proofs of this theorem, the first one being a mere extension of the second strong normalization proof of Ch. IV, the second one rather based on the first strong normalization proof in IV and making use of the *book-keeping pairs* from de Vrijer's proof of the big tree theorem for his system $\lambda\lambda$ [70].

Finally we compare various versions of Λ: with and without *constants* (resp. *defined constants*), the *single-line version* and the *book-and-context version* etc.

As regards the three celebrated desirable properties for Λ, Ch. VII just duplicates the Chs. IV-VI.

6.7 Chapter VIII discusses *extended systems*, in particular AUT-Pi. In the first section the additional *type forming operations*: *binary union* ($\oplus$), *disjoint sum* (Σ), *cartesian product* (Π), the additional *term forming operations*: *injection* (i_1 and i_2), *plus* ($\oplus$) and *pairs* ($<\cdot,\cdot>$), and the additional *reductions*: $+$, ε, π, σ are introduced informally, and the connection with *full intuistionistic predicate logic* is exhibited.

We generate AUT-Pi by an E-*definition* and prove the *closure property*. We tackle *strong normalization* as in IV (and VII): we extend the notion of form and define two systems AUT-Pi_0 and AUT-Pi_1 which are *extended normable*. For these systems we prove a variety of strong normalization results. First we show that the methods of IV immediately cover the $\beta\pi\eta\sigma$-case, but that the presence of $+$-reduction requires additional attention (the socalled *dead end set* becomes unmanageable).

Three new proofs for strong $\beta\pi+\eta\sigma$-*normalization* are presented, two of them making use of some additional technical reductions (*permutative* and *improper reductions*), the third one using *computability*. Then these strong normalization results are transferred to AUT-Pi.

However, for *full* (i.e. $\beta\pi+\eta\sigma\varepsilon$-) AUT-Pi the language theory is *not yet finished, full Church-Rosser* is *simply false*, and *full strong normalization we have not been able to settle* (though we strongly believe in it).

6.8 The results of this thesis, even when pertaining to type-free λ-calculus, are derived by *syntactic, combinatorial* methods (in contrast with the model theoretic and recursion theoretic reasoning often used in λ-calculus nowadays).

Another point about methods is, that we have been able to avoid the notion of *residual* (and we don't employ the *underlining method* of Barendregt [2] either). Cf. the reduction-under-substitution lemma in II.11.

Finally we mention that (except in VIII, the last proof) we have *not used* any notion of *computability* or the like in our strong normalization proofs, but have restricted ourselves to a priori elementary methods (cf. IV.1.6.3).

6.9 Now we list some language theoretical subjects which we think to require further attention.

In view of 6.7 a further analysis of the definitional equality in AUT-Pi is needed. In particular a decision procedure is wanted (though not absolutely necessary, see 2.8) that does not rely on Church-Rosser (a suggestion is made in VIII.6.2). Or, alternatively, a new reduction relation may be indicated that generates ε-equality and *does* satisfy Church-Rosser.

Secondly, some more work on the *comparison of languages* would be welcome. E.g. the *precise* connections between AUT-68 and AUT-QE have never been made explicit. Here we do not mean the connections between their rules, but rather between *what can be said* in these languages. To be specific, we think that AUT-QE books can be translated into AUT-68 books, and that AUT-Synt might play a role in this respect as well.

Another point deserving interest is the role of the "*extensional*" reductions η, σ and ε. Notably, we think that these reductions can be avoided by first *translating* (performing η-*expansion* etc.) and afterwards performing the corresponding *introduction-elimination* reductions

β, π and + (compare [37, sec. 4.1.1]). Actually we have tried the η-case but got stuck in technical difficulties with the type-labels.

In VIII.2.7 we describe a *natural extension* of AUT-Pi, which nevertheless causes our treatment of strong normalization to fail hopelessly. This is an interesting point of study too.

Finally we mention some subjects that fall somewhat outside the scope of this thesis but are very important for the actual implementation: (1) *iterated references* etc. (see 3.4), (2) AUT-synt, (3) *strings-and-telescopes*. Work in this direction has been done by Zandleven, De Bruijn, Jutting and Wieringa (see 3.4) but we think that further study is required.

CHAPTER II. MISCELLANEA

Section 0 of this chapter gives some comment on methods (inductive definition and inductive proof) and introduces some notational conventions.

The sections 1-4 form a brief introduction to the various λ-calculus systems considered in this thesis. The sections 5-7 contain some general considerations on the closure property, the Church-Rosser property, (strong) normalization and postponement (for a combination of reductions). Also some results of this kind are stated, and a proof of the $\beta\eta$-Church-Rosser property for untyped λ-calculus is included.

In the sections 8 and 9 the Church-Rosser property and postponement are discussed for the specific reduction relations considered.

Section 10 defines the concept of multiple substitution, and section 11 proves a lemma (the reduction-under-substitution lemma) which has interesting applications in untyped λ-calculus.

II. 0. Preliminaries

0.1. Inductive definitions

Throughout this thesis many notions (predicates and relations) are given by so-called *ordinary inductive definitions*. An ordinary inductive definition of, e.g., the predicate P consists of a *finite* set of *inductive clauses* or *rules* of the form:

"if $P(a_1)$ and $P(a_2)\ldots$ and $P(a_k)$ then $P(\phi(a_1,\ldots,a_k))$" ,

where $k \geq 0$, ϕ is a k-ary operation and $a_1,\ldots,a_k$ are variables. *)

In such an inductive definition it is, without further notice, intended that $P(a)$ holds, only if this follows from iterated application of the rules. We may assume that there is at least one clause with $k = 0$ and ϕ a constant - a starting clause -. We say that P is inductively generated from the starting clauses by closure under the other clauses.

It will be clear how inductive definitions of binary relations, or of several notions simultaneously have to be interpreted. With inductive definitions of (partial) functions, we have to be more careful, of course.

---

*) In fact, the definition of *computability* in VIII.5.3 is of a more general nature.

0.2. Inductive proofs

Let $<$ be a partial order and let $<$ be well-founded, i.e. there are no infinite (strictly) descending sequences $a_1 > a_2 > \ldots$. Call b a *descendant* of a if $a > b$; b is a *direct descendant* of a if $a > b$ and there is no c in between. If we can show, for all b,

$$(\forall_{a<b} P(a)) \Rightarrow P(b)$$

then we can conclude $\forall_a P(a)$. This is called *proof by induction on* $<$.

If there are no infinite (strictly) increasing, bounded above, sequences $a1 < a2 < \ldots < b$ either, then for all b, b is either an endpoint - i.e. minimal with respect to $<$ - or b has a direct descendant. So, in this case, if for all b, c,

$$b \text{ endpoint} \Rightarrow P(b) \ ,$$

and

$$(P(b) \wedge b \text{ direct descendant of } c) \Rightarrow P(c)$$

then $\forall_a P(a)$. This principle of proof is also induction on $<$.

Call $<$ finitary, if each a has only a finite number (possibly zero) of direct descendants. If $<$ is finitary and well-founded and has no infinite increasing, bounded above, sequences, then by the lemma of Brouwer-König, for each a there is a maximum to the length of descending sequences starting in a. Call this maximum $\theta(a)$. Then the various inductive proofs of $P(a)$ can simply be reduced to mathematical induction, viz. to induction on $\theta(a)$.

0.3. Induction on definitions

Let P be given by an ordinary inductive definition. If, for each clause in the definition of P, as above,

$$(Q(a_1) \wedge Q(a_2) \wedge \ldots \wedge Q(a_k)) \Rightarrow Q(\phi(a_1,\ldots,a_k))$$

then, clearly, $P(a) \Rightarrow Q(a)$ for arbitrary a.

This kind of inductive proofs can be considered as proofs by induction on the finitary, well-founded partial order generated by the definition of P (in fact, this order pertains to the objects *a labelled with* a derivation of $P(a)$. The a_i (with labels) are the direct descendants of $\phi(a_1,\ldots,a_n)$ (with its label)).

We shall speak about proofs *by induction on* P, or *over* P or *on the length of proof of* $P(a)$.

0.4. Notational conventions

0.4.1. Syntactic variables

Syntactic variables are the variables of our meta-language, denoting syntactical objects such as, e.g., the expressions of an Automath language. Often we reserve some specific syntactic variables (possibly indexed or primed) to denote exclusively objects of a specific syntactic category. E.g. Σ, Γ denote expressions, x, y denote variables, $\mathcal{B}$ denotes books etc.

0.4.2. Logical symbolism

We freely include logical symbols in our meta-language, to shorten and to clarify the discussion. As an example of our notational conventions concerning the logical symbolism consider:

$$A \geq B,\ \ A \geq C \Rightarrow B \geq D,\ \ C \geq D$$

the so-called Church-Rosser property. Written out in full, it would read

$$\forall_A\ \forall_B\ \forall_C((A \geq B \wedge A \geq C) \Rightarrow \exists_D(B \geq D \wedge C \geq D))\ .$$

So, the conventions are:

(i) $\Rightarrow$ binds loosely, the comma denotes $\wedge$

(ii) free variables are tacitly quantified: by an *existential* quantifier if their first occurrence shows up after the main $\Rightarrow$-symbol, otherwise by a universal quantifier.

0.4.3. Reasoning about inductive definitions

Let P be a predicate given by an ordinary inductive definition. Let $\Phi_1,\ldots,\Phi_m$ and $\Psi_1,\ldots,\Psi_n$ be additional inductive clauses for P. Let P' be generated by adjoining $\Phi_1,\ldots,\Phi_m$ to the definition of P (so clearly $\forall_x$ (P(x) $\Rightarrow$ P'(x))). We say that $\Phi_1,\ldots,\Phi_m$ are *derived rules* of P if $\forall_x$ (P(x) $\Leftrightarrow$ P'(x)).

Let P" be generated by adjoining $\Psi_1,\ldots,\Psi_n$ to the definition of P. Then, the rules $\Psi_1,\ldots,\Psi_n$ are derived rules of P' if and only if $\forall_x$ (P"(x) $\Rightarrow$ P'(x)). As an easy shorthand notation for this situation we write (sic)

$$\Phi_1,\ldots,\Phi_m \Rightarrow \Psi_1,\ldots,\Psi_n$$

(i.e, by adjoining $\Phi_1,\ldots,\Phi_m$, the rules $\Psi_1,\ldots,\Psi_n$ become derived rules)

II.1. Expressions

1.1. Here we define our universe of discourse, the expressions of generalized typed λ-calculus. The expressions are formed from *variables* and *constants* using various operations such as *abstraction, application* etc. We take (as in de Bruijn [10]) λ as our only variable binding operation and denote the other operations by so-called *basic* constants, such as abstr, appl etc.

1.2. Variables and constants

The constants are distinguished in *basic* or *language* constants and the *book* constants. The latter fall apart in *primitive* and *defined* constants. All constants have a certain *arity*, the number of arguments going with them. The arity of a constant f is denoted $|f|$.

There is only a small number of basic constants, as listed below

arity 0 :	type,	prop			
arity 1 :	prod,	sum,	proj1,	proj2	
arity 2 :	appl,	abstr,	plus,	inj1,	inj2
arity 3 :	pair				

In contrast with this, any alphanumeric string can serve as a variable or a book constant. The syntactic categories: variables, primitive constants, defined constants, and basic constants, are assumed to be mutually disjoint.

We use x,y,z,u,v as syntactic variables for variables, f for constants, c for book constants, p, q for primitive constants, d for defined constants and $\Sigma,\Gamma,\Delta,..,A,B,C,..,\alpha,\beta,\gamma,...$ as syntactic variables for expressions.

1.3. The expressions are inductively defined:

(i) variables: x is an expression

(ii) λ-expressions: $\lambda x \cdot \Sigma$ is an expression

(iii) constant expressions: 1. $|f| = 0 \Rightarrow f$ is an expression

2. $|f| = k \Rightarrow f(\Sigma_1,...,\Sigma_k)$ is an expression

1.4. Various systems of expressions can be defined inside this framework by specifying the set of (basic) constants. Thus we have *free*,

i.e. *untyped* λ-calculus with appl as its only constant, the *abbreviation calculus* LSP (Ch. III) with book constants only and, of course, the Automath languages.

In the latter languages, the λ-expressions are not present as such, but only inside *abstraction expressions*: abstr $(\Sigma_1, \lambda x \cdot \Sigma_2)$. And only such abstraction expressions abstr (Σ_1, Σ_2) are allowed where Σ_2 is a λ-expression.

The Automath languages AUT-68, AUT-QE and Λ have type (and possibly prop), abstr and appl as their only basic constants, and are called the *pure* Automath languages. Besides these basic constants, AUT-Pi has all the additional operations mentioned, such as prod, sum, plus, inj1 etc.

1.5. We use the ordinary Automath notations:

τ for type, π for prop, Π for prod and Σ for sum

$\{A\}B$ for appl(B,A) , $[x{:}A]B$ for abstr$(A,\lambda x \cdot B)$,

$A_{(1)}$ for proj1(A), $A_{(2)}$ for proj2(A), $\langle A,B,C\rangle$ for pair(A,B,C)

$i_1(A,B)$ for inj1(A,B), $i_2(A,B)$ for inj2(A,B)

and $A \oplus B$ for plus(A,B)

In free λ-calculus simple juxtaposition is used to denote application: BA for $\{A\}B$.

1.6. In $\{A\}B$ we call A the *argument part* and B the *function part*. In $[x{:}A]B$ we call A the *domain part* and B the *value part*.

The domain part A of $[x{:}A]B$ and further: the A of $\langle A,B,C\rangle$, the B of $i_1(A,B)$ and the B of $i_2(A,B)$ are just *type-labels*, present in order to fix the type of the expression. For an explanation we refer to I.4.2 and VIII.1.3. In case we are not interested in the type of the expression, we simply leave out the type-labels, writing $[x]B$, $\langle B,C\rangle$ $i_1(A)$, $i_2(A)$ respectively.

The symbol ⊕ is assumed to have less binding power than the other symbols for expression formation. Additional parentheses are inserted whenever useful to avoid ambiguity.

1.7. Strings

Expression strings $\Sigma_1,\ldots,\Sigma_k$ are denoted by $\bar{\Sigma}$, variable strings $x_1,\ldots,x_k$ by $\bar{x}$. The empty string is not a priori excluded. The *multiplicity* of a string $\Sigma_1,\ldots,\Sigma_k$ is k and is denoted by $|\bar{\Sigma}|$. So we can rephrase clause 1.3.(iii)2 by

$$|f| = |\bar{\Sigma}| \Rightarrow f(\bar{\Sigma}) \text{ is an expression}$$

Further, if $|\bar{A}| = k$, $|\bar{x}| = k$ then
$\{\bar{A}\}B$ is shorthand for $\{A_k\}\ldots\{A_1\}B$, $B\bar{A}$ for $(\ldots(BA_1)\ldots A_k)$ and $[\bar{x}:\bar{A}]B$ for $[x_1:A_1]\ldots[x_k:A_k]B$.

Sometimes, by abuse of notation, we treat variable strings as sets, writing, e.g. $y \in \bar{x}$ instead of : y is among $x_1,\ldots,x_k$, etc.

1.8. Length, subexpressions

In agreement with 0.3, induction on the definition 1.3 is called induction *on expressions* or, also, on the *structure* of expressions. Counting variables and constants as single *atomic* symbols, the *length* $\ell(\Sigma)$ of an expression Σ can be defined by:

$$\ell(x) = 1,\ \ell(\lambda x{\bullet}\Sigma) = \ell(\Sigma) + 1,\ \ell(f(\bar{\Sigma})) = 1 + \sum_{i=1}^{k} \ell(\Sigma_i)\ .$$

Similarly, Γ is said to be a *subexpression* of Σ, for short $\Gamma \subset \Sigma$, according to the following inductive definition:

(i) $\quad \Sigma \subset \Sigma$

(ii) $\quad \Gamma \subset \Sigma \Rightarrow \Gamma \subset \lambda x{\bullet}\Sigma$

(iii) $\quad \Gamma \subset \Sigma_i \Rightarrow \Gamma \subset f(\Sigma_1,\ldots,\Sigma_i,\ldots,\Sigma_k) \quad (i = 1,\ldots,k)$.

Clearly, $\subset$ is a partial order. We say that Σ is a *direct subexpression* of $\lambda x{\bullet}\Sigma$ and that Σ_i is a *direct subexpression* of $f(\Sigma_1,\ldots,\Sigma_k)$.

We want that the Automath expressions are closed under taking subexpressions. So, when discussing these, instead of (ii) we include (ii')

(ii') $\quad \Sigma \subset A$ or $\Sigma \subset B \Rightarrow \Sigma \subset [x:A]B$

and we restrict clause (iii) to constants f different from abstr. In this case A and B are the direct subexpressions of $[x:A]B$.

1.9. Occurrences, suggestive dots

If $\Sigma \subset \Gamma$, then Σ can have several *occurrences* inside Γ. Such occurrences can be distinguished by their positions inside Γ, e.g. like in Nederpelt [51, p.18]. We shall treat occurrences in an informal way. Two occurrences are *disjoint* if they have no occurrences of symbols in common.

Often, to denote an arbitrary expression with one or possibly more specific occurrences of a subexpression Σ we write:

$$\ldots\Sigma\ldots \text{ , resp. } \ldots\Sigma\ldots\Sigma\ldots$$

The meaning of these *suggestive dots* will be clear from the context.

We formulate the *fundamental property of subexpressions* in terms of suggestive dots: if $\ldots\Sigma\ldots\Gamma\ldots$ is an expression then one of the following alternatives holds

(i) Σ and Γ disjoint, or (ii) $\Sigma \subset \Gamma$, or (iii) $\Gamma \subset \Sigma$.

Notice that these cases do not exclude each other.

II.2. Syntactic identity, α-equality and substitution

2.1. Free and bound variables

The *free variables* and the *binding variables* of an expression can be defined informally, as follows:

(i) the first occurrence of x in $\lambda x \cdot \Sigma$ is called a *binding occurrence*; Σ is called the *scope* of the binding x.

(ii) an occurrence of x, not being a binding occurrence, is called *free* if it does not fall inside the scope of a binding x.

(iii) a free occurrence of x in Σ is called *bound* in $\lambda x \cdot \Sigma$ (*by* the *binding* x)

(iv) x is a free variable of Σ (resp. a binding variable of Σ) if there is a free (resp. binding) occurrence of x in Σ.

The set of free variables of Σ is called $FV(\Sigma)$. If we write $\ldots x\ldots x\ldots$, we intend an expression with some free occurrences of x. For a string $\bar{\Sigma}$, $FV(\bar{\Sigma}) = \bigcup FV(\Sigma_i)$.

2.2. Syntactic identity and α-equality

By ≡ we denote *syntactic identity*, i.e. symbol-for-symbol-equality, of expressions, modulo α-*equality*, i.e. renaming of bound variables. So a *name-carrying* expression is considered to *represent* a certain *name free* skeleton - or, alternatively, an equivalence class of α-equal name-carrying expressions -. Our point of view,*) viz. of simply identifying $\ldots(\lambda x \cdot \ldots x \ldots x \ldots) \ldots$ and $\ldots(\lambda y \cdot \ldots y \ldots y \ldots) \ldots$ can be justified by referring to Curry [25], Nederpelt [51] or de Bruijn [10]. The latter reference gives a treatment of a formalism of nameless dummies (see I.3), which is actually used in the currently implemented verifier for Automath languages.

The notation ≡ extends to strings: $\bar{\Sigma} \equiv \bar{\Gamma}$, if $|\bar{\Sigma}| = |\bar{\Gamma}|$ and, for $i = 1,\ldots,|\bar{\Sigma}|$, $\Sigma_i \equiv \Gamma_i$. Further, $\Sigma \not\equiv \Gamma$ means: not $(\Sigma \equiv \Gamma)$, and similarly for strings.

2.3. Now that we have introduced ≡ we return to the notion of subexpression. We say that Σ is a *proper subexpression* of Γ, for short Σ sub Γ, if $\Sigma \subset \Gamma$ and $\Sigma \not\equiv \Gamma$. Clearly, sub is the transitive relation, inductively generated by the relation ... is direct subexpression of ... . We have such properties as:
$\Sigma \subset \Gamma$, Γ a variable or constant $\Rightarrow \Sigma \equiv \Gamma$

And we can make the fundamental property of subexpressions more precise: if $\Sigma \subset \Delta$, $\Gamma \subset \Delta$ then precisely one of the following alternatives holds: (i) Σ and Γ disjoint, (ii) Σ and Γ are the same occurrence (so $\Sigma \equiv \Gamma$), (iii) Σ sub Γ, or (iv) Γ sub Σ.

2.4. Substitution

By $\Sigma[\![x/A]\!]$ we denote the result of *substituting* the expression A for all free occurrences of x in Σ. Similarly by the operator $[\![\bar{x}/\bar{A}]\!]$ we denote *simultaneous* substitution of A_i for the free occurrences of x_i, for $i = 1,\ldots,k$ (where $k = |\bar{x}| = |\bar{A}|$ and all x_i are mutually distinct). The notation extends to strings in a straightforward way. One has to take care that no free variables of the substituted expressions come under the "wrong influence" and become bound after substitution.

For definiteness we give the definition of simultaneous substitution. Let Σ^* locally abbreviate $\Sigma[\![\bar{x}/\bar{A}]\!]$. Then by induction on Σ, we

*) Actually in Chs. IV, VII and VIII there are used certain methods which are not completely compatible with this approach.

define Σ^*, as follows:

(i) a. $y \equiv x_i \Rightarrow y^* :\equiv A_i$

b. $y \notin \bar{x} \Rightarrow y^* :\equiv y$

(ii) $y \notin \bar{x}$, $\forall_{i=1,\ldots,|\bar{x}|}(x_i \in FV(\Sigma) \Rightarrow y \notin FV(A_i)) \Rightarrow$
$(\lambda y \cdot \Sigma)^* :\equiv \lambda y \cdot \Sigma^*$ - otherwise rename y in $\lambda y \cdot \Sigma$ -

(iii)a. $f^* :\equiv f$

b. $f(\bar{\Sigma})^* :\equiv f(\bar{\Sigma}^*)$.

Single substitution $[\![x/A]\!]$ amounts to the case $|\bar{x}| = 1$ above.

Sometimes, if the $\bar{x}$ are not relevant or clear from the context, then we write

$\Sigma[\![\bar{A}]\!]$ instead of $\Sigma[\![\bar{x}/\bar{A}]\!]$.

2.5. Two fundamental substitution properties

Substitution property I: If all free variables of Σ are among $\bar{y}$ then

$$\Sigma[\![\bar{y}/\bar{B}]\!][\![\bar{x}/\bar{A}]\!] \equiv \Sigma[\![\bar{y}/\bar{B}\ [\![\bar{x}/\bar{A}]\!]]\!]$$

Substitution property II: If no free variables of $\bar{A}$ are among $\bar{y}$ and $\bar{x}$ and $\bar{y}$ have no variables in common, then

$$\Sigma[\![\bar{y}/\bar{B}]\!][\![\bar{x}/\bar{A}]\!] \equiv \Sigma\ [\![\bar{x}/\bar{A}]\!][\![\bar{y}/\bar{B}[\![\bar{x}/\bar{A}]\!]]\!]$$

Both proofs are by induction on Σ. To illustrate I (in the case of single substitution), let $\Sigma \equiv \ldots y \ldots$. Then $\Sigma[\![y/B]\!] \equiv \ldots B \ldots \equiv \ldots(\ldots x \ldots)\ldots$ and there are no free variable occurrences outside B. So $\Sigma[\![y/B]\!][\![x/A]\!] \equiv \ldots(\ldots A \ldots)\ldots \equiv \Sigma[\![y/B[\![x/A]\!]]\!]$ q.e.d. And to illustrate II, (in the case of single substitution the conditions read: $y \notin FV(A)$ and $y \neq x$), let $\Sigma \equiv \ldots y \ldots x \ldots$. Then $\Sigma[\![y/B]\!] \equiv \ldots B \ldots x \ldots \equiv \ldots(\ldots x \ldots)\ldots x \ldots$, $\Sigma[\![y/B]\!][\![x/A]\!] \equiv \ldots(\ldots A \ldots)\ldots A \ldots$. Further $\Sigma[\![x/A]\!] \equiv \ldots y \ldots A \ldots$ and $\Sigma[\![x/A]\!][\![y/B[\![x/A]\!]]\!] \equiv \ldots(\ldots A \ldots)\ldots A \ldots$ q.e.d.

2.6. Substitution and subexpressions

Let, again, $\Sigma^* \equiv \Sigma[\![\bar{x}/\bar{A}]\!]$. Then of course, if $\Sigma \equiv \ldots \Gamma \ldots$ then $\Sigma^* \equiv \ldots \Gamma^* \ldots$. And about the "converse" question: where do occurrences

of subexpressions in Σ^* arise from? Let $\Sigma^* \equiv \ldots\Gamma\ldots$. Then precisely one of the following alternatives holds:

(i) $\Sigma \equiv \ldots\Gamma_0\ldots$, $\Gamma_0^* \equiv \Gamma$, *for some* $\Gamma_0 \subset \Sigma$, or

(ii) $\Sigma \equiv \ldots x_i\ldots$, $\Sigma^* \equiv \ldots A_i\ldots \equiv \ldots(\ldots\Gamma\ldots)\ldots$, Γ sub A_i for some i.

(I.e. Γ occurs as a proper subexpression inside one of the substituted occurences A_i).

If, e.g., $\Gamma \equiv f(\bar{\Delta})$ then (i) specializes to:

(i)a. $\Sigma \equiv \ldots f(\bar{\Delta}_0)\ldots$, $\bar{\Delta}_0^* \equiv \bar{\Delta}$, or

(i)b. $\Sigma \equiv \ldots x_i\ldots$, $\Gamma \equiv A_i$.

II.3. Elementary and one-step reductions

3.1. The relations of *definitional equality* of expressions will be defined inductively. We start with *elementary reductions*, then define *one-step reductions*, proceed to *more-step reductions* and finally to definitional equality. Since we only discuss purely syntactical aspects here, all these relations are defined on the full universe of expressions.

3.2. Elementary reductions

3.2.1. β- and η-reductions

These are the usual λ-calculus reductions, associated with the basic constants **abstr** and **appl**.

β: $\{A\}[x{:}B]C$ elementary reduces to $C[\![A]\!]$

η: $[x{:}B]\{x\}C$ elementary reduces to C , if $x \notin \mathrm{FV}(C)$

In free λ-calculus, with the alternative notations, these elementary reductions read

β: $(\lambda x{\cdot}C)A$ elementary reduces to $C[\![A]\!]$

η: $\lambda x{\cdot}Cx$ elementary reduces to C if $x \notin \mathrm{FV}(C)$

3.2.2. π- and σ-reductions

These reductions are associated with pair and proj1, proj2. Here π is intended to suggest "projection" and σ stands for "surjectivity of pairing", after Barendregt [3].

π: $\langle A,B\rangle_{(1)}$ el. red. to A

$\langle A,B\rangle_{(2)}$ el. red. to B

σ: $\langle A_{(1)},A_{(2)}\rangle$ (or, with type-label, $\langle B,A_{(1)},A_{(2)}\rangle$)

el. red. to A (However, see VIII.2.5.1.)

3.2.3. +- and ε-reductions

These reductions are associated with plus and inj.

+: $\{i_1(A)\}(B \oplus C)$ el. red. to $\{A\}B$

$\{i_2(A)\}(B \oplus C)$ el. red. to $\{A\}C$

ε: $([x{:}A]\{i_1(x,D)\}B) \oplus ([x{:}C]\{i_2(x,E)\}B)$ el. red. to B, if $x \notin FV(B)$.

As an alternative version of +, suitable for the case where all plus-expressions are of the form $[x{:}A]B \oplus [y{:}C]D$, we have (this is + combined with β)

+': $\{i_1(E,F)\}([x{:}A]B \oplus [x{:}C]D)$ el. red. to $B[\![E]\!]$, etc..

In the chapter on AUT-Pi, some further reductions connected with ⊕ will be introduced, the *permutative reductions*.

3.2.4. δ-reduction

Here δ is intended to suggest *"definitional"*. This reduction is of course associated with defined constants, for which a *defining axiom* is given.

δ: $d(\bar{\Sigma})$ el. red. to $\Delta[\![\bar{x}/\bar{\Sigma}]\!]$,

if d is a defined constant with defining axiom $d(\bar{x}) := \Delta$ - where $FV(\Delta) \subset \bar{x}$ -.

This kind of δ- or definitional reductions must not be confused with Curry's δ-reduction [25] , Church's δ (in Barendregt et al.[5]), or the δ-reduction proposed in Staples [65].

3.3. In all the definitions of elementary reductions above, the left hand side is called *redex* and the right hand side is called the *contractum* of the reduction. Elementary reductions are also called *contractions*.

We use some terminology like in Prawitz' theory of natural deduction systems [59]: abstr and pair are the *negative*, and inj1, inj2 are the *positive introduction* operations. Further appl, proj1 and proj2 are the *elimination* operations.*) Correspondingly, β-, π- and +-reductions are called the *introduction-elimination* (I.E.) reductions. The reductions η, σ and ε are called the *extensional* (ext) reductions.

3.4. One-step reductions

We consider three kinds of one-step reductions $>$, generated inductively from the elementary reductions by certain *monotonicity* rules. A subscript or a combination of subscripts indicates which of the elementary reductions are included. E.g. $>_{\beta\eta\delta}$ is a one-step reduction generated from elementary β-, η- and δ-reduction. The three kinds of one-step reductions differ by the monotonicity rules used in their definitions.

For $>$, and the other relations between expressions, defined here, the notation extends in a straightforward way to strings. E.g. $\bar{\Sigma} > \bar{\Gamma}$ if $|\bar{\Sigma}| = |\bar{\Gamma}|$ and, for $i = 1,\ldots,|\Sigma|$, $\Sigma_i > \Gamma_i$.

We define $\Sigma > \Sigma'$ by induction on the structure of Σ. First, *ordinary* one-step reduction has the following clauses

(i) if Σ elementary reduces to Σ' then $\Sigma > \Sigma'$

(ii) if $\Sigma > \Sigma'$ then $\lambda x\cdot\Sigma > \lambda x\cdot\Sigma'$

(iii) if $\Sigma_i > \Gamma$ then $f(\Sigma_1,\ldots,\Sigma_i,\ldots,\Sigma_k) > f(\ldots,\Sigma_{i-1},\Gamma,\Sigma_{i+1},\ldots)$ $(i = 1,\ldots,k)$.

Secondly, the *disjoint* one-step reduction has an additional clause

(0) $\Sigma > \Sigma$,

and instead of (iii)

(iii') if $\bar{\Sigma} > \bar{\Sigma}'$ then $f(\bar{\Sigma}) > f(\bar{\Sigma}')$

Finally, the *nested* one-step reduction has the clause (0) - reflexivity -, the *monotonicity rules* (ii) and (iii') - just like the disjoint one-step reduction -, but instead of (i) it has (i'), with inductively given elementary reductions:

*) The operation plus falls somewhat out of this classification.

(i'): β: $A > A'$, $C > C' \Rightarrow \{A\}[x:B]C > C'[\![A']\!]$

η: $C > C'$, $x \notin \mathrm{FV}(C) \Rightarrow [x:B]\{x\}C > C'$

- and similarly in free λ-calculus -

π: $A > A'$, $B > B' \Rightarrow \langle A,B\rangle_{(1)} > A'$, $\langle A,B\rangle_{(2)} > B'$

σ: $A > A' \Rightarrow \langle A_{(1)}, A_{(2)}\rangle > A'$

$+$: $A > A'$, $B > B'$, $C > C' \Rightarrow$

$\{i_1(A)\}(B \oplus C) > \{A'\}B'$, $\{i_2(A)\}(B \oplus C) > \{A'\}C'$

ε: $B > B'$, $x \notin \mathrm{FV}(B) \Rightarrow ([x:A]\{i_1(x)\}B \oplus [x:C]\{i_2(x)\}B) > B'$

δ: if d is a defined constant with defining axiom $d(\bar{x}) := \Delta$ ($\mathrm{FV}(\Delta) \subset \bar{x}$) then $\bar{\Sigma} > \bar{\Sigma}' \Rightarrow d(\bar{\Sigma}) > \Delta[\![\bar{x}/\bar{\Sigma}']\!]$

3.5. If $\Sigma > \Gamma$ and actually some contractions take place in the reduction step (e.g. when it is an ordinary one-step reduction) then Γ is a *direct reduct* of Σ. By induction on Σ it appears that: (1) the set of direct reducts of Σ is finite (provided there are only finitely many defining axioms for each defined constant) and effectively constructible, so certainly (2) $\Sigma > \Gamma$ is decidable.

3.6. The disjoint and the nested one-step reductions are so-called *compound* (after Curry) or *special* (Nederpelt [51]) one-step reductions. Troelstra [69] speaks about "clever counting of contractions".

The terminology can be explained as follows: whereas ordinary one-step reduction contracts precisely one redex, both special reductions allow to contract *several* (possibly: *none*) redices at a time. In the "disjoint case" these simultaneously contracted redices have to be disjoint, but in the "nested case" they may also occur inside each other, i.e. nested.

3.7. Let, if ρ is a reduction relation, $\bar{\rho}$ denote the "disjoint version" of ρ, i.e. the closure of ρ under (0), (ii) and (iii') and let $\tilde{\rho}$ denote the nested version of ρ, generated by (0), (i'), (ii) and (iii').

Let us write $>_1$ for ordinary one-step reduction. Then disjoint one-step reduction is $\bar{>}_1$ and nested one-step reduction is $\tilde{>}_1$. Clearly,

(0), (i') $\Rightarrow$ (i)

i.e. if an inductive definition contains the rules (0) and (i'), then (i) is a *derived rule*. And, under the same interpretation

$$(0),\ (iii') \Rightarrow (iii)$$

So, we have:

$$>_1 \Rightarrow \bar{>}_1 \Rightarrow \tilde{>}_1 .$$

And, since closing once more under a rule has no effect

$$\bar{=}_1 \leftrightarrow \bar{>}_1 \text{ and } \quad \tilde{\approx}_1 \leftrightarrow \bar{\tilde{>}}_1 \leftrightarrow \tilde{\bar{>}}_1 \leftrightarrow \tilde{>}_1$$

3.8. Substitution and one-step reduction

The point of the special reductions lies in their behaviour under substitution. For each of the one-step reductions, we have property I:

I: $\quad B > B' \Rightarrow B[\![A]\!] > B'[\![A]\!]$

Proof: By induction on $B > B'$, using the substitution properties I and II in the case of δ- and β-contractions respectively. □

And, property II:

II: $\quad \bar{A} > \bar{A}' \Rightarrow B[\![\bar{A}]\!] \ \bar{>} \ B[\![\bar{A}']\!]$

Proof: By induction on B. Notice that possibly several substituted occurrences of A_i (which are disjoint) have to be contracted. □

So, by 3.7, we have

III: $\quad \bar{A} \ \bar{>}_1 \ \bar{A}' \Rightarrow B[\![\bar{A}]\!] \ \bar{>}_1 \ B[\![\bar{A}']\!]$

Combining the reductions in B and $\bar{A}$, there is property

IV: $\quad \bar{A} > \bar{A}',\ B > B' \Rightarrow B[\![\bar{A}]\!] \ \tilde{>} \ B'[\![\bar{A}']\!]$

Proof: By induction on $B > B'$. In the case of clause (0), use property II and 3.7. □

So, by 3.7 again, we have

V: $\quad \bar{A} \ \tilde{>}_1 \ \bar{A}',\ B \ \tilde{>}_1 \ B' \Rightarrow B[\![\bar{A}]\!] \ \tilde{>}_1 \ B'[\![\bar{A}']\!]$

II.4. Reductions and definitional equality

4.1. Reduction sequences

Let $>$ be a one-step reduction. Then a (possibly infinite) sequence of expressions $\Sigma_1 > \Sigma_2 > \ldots > \Sigma_k > \ldots$ is called a *reduction sequence* of Σ_1 with respect to $>$. Reduction sequences with respect to $>_1$ are *ordinary* reduction sequences. If each Σ_{k+1} in the sequence is a direct reduct of Σ_k then the reduction sequence is a *strict* or *proper* reduction sequence. So, e.g., ordinary reduction sequences are strict.

4.2. Reduction trees

The strict reduction sequences of an expression Σ can be arranged in a (possibly infinite) finitary labelled tree, the *reduction tree* of Σ . We think of reduction trees as growing downward: label the root with Σ, at the first level below come all the direct reducts etc.

4.3. *More-step reduction* (or just: *reduction*), denoted $\geq$, is defined as the transitive and reflexive closure of $>_1$, i.e.:

(i) $\quad \Sigma >_1 \Sigma' \Rightarrow \Sigma \geq \Sigma'$,

(ii) $\quad \Sigma \geq \Sigma$,

(iii) $\quad \Sigma \geq \Sigma', \Sigma' \geq \Sigma'' \Rightarrow \Sigma \geq \Sigma''$.

Again, subscripts going with $\geq$ indicate which elementary reductions are included.

If $\Sigma \geq \Gamma$, Γ is a *reduct* of Σ. Clearly Γ is a reduct of Σ iff either $\Sigma \equiv \Gamma$ or there is an ordinary reduction sequence *from* Σ *to* Γ. In the latter case Γ is a *proper* reduct of Σ.

4.4. Let, if ρ is a relation, ρ^* be its reflexive and transitive closure. So, by definition $\geq$ is just $>_1^*$. Of course, $\geq$ satisfies all the monotonicity clauses:

$$\Sigma \geq \Gamma \Rightarrow \ldots\Sigma\ldots \geq \ldots\Gamma\ldots$$

and

$$\geq \Leftrightarrow \bar{\geq} \Leftrightarrow \tilde{\geq} \Leftrightarrow \geq^*$$

As in 3.8, $\bar{A} \geq \bar{A}'$, $B \geq B' \Rightarrow B[\![\bar{A}]\!] \geq B'[\![\bar{A}]\!]'$

Further,

$$\bar{>}_1 \Rightarrow >_1^* \quad \text{and} \quad \tilde{>}_1 \Rightarrow >_1^*$$

whence

$$\geq \Leftrightarrow >_1^* \Leftrightarrow \bar{>}_1^* \Leftrightarrow \tilde{>}_1^*$$

4.5. We write $\Gamma < \Sigma$ for $\Sigma > \Gamma$, $\Sigma \not> \Gamma$ for not $(\Sigma > \Gamma)$. Similarly for $\geq$.
We define: $\Sigma \downarrow \Gamma :\Leftrightarrow \Sigma \geq \Delta \leq \Gamma$ for some Δ.
So, $\Sigma \downarrow \Gamma$ iff Σ and Γ have a common reduct.

4.6. As usual, the relation $=$ (possibly with subscripts $=_\beta$, $=_{\beta\eta}$ etc.) of *definitional equality* (or just: *equality*) is the equivalence relation inductively generated from $\geq$ (resp. $\geq_\beta$, $\geq_{\beta\eta}$ etc.).

Again, $=$ satisfies all the monotonicity rules:

$$\Sigma = \Gamma \Rightarrow \ldots\Sigma\ldots = \ldots\Gamma\ldots$$

and

$$= \Leftrightarrow (\bar{=}) \Leftrightarrow (\tilde{=}) \Leftrightarrow =^*$$

So, for equality too,

$$\bar{A} = \bar{A}',\ B = B' \Rightarrow B[\![\bar{A}]\!] = B'[\![\bar{A}']\!]$$

Clearly, $=$ is just $\downarrow^*$. I.e. $\Sigma = \Gamma$ if for some $k \geq 0$ and some $\bar{\Sigma}$,

$$\Sigma \equiv \Delta_0 \downarrow \Delta_1 \ldots \downarrow \Delta_k \equiv \Gamma$$

4.7. In some cases we rather consider a restricted form of $=$. Let A be a set of expressions. Then, we define, for $\Sigma \in A$, $\Gamma \in A$,

$$\Sigma \sim_A \Gamma :\Leftrightarrow \Sigma \downarrow \Delta_1 \downarrow \ldots \downarrow \Delta_k \downarrow \Gamma \quad \text{for some } \Delta_1 \in A, \ldots, \Delta_k \in A.$$

So, if $>_A$ and $\downarrow_A$ are the restrictions of $>$ and $\downarrow$ to A, respectively, then

$$(>_A)^* \Rightarrow (\downarrow_A)^* \Leftrightarrow \sim_A$$

4.8. The relations $=$, $\vdash$ and $\geq$ (and, if A is recursively enumerable, $\sim_A$ and $\vdash_A$) are, in view of the recursivity of $>$, by their definitions recursively enumerable, and, in contrast with $>$, not a priori decidable.

Indeed, in free λ-calculus equality and reduction are not recursive (Scott, in Barendregt [4]). Below we shall introduce some properties which imply the decidability of the various notions.

4.9. An ordinary reduction sequence $\Sigma \equiv \Delta_0 > \Delta_1 > \ldots > \Delta_k \equiv \Gamma$ is a *main reduction sequence* if at least one of the steps $\Delta_i > \Delta_{i+1}$ is an elementary reduction. We say that Σ *main reduces to* Γ, for short $\Sigma \geq_{MR} \Gamma$. If for $j < k$, the reduction sequence from Σ to Δ_j is not main, then Γ is called *a first main reduct* of Σ.

It is just the main reductions that affect the "outside form" of expressions: if f_1 and f_2 are distinct constants and $f_1(\bar{\Sigma}) \geq f_2(\bar{\Gamma})$ then $f_1(\bar{\Sigma}) \geq_{MR} f_2(\bar{\Gamma})$.

Expressions (and their "leading" constants, such as f_1 in $f_1(\bar{\Sigma})$) are said to be *immune* if they do not main reduce. E.g., the primitive constants, inj1 and inj2 are immune for all, and the defined constants and introduction constants (sec. 3.3.) are immune for I.E. reductions.

II.5. Some important properties

5.1. Below we introduce some important properties, such as closure (CL), strong normalization (SN) and the Church-Rosser property (CR). All these properties (and some connected concepts, such as normal form, length of reduction tree (θ)) are defined relative to a reduction relation $\geq$ (and possibly a one-step reduction $>$). Now, prefixes or subscripts going with the introduced notions indicate what elementary reductions we included in the intended reduction relation. So we speak about β-closure, βδ-SN, βη-CL, $\theta_{\beta\eta\delta}$ etc.

5.2. The closure property

5.2.1. A set A of expressions is *closed* w.r.t. $\geq$ (or just: *closed*), if it satisfies CL, the *closure property* (after Nederpelt):

$$\text{CL: } \Sigma \in A,\ \Sigma \geq \Gamma \Rightarrow \Gamma \in A$$

(do not confuse CL with "combinatory logic")

We also define CL_1, *one-step closure*, for a one-step reduction $>$:

$$CL_1\colon \Sigma \in A,\ \Sigma > \Gamma \Rightarrow \Gamma \in A$$

For each of our one-step reductions $>_1$, $\bar{>}_1$ and $\tilde{>}_1$, we have $CL_1 \Rightarrow CL$.

The crucial point in a proof of CL_1 is often to prove *closure under substitution:*

$$\Gamma \in A,\ \Sigma_1 \in A,\ldots,\Sigma_k \in A \Rightarrow \Gamma[\![\bar{\Sigma}]\!] \in A$$

(in most of the cases additional restrictions on the $\Sigma_1,\ldots,\Sigma_k$ have to be imposed).

5.2.2. Clearly, if A is closed, then $\sim_A$ is precisely the equivalence relation generated by $>_A$ (see 4.7). Proofs by induction on (the definition of) $\geq$ (or on reduction trees, if these are well-founded) require that the system under consideration is closed.

If $\geq$ and $\geq'$ are two reduction relations, $\geq \Rightarrow \geq'$, and A is closed w.r.t. $\geq'$ then A is closed w.r.t. $\geq$.

5.2.3. Let $\bar{f}$ be a string of constants. Call Σ an $\bar{f}$-expression if the constants of Σ are among $\bar{f}$. The $\bar{f}$-expressions are closed under substitution, so they satisfy CL_1 (provided that the defining axioms do not contain constants outside $\bar{f}$), so they satisfy CL. Similarly, the full universe of expressions is closed under substitution (as we already tacitly assumed) so it is CL. Free λ-calculus, and the various systems of Automath expressions are CL too (sec. 1.4).

Clearly, the set of reducts of an expression is closed. In chapter IV , we prove that the so-called *normable* expressions form a closed set. In chapter V and VIII we prove that various systems of so called *correct* Automath expressions are closed.

5.3. Normalization and strong normalization

5.3.1. We define (relative to a reduction relation)

(i) Σ *is in normal form* (or just: *normal*) if not $\Sigma >_1 \Gamma$

(ii) Σ *has a normal form* if $\Sigma = \Gamma$ for some normal Γ

(iii) Σ *normalizes* (or just: $N(\Sigma)$) if $\Sigma \geq \Gamma$ for some normal Γ

(iv) Σ *strongly normalizes* (or: $SN(\Sigma)$) if all proper reduction se-

quences of Σ terminate.

(v) A set A of expressions is said to be N (resp.SN) if

$$\Sigma \in A \Rightarrow N(\Sigma) \quad (\text{resp. } SN(\Sigma))$$

5.3.2. Clearly, Σ is normal iff Σ does not reduce properly iff Σ does not contain redices. So the property of being normal is decidable.

Of course, Σ normal $\Rightarrow$ $SN(\Sigma) \Rightarrow N(\Sigma) \Rightarrow \Sigma$ has normal form.

If $SN(\Sigma)$ then the reduction tree of Σ is well-founded, so (by the Brouwer-König lemma) it is finite. Hence, if $SN(\Sigma)$ then we can define $\partial(\Sigma)$ as the length of the reduction tree of Σ, i.e. the maximum length of proper reduction sequences starting in Σ. And, if $SN(\Sigma)$, then the relation $\Sigma \geq \Gamma$ is decidable.

5.3.3. Call a reduction sequence $\Sigma_0 > \Sigma_1 > \ldots$ *secured* if for some k, Σ_k is SN. Then $SN(\Sigma)$ iff all the reduction sequences of Σ are secured iff all the direct reducts of Σ are SN.

By monotonicity, we have: $SN(\Sigma)$, $\Gamma \subset \Sigma \Rightarrow SN(\Gamma)$.
Conversely, if (1) Γ sub $\Sigma \Rightarrow SN(\Gamma)$ and (2) all first main reducts of Σ are SN, then $SN(\Sigma)$ - because all its reduction sequences are secured -.

5.3.4. Let A and A' be sets of expressions, $A \subset A'$. Let $\geq$ and $\geq'$ be reduction relations, with $\geq \Rightarrow \geq'$. Let A' be SN with respect to $\geq'$. Then A is SN with respect to $\geq$ (compare 5.2.2). So, in order to conclude SN for a variety of sets A and reduction relations $\geq$ it is sufficient to prove SN for the "union" of these systems.

As for property N, the implications rather work in the other direction: let $\geq'$ and $\geq''$ be reduction relations, $\geq$ is the "union" of $\geq'$ and $\geq''$. If A is closed w.r.t. $\geq'$, N both w.r.t. $\geq'$ and $\geq''$, and we have: $(\Sigma$ normal w.r.t. $\geq'$, $\Sigma \geq'' \Gamma) \Rightarrow (\Gamma$ normal w.r.t. $\geq')$ then A is N w.r.t. $\geq$.

5.3.5. It is well-known that free λ-calculus does not β-normalize (e.g. consider $B :\equiv AA$ with $A :\equiv \lambda x \cdot xx$) and that not necessarily $N(\Sigma) \Rightarrow SN(\Sigma)$ (e.g. consider $(\lambda y \cdot A)B$).

However, the correct expressions of all the Automath languages do strongly normalize under all the associated reductions: chapter III proves δ-SN, chapter IV deals mainly with β-SN and chapter VIII proves the strong normalization of AUT-Pi w.r.t. all the reductions considered

(and the permutative reductions) except ε.

5.4. Church-Rosser property and Church-Rosser theorem

5.4.1. We define (relative to a reduction relation):

(i) (*Church-Rosser property*): $CR(\Sigma)$ if $\Delta \leq \Sigma \geq \Gamma \Rightarrow \Delta \downarrow \Gamma$

(ii) (*Weak Church-Rosser property*): $CR_1(\Sigma)$ if $\Delta <_1 \Sigma >_1 \Gamma \Rightarrow \Delta \downarrow \Gamma$

(iii) *Church-Rosser theorem* (C-R-thm) for A: if $\Sigma \in A, \Gamma \in A$ then

$$\Sigma = \Gamma \Rightarrow \Sigma \downarrow \Gamma$$

(iv) *Weak Church-Rosser theorem* for A: if $\Sigma \in A$, $\Gamma \in A$ then

$$\Sigma \sim_A \Gamma \Rightarrow \Sigma \downarrow \Gamma$$

(v) A is CR (resp. CR_1) if $\Sigma \in A \Rightarrow CR(\Sigma)$ (resp. $CR_1(\Sigma)$).

5.4.2. Clearly, $CR \Rightarrow CR_1$ (for the converse implication see 6.1.5.), and (C-R-thm for A) $\Rightarrow$ (weak C-R-thm for A). And, if A is closed then (A satisfies the weak C-R-thm) $\Leftrightarrow$ (A is CR).

Since $=$ is $\downarrow^*$ and $\sim_A$ is $(\downarrow_A)^*$ (sec. 4.7), the C-R-thm (resp. the weak C-R-thm) asserts the transitivity of $\downarrow$ (resp. $\downarrow_A$).

If A satisfies the C-R-thm, $\Sigma \in A$, Σ has normal form $\Gamma \in A$ then $\Sigma \geq \Gamma$, so $N(\Sigma)$. Hence, if $\Sigma \in A$, Σ has normal forms $\Gamma \in A$ and $\Delta \in A$ then $\Gamma \equiv \Delta$. Conversely, if A is N and, for normal $\Sigma, \Gamma \in A$ we have $\Sigma = \Gamma \Rightarrow \Sigma \equiv \Gamma$, then A satisfies the C-R-thm.

5.4.3. Anyhow, if $CR(\Sigma)$, $\Sigma \geq \Gamma$, $\Sigma \geq \Delta$, both Γ and Δ are normal then $\Gamma \equiv \Delta$ (*uniqueness of normal forms*). Hence, if $CR(\Sigma)$ and $N(\Sigma)$ then we can define *the normal form*, $nf(\Sigma)$, of Σ. Conversely, if A is closed and N and all $\Sigma \in A$ have just one normal form then A is CR.

5.4.4. If A is N and CR then, for all $\Sigma \in A$, $nf(\Sigma)$ can be effectively computed, so the relation $\downarrow_A$ is decidable. So, if A is N and A satisfies the C-R-thm (resp. the weak C-R-thm) then the definitional equality $=$ (resp. $\sim_A$) is decidable on A.

5.4.5. Finally, let A and A' be sets, $A \subset A'$. If A' is CR (resp. CR_1, etc.) then A is so too (compare 5.2.2 and 5.3.4).

II 6. CR continued

6.1. How to prove CR

6.1.1. Here follow some elementary considerations on two possible methods of proving CR, viz. with and without making use of SN. The first method, i.e. with use of SN, reduces the CR-problem to CR_1. The point of this is that CR_1 is usually easily verified. A case analysis of CR_1 w.r.t. our list of elementary reductions follows in sec. II.8. The second method, without use of SN, employs our "nested" one-step reductions.

For more complete comment on CR-proofs, we refer to, e.g. [2].

6.1.2. For good comparison of the methods we introduce a slightly more general situation. Let $\to$ be some binary relation (think of a reduction relation). Let $\overset{*}{\to}$ (resp. $\overset{\circ}{\to}$) be the transitive and reflexive (resp. the reflexive) closure of $\to$. Let $B \leftarrow A$ stand for $A \to B$ etc. Let Σ be an expression. We define, for $\to$ and Σ: (with quantification conventions as in sec. II.0.4.3)

(i) *diamond property* : $\Gamma \leftarrow \Sigma \to \Delta \Rightarrow \Gamma \overset{\circ}{\to} \Sigma' \overset{\circ}{\leftarrow} \Delta$

(ii) *plank property* : $\Gamma \leftarrow \Sigma \overset{*}{\to} \Delta \Rightarrow \Gamma \overset{*}{\to} \Sigma' \overset{\circ}{\leftarrow} \Delta$

(iii) *weak plank property* : $\Gamma \leftarrow \Sigma \overset{*}{\to} \Delta \Rightarrow \Gamma \overset{*}{\to} \Sigma' \overset{*}{\leftarrow} \Delta$

(iv) *weak diamond property*: $\Gamma \leftarrow \Sigma \to \Delta \Rightarrow \Gamma \overset{*}{\to} \Sigma' \overset{*}{\leftarrow} \Delta$

where the terminology refers to the geometry of the illustrating diagrams intended.

We say that the property holds in A, if all $\Sigma \in A$ satisfy that property - but it is not required that the Γ, Δ and Σ' mentioned are themselves in A too - .

6.1.3. Let us abbreviate the diamond property for $\overset{*}{\to}$ by $(i)^*$. Then it is clear from the definition that (i) $\Rightarrow$ (iv), that (ii) $\Rightarrow$ (iii) $\Rightarrow$ (iv) and that $(i)^* \Rightarrow$ (iii). Further, if A is closed under $\to$, then by induction on (the definition of) $\overset{*}{\to}$: ((i) holds in A $\Rightarrow$ (ii) holds in A), and: ((iii) holds in A $\Rightarrow$ $(i)^*$ holds in A). So in a closed (under $\to$) set A:

(diamond property for $\to$) $\Rightarrow$ (diamond property for $\overset{*}{\to}$) .

6.1.4. But if A is closed under $\to$, and additionally $\overset{*}{\to}$ is well-founded, then we can say more: (iv) holds in A $\Rightarrow$ (i)* holds in A. Proof: assume that (iv) holds in A. By induction on the well-founded relation $\overset{*}{\to}$ we prove that the diamond property for $\overset{*}{\to}$, i.e. (i)*, holds in A. So, let $\Sigma \in A$, $\Gamma \overset{*}{\leftarrow} \Sigma \overset{*}{\to} \Delta$. We want a Σ' with $\Gamma \overset{*}{\to} \Sigma' \overset{*}{\leftarrow} \Delta$. If $\Sigma \equiv \Gamma$ (or $\Sigma \equiv \Delta$) then simply take $\Sigma' \equiv \Delta$ (resp. $\Sigma' \equiv \Gamma$). Otherwise (it is advised to draw a diagram), for some $\Gamma_1 \not\equiv \Sigma$, $\Delta_1 \not\equiv \Sigma$, $\Gamma \overset{*}{\leftarrow} \Gamma_1 \leftarrow \Sigma \to \Delta_1 \overset{*}{\to} \Delta$. By (iv) for some Σ_1', $\Gamma_1 \overset{*}{\to} \Sigma_1' \overset{*}{\leftarrow} \Delta_1$. By the induction hypothesis applied to Γ_1 and Δ_1 we find Γ_1', Δ_1' with $\Gamma \overset{*}{\to} \Gamma_1' \overset{*}{\leftarrow} \Sigma_1' \overset{*}{\to} \Delta_1' \overset{*}{\leftarrow} \Delta$. Finally, by the induction hypothesis applied to Σ_1' we find the desired Σ' with $\Gamma \overset{*}{\to} \Gamma_1' \overset{*}{\to} \Sigma' \overset{*}{\leftarrow} \Delta_1' \overset{*}{\leftarrow} \Delta$, q.e.d. So, in this case:
(weak diamond property for $\to$) $\Rightarrow$ (diamond property for $\overset{*}{\to}$).

6.1.5. Now we come back to the original situation: if $\to$ is one-step reduction then the diamond property for $\overset{*}{\to}$ is just property CR. And if we take ordinary one-step reduction for $\to$ then the weak diamond property is precisely CR_1.

So 6.1.4. provides the first method of proving CR: If A is closed, SN and CR_1 then A is CR.

And 6.1.3. provides the second method, as follows: call a compound one-step reduction $>$ *suitable* if (1) $\overset{*}{>} \Leftrightarrow \geq$ (i.e. $>_1 \Rightarrow > \Rightarrow >_1^*$) and (2) $>$ satisfies the diamond property. Once such a one-step reduction has been indicated, one can apply 6.3 and prove CR. Indeed, the common CR-proofs (for free λ-calculus, where SN does not hold) work in this way - i.e. they can be rephrased along these lines - .

6.2. A survey of results

6.2.1. The analysis in sec. II.8 of CR_1 yields at least - i.e. as long as we do not use SN - some negative results concerning CR. These negative results are of two kinds: first there are the problems with the type-labels which were first mentioned by Nederpelt [51,p.71] in connection with $\beta\eta$-reductions. As a result $\beta\eta$-CR simply does not hold in the full universe of expressions but only for the correct Automath expressions (chapter V, chapter VI). Analogous problems arise from $\pi\delta$-reductions and $+\varepsilon$-reductions (chapter VIII).

The second kind of negative result is more serious: it appears that for any reduction relation including $\beta\varepsilon$-reductions, CR is false,

even if the type labels are ignored. More about this in chapter VIII too (VIII.6).

6.2.2. Now we mention some facts which show the relevance of our compound reductions $\bar{>}_1$ and $\tilde{>}_1$. First, $\bar{>}_{1,\delta}$ (i.e. disjoint one-step δ-reduction) is suitable (in the sense of 6.1.5) for δ-reduction (chapter III, sec. 3.3). Secondly, by the way, the disjoint one-step reduction generated by weak reductions is suitable for weak combinatory logic (Rosser, in Troelstra [69]). Further, $\tilde{>}_{1,\beta}$ is suitable for β-reduction in free λ-calculus (Tait, Martin-Löf, in Barendregt [2]) and in the generalized typed λ-calculus (Nederpelt [51]). In fact, $\tilde{>}_1$ is suitable for the combination of all the elementary reductions, except σ and ε, provided we leave out the type-labels. This was proved for $\beta\eta\pi$-reduction by Mann [43] ; he also indicated the problem with σ as explained in sec II 8.4. Below we prove the suitability of $\tilde{>}_{1,\beta\eta}$ for free λ-calculus, simplifying the proof of Mann.

6.3. A proof of $\beta\eta$-CR in free λ-calculus

6.3.1. This proof via the suitability of $\tilde{>}_{1,\beta\eta}$ for free λ-calculus (which fact was claimed by Barendregt [2]) is just slightly more involved than in the β-case, in contrast with Mann's proof which is unnecessarily complicated. As explained in sec. 6.1.5, the suitability is sufficient to prove CR.

6.3.2. The expressions are: variables x, λ-expressions $\lambda x{\cdot}A$, application expressions BA. By writing A', B' we implicitly intend that A > A', B > B', etc. The elementary reductions are, as in sec. II.3.4: (β) $(\lambda x{\cdot}B)A > B'[\![A']\!]$, ($\eta$) $x \notin \mathrm{FV}(A) \Rightarrow \lambda x{\cdot}Ax > A'$. From sec. 3.8 we recall the substitution property V: $B[\![A]\!] > B'[\![A']\!]$.

6.3.3. If $\lambda x{\cdot}A > B$ then either (1) $B \equiv \lambda x{\cdot}A'$, or (2) $A \equiv Cx$, $x \notin \mathrm{FV}(C)$, $C > B$. So, if $\lambda x{\cdot}Cx > B$ then either (1a) $B \equiv \lambda x{\cdot}C'x$, or (1b) $C \equiv \lambda y{\cdot}D$, $B \equiv \lambda x{\cdot}D'[\![y/x]\!]$, or (2) $x \notin \mathrm{FV}(C)$, $C > B$.

6.3.4. If $BA > C$ then either (1) $C \equiv B'A'$, or (2) $B \equiv \lambda x{\cdot}D$, $C \equiv D'[\![A']\!]$. So, if $(\lambda x{\cdot}D)A > C$ then either (1a) $C \equiv (\lambda x{\cdot}D')A'$, or (1b) $D \equiv Ex$,

$C \equiv E'A'$, or (2) $C \equiv D'[\![A']\!]$.

6.3.5. Now we just have to prove the diamond property: $A_1 < A > A_2 \Rightarrow A_1 > A_3 < A_2$. We use induction on A. If $A \equiv x$ then there is nothing to prove. If A is a λ-expression or an application expression then we must confront the various possibilities ((1), resp. (1a) and (1b), and (2) of 6.3.3, resp. 6.3.4) of reducing A to A_1 and A_2 with each other. In both cases (A is λ-expression or not) the combination (1)v.(1) (i.e. $A > A_1$, $A > A_2$ both by "internal" reduction), (2)v.(2) (i.e. $A > A_1$, $A > A_2$ both by an "outside" reduction) and (1a) v.(2) are just standard.

6.3.6. So, let ((1b)v.(2)) $A \equiv \lambda x \cdot (\lambda y \cdot D)x$, $x \notin \mathrm{FV}(D)$, $A_1 \equiv \lambda x \cdot D'[\![y/x]\!]$, $A_2 \equiv E$, $\lambda y \cdot D > E$. Applying the ind. hyp. to $\lambda y \cdot D$ we find A_3 with $\lambda y \cdot D' > A_3 < E$. Since $x \notin \mathrm{FV}(D')$, $\lambda y \cdot D' \equiv \lambda x \cdot D'[\![y/x]\!]$, so A_3 does the work.

6.3.7. And, let ((1b)v.(2)) $A \equiv (\lambda x \cdot Ex)D$, $x \notin \mathrm{FV}(E)$, $A_1 \equiv E'D'$, $A_2 \equiv F[\![D"]\!]$, $Ex > F$, $D > D"$. Applying the ind. hyp. to Ex and to D we find H and D''' with $F > H < E'x$, $D' > D''' < D"$. By the substitution property $F[\![D"]\!] > H[\![D'''\!]] < (E'x)[\![D']\!] \equiv E'D'$ (because $x \notin \mathrm{FV}(E')$). So this $H[\![D'''\!]]$ can serve as A_3, q.e.d.

II.7. Combined reductions

7.1.1. In some cases desirable properties, such as N, SN and CR, for a combination of reduction relations $\geq$ and $\geq'$ can appropriately be proved by first considering $\geq$ and $\geq'$ separately and then use certain connections between $\geq$, $\geq'$ and their "union". An example of this can be found in sec. 5.3.4 (second half).

Interesting questions on the connections of $\geq$, $\geq'$ and their "union" are whether $\geq$ and $\geq'$ *commute* (cf. sec. 7.2 below) and whether $\geq'$-*postponement* holds (cf. sec. 7.3 below).

7.1.2. Let i and j stand for (combinations of) elementary reductions, and let ij refer to their "union". E.g. if i denotes $\beta\eta$ and j denotes δ then ij stand for $\beta\eta\delta$. We write $>_i$, $>_{1,i}$, $\geq_i$ etc. for the corresponding (one-step) reductions. We use $\leq_i$ etc. in the usual sense.

We say that $\Sigma >_i >_j \Gamma$, resp. $\Sigma >_i \geq_j \Gamma$, if for some Δ, $\Sigma >_i \Delta >_j \Gamma$ resp. $\Sigma >_i \Delta \geq_j \Gamma$. Similarly $\geq_i >_j$ and $\geq_i \geq_j$.

The notation $\Sigma \leq_i <_j \Gamma$ is used for $\Gamma >_j \geq_i \Sigma$ etc.

7.2.1. Church-Rosser for combined reductions

In Staples [65] *) we find some ingenious constructions for proving that a combined system is CR. Here we restrict ourselves to some simple properties.

We assume that A, a set of expressions, is ij-closed (i.e. closed under $\geq_{ij}$), and that all expressions considered are elements of A.

Clearly, $(\geq_i \geq_j)^*$ is just $\geq_{ij}$, so if $\geq_i \geq_j$ satisfies the diamond property then we have ij-CR - because $\geq_i \geq_j$ is a suitable one-step reduction for $\geq_{ij}$, in the sense of sec. 6.1.5.

We say that $\geq_i$ and $\geq_j$ *commute* if, for all Σ (quantification as in 0.4.2),

$$\Gamma \leq_i \Sigma \geq_j \Delta \Rightarrow \Gamma \geq_j \Sigma' \leq_i \Delta$$

Thus, if i-CR, j-CR and $\geq_i$ and $\geq_j$ commute then ij-CR.

7.2.2. When do $\geq_i$ and $\geq_j$ commute?

We give an analysis analogous to sec. II.6.1. Define, for one-step reductions $>_i$ and $>_j$,

(i) *diamond property* : $\Gamma <_j \Sigma >_i \Delta \Rightarrow \Gamma >_i \Sigma' <_j \Delta$

(ii) *trapezium property* : $\Gamma <_j \Sigma >_i \Delta \Rightarrow \Gamma \geq_i \Sigma' <_j \Delta$

(iii) *plank property* : $\Gamma <_j \Sigma \geq_i \Delta \Rightarrow \Gamma \geq_i \Sigma' <_j \Delta$

(iv) *weak plank property* : $\Gamma <_j \Sigma \geq_i \Delta \Rightarrow \Gamma \geq_i \Sigma' \leq_j \Delta$

(v) *weak diamond property*: $\Gamma <_j \Sigma >_i \Delta \Rightarrow \Gamma \geq_i \Sigma' \leq_j \Delta$

As in sec. 6.1.3 , (i) ⇒ (ii) ⇔ (iii) ⇒ (iv) ⇔ ($\geq_i$ and $\geq_j$ commute) ⇒ (v).

And if ij-SN, $>_i$ and $>_j$ satisfy (v) then also $\geq_i$ and $\geq_j$ commute (as in sec. 6.1.4).

7.2.3. So, just as in the case of ordinary CR, there are two possible ways of proving that $\geq_i$ and $\geq_j$ commute (viz. with and without SN). With SN, it is sufficient to prove the weak diamond property for $>_{1,i}$ and $>_{1,j}$. But without SN, we rather look for a compound reduction $>_i$

*) See also de Bruijn [19]

such that $>_i$ and, say, $>_{1,j}$ satisfy at least the trapezium property.

7.2.4. The analysis of sec. 8 provides us with the weak diamond property for all combination of η-, π-, β-, δ- and +-reduction (but for the type-labels, of course). Let $>^{\circ}_{1,i}$ stand for the reflexive closure of $>_{1,i}$ (i.e. contract one or zero i-redices at a time). Then sec. 8.8 also shows that all combination of $>^{\circ}_{1,\eta}, >^{\circ}_{1,\pi}$ and $>^{\circ}_{1,+}$ satisfy the diamond property, and that all combinations of $>^{\circ}_{1,\beta}$ (resp. $>^{\circ}_{1,\delta}$) with $>_{1,\eta}$, $>_{1,\pi}$ and $>_{1,+}$ satisfy the trapezium property (modulo the type labels). In the $\beta\eta$-case this gives an easy alternative proof of $\beta\eta$-CR (compare sec. 6.3) for the free λ-calculus, viz. from β-CR (e.g. by the Tait-Martin-Löf method) and η-CR (which is trivial from e.g. η-SN).

A simple variant of the CR-proof in sec. 6.3 (or rather of the CR-proof in chapter III, sec. 3.3) shows that $\bar{>}_{1,\beta}$ and $>_{1,\delta}$ satisfy the trapezium property: $\Gamma \bar{<}_{1,\beta} \Sigma >_{1,\delta} \Delta \Rightarrow \Gamma \geq_{\delta} \Sigma' <_{1,\beta} \Delta$. Alternatively, one can prove that $\tilde{>}_{1,\beta}$ and $\tilde{>}_{1,\delta}$ together satisfy the diamond property.

Resuming, η-, π-, β-, δ- and +-reductions commute with each other (but for the type-labels).

7.2.5. Further, sec. 8.8 yields some negative results about the commuting of reductions, even if we ignore the type-labels. First there is the $\beta\varepsilon$-problem.

Secondly, there are the problems with σ and ε: neither σ nor ε commutes with any other reduction.

7.3.1. Postponement

For some cases of i,j no "new" i-redices are created by j-reductions, and the i-contractions in an ij-reduction can be carried out first. This property is called ij-*postponement*, for short ij-PP. We say that Σ satisfies ij-PP if

$$\Sigma \geq_{ij} \Gamma \Rightarrow \Sigma \geq_i \geq_j \Gamma$$

and we say that ij-PP holds in a set A if all $\Sigma \in A$ satisfy ij-PP.

Clearly we have ii-PP. Use the index $i^{\leftarrow}$ for the "converse" i-reduction:

$$\Sigma \geq_i^{\leftarrow} \Gamma :\Leftrightarrow \Sigma \leq_i \Gamma. \text{ Similarly } >_i^{\leftarrow} \text{ etc.}$$

Then ij-PP is strongly connected with $ij^{\leftarrow}$-CR. In fact, in a closed (under $\geq_{ij}$) set A, ij-PP is equivalent with the property

$$\Sigma \geq_j \geq_i \Gamma \Rightarrow \Sigma \geq_i \geq_j \Gamma$$

i.e. $\geq_j^{\leftarrow}$ and $\geq_i$ commute in the sense of 7.2.1.

7.3.2. When does postponement hold?

Let us confine the discussion to a closed (under $\geq_{ij}$) set A. Since the question of ij-PP just amounts to the question whether $\geq_i$ and $\geq_j^{\leftarrow}$ commute, we can simply follow the development in sec. 7.2. Define, for one-step reductions $>_i$ and $>_j$,

(i) trapezium property I : $\Sigma >_j >_i \Gamma \Rightarrow \Sigma \geq_i >_j \Gamma$

(ii) trapezium property II: $\Sigma >_j >_i \Gamma \Rightarrow \Sigma >_i \geq_j \Gamma$

Since both trapezium properties imply ij-PP, it is sufficient for ij-PP to indicate a suitable one-step reduction $>_j$ (resp. $>_i$) satisfying trapezium property I (resp. II).

But, if we have i-SN, we can do with a weaker form of (ii),

(iii) $\Sigma >_j >_{1,i} \Gamma \Rightarrow \Sigma >_{1,i} \geq_{ij} \Gamma$

For, using induction on $\geq_j$, we find

(iv) $\Sigma \geq_j >_{1,i} \Gamma \Rightarrow \Sigma >_{1,i} \geq_{ij} \Gamma$

So, assuming i-SN, we can use induction on the well-founded relation $\geq_i$ and prove ij-PP, as follows: let $\Sigma \geq_j \geq_i \Gamma$. If $\Sigma \geq_j \Gamma$ there is nothing to prove. Otherwise, $\Sigma \geq_j >_{1,i} \Gamma_1 \geq_i \Gamma$, for some Γ_1. By (iv), $\Sigma >_{1,i} \Sigma_1 \geq_{ij} \Gamma_1$. By the induction hypothesis applied to Σ_1 we find that $\Sigma_1 \geq_i \geq_j \Gamma$ q.e.d.

7.3.3. Some results

The fact that $\beta\eta$-PP holds belongs to the tradition of the free λ-calculus. Nederpelt's proof (in [51]) shows the trapezium property I for the combination of $>_{1,\beta}$ and $\tilde{>}_{1,\eta}$. As Nederpelt points out, Curry's proof in [25] which instead aims at the trapezium property II for a compound one-step β-reduction (with $>_{1,\eta}$) is defective (though $\tilde{>}_{1,\beta}$

would have worked).

From sec. 9.2.4 it is clear that the following combinations of reductions satisfy property (iii) of sec. 7.3.2: of $>_{1,\beta}$ with $>_{1,\eta}$, of $>_{1,\pi}$ with $>_{1,\sigma}$, and of $>_{1,\beta+}$ with $>_{1,\eta}$. Assuming some weak type restrictions (clearly satisfied by correct Automath expressions) we also get property (iii) for $>_{1,\beta}$ with $>_{1,\sigma}$, and for $>_{1,\pi}$ with $>_{1,\eta}$. This, together with the appropriate SN-assumptions, yields that the combination of the I.E. reductions (sec. 3.3) $\beta\pi+$ with the ext-reductions σ and η allows postponement of these ext-reductions.

Alternatively, we can extend Nederpelt's construction to these cases, with the nested version $\tilde{>}_{1,\eta\sigma}$ (and get PP without resorting to SN).

Anyhow, ε-reduction is an exception: its postponement is not possible, viz. in combination with $\beta+$-reduction.

7.3.4. As an application of PP in general, we give the following theorem:
if i-SN, j-SN and ij-PP then ij-SN
Proof: Let Σ be an expression. By induction on the i-reduction tree of Σ we show that all ij-reduction sequences of Σ are secured. Let $\Sigma >_1 \Gamma_1 >_1 \Gamma_2 >_1 \ldots$. By PP, for all k, $\Sigma \geq_i \geq_j \Gamma_k$. The j-reduction tree of Σ is finite, so, if for all k, $\Sigma \geq_j \Gamma_k$, the reduction sequence is finite (whence secured). Otherwise, for some proper i-reduct Σ' and some Γ in the reduction sequence, $\Sigma' \geq_{ij} \Gamma$. By induction hypothesis, Σ' is SN, so Γ is SN so the reduction sequence is secured q.e.d.

7.3.5. In fact it is more straightforward to prove the theorem from property (iii), section 7.3.2. (which holds in all our PP-cases), directly:
If i-SN, j-SN and property (iii) holds (i.e. $\geq_j \geq_{1,i} \Rightarrow >_{1,i} \geq_{ij}$) then ij-SN.
Proof: Let Σ be an expression, let $\Sigma >_1 \Gamma_1 >_1 \Gamma_2 >_1 \ldots$. Again, we use induction on the i-reduction tree. If the reduction sequence just contains j-reductions, then it is finite, by j-SN. Otherwise, for some k, $\Sigma \geq_j \Gamma_k >_{1,i} \Gamma_{k+1}$. By (iii), for some Σ', $\Sigma >_{1,i} \Sigma' \geq_{ij} \Gamma_{k+1}$.
By the ind. hyp. Σ' (so Γ_{k+1}) is ij-SN, and the reduction sequence is secured q.e.d.

As a corollary of this, we have β-SN $\Rightarrow$ $\beta\eta$-SN.

7.4.1. Weak postponement

For some cases of i,j, indeed no essentially new i-redices are created by the j-reductions, but if one starts with carrying out the i-contractions, possibly too many i-redices are contracted. We say that *weak ij-postponement* (weak ij-PP) holds, if for all Σ,

$$\Sigma \geq_{ij} \Gamma \Rightarrow \Sigma \geq_i \geq_j \Gamma' \leq_i \Gamma$$

In particular, as sec. 9.3.1 shows, we have only weak $\delta\beta$-PP.

There are two relevant ways of proving weak ij-PP, viz. with and without use of i-SN. First, without i-SN. We introduce some properties:

(i) $\Sigma >_j >_i \Gamma \Rightarrow \Sigma >_i \geq_j \Gamma' \leq_i \Gamma$ (a kind of weak trapezium property I)

(ii) $\Sigma \geq_j >_i \Gamma \Rightarrow \Sigma >_i \geq_j \Gamma' \leq_i \Gamma$ (a kind of plank property)

(iii) $\Sigma \geq_j \geq_i \Gamma \Rightarrow \Sigma \geq_i \geq_j \Gamma' \leq_i \Gamma$

Assume that i- and j-reduction commute. Clearly, (iii) implies weak ij-PP, and (by induction on $\geq_j$) (i) implies (ii). Further, if $>_i$ satisfies the plank property for CR ($<_i \geq_i \Rightarrow \geq_i <_i$), then (ii) implies (iii). So: if

(1) $>_i$ satisfies the plank property for CR,

(2) i and j commute,

(3) property (i) holds, then we have weak ij-PP

(hence without using SN).

Then, with SN. We introduce a weak form of property (iii) sec. 7.3.2.

(iv) $\Sigma >_j >_{1,i} \Gamma \Rightarrow \Sigma >_{1,i} \geq_{ij} \Gamma' \leq_i \Gamma$

Assume that i-reduction and ij-reduction commute. Then (iv) gives, by induction on $\geq_j$,

(v) $\Sigma \geq_j >_{1,i} \Gamma \Rightarrow \Sigma >_{1,i} \geq_{ij} \Gamma' \leq_i \Gamma$

By induction on i-reduction trees, we get:
if (1) i-SN, (2) i-CR, (3) i and j commute (so, with i-CR, i and ij commute), (4) property (iv) holds then weak ij-PP.

7.4.2. As a corollary of (1) i-CR, (2) i and j commute, (3) weak ij-PP, (4) i-N (i.e. i-normalization) we get: $\Sigma \geq_j \Gamma \Rightarrow$ i-nf(Σ) $\geq_j$ i-nf(Γ).
An alternative way of getting the latter property (which, in turn, implies weak ij-PP) avoiding the question whether i and j commute, is

from: (1) i-CR, (2) i-N , (3) weak ij-PP and (4): for all Σ,

$$\Gamma \leq_j \Sigma \geq_i \Delta \Rightarrow \Gamma \geq_i \Sigma' \leq_{ij} \Sigma.$$

7.4.3. Section 9.3.4 learns us that δj-PP holds for all reductions j except β-reduction. This can be proved either from δ-SN (Chapter III) and property (iii) section 7.3.2 or without δ-SN, by showing trapezium property II (sec. 7.3.2) for $\bar{>}_{1,\delta}$ and $>_{1,j}$.

Further δ commutes with all reductions but σ and ε. For the latter two reductions, however, we can prove (with σ and ε in the role of j):

$$\Gamma \leq_j \Sigma \geq_\delta \Delta \Rightarrow \Gamma \geq_\delta \Sigma' \leq_{\delta j} \Delta$$

So, assuming δ-CR and δ-N, for all reductions j but β we have already:

$$\Sigma \geq_j \Gamma \Rightarrow \delta\text{-nf}(\Sigma) \geq_j \delta\text{-nf}(\Gamma)$$

Finally, we have weak $\delta\beta$-PP, with use of δ-SN and property (iv) above, or alternatively from property (i) above (quite simple, with $>_{1,\delta}$ and $>_{1,\beta}$). So, in this case too, if δ-N and δ-CR then

$$\Sigma \geq_\beta \Gamma \Rightarrow \delta\text{-nf}(\Sigma) \geq_\beta \delta\text{-nf}(\Gamma) \ .$$

7.4.4. For the rest, weak postponement is just what we get in the following situation: let D_1 and D_2 be disjoint sets of definitional constants, let $\geq_{\delta D_1}$, resp. $\geq_{\delta D_2}$ denote the reduction relation generated by contracting constants from D_1 (resp. D_2) exclusively. If the defining axioms of the constants in D_1 do not contain constants in D_2 then we have weak $\delta_{D_2}\delta_{D_1}$-PP.

II.8. An informal analysis of CR_1

8.1. In presence of SN, the weak CR-property CR_1 is sufficient for CR (see sec. 6.1.5). Anyhow, for the heuristics of a CR-proof an analysis of CR_1 is indispensable. Let i and j indicate kinds of elementary reduction, such as β, η etc. Let Σ be an expression, with an i-redex $R \subset \Sigma$ and a j-redex $S \subset \Sigma$. By contracting R to R' (resp. S to S') we get $\Sigma >_{1,i} \Gamma$ (resp. $\Sigma >_{1,j} \Delta$). We want to find out whether Γ and Δ have a common reduct Σ' and if so, by what kind of and by how many contractions, Σ' can be reached from Γ and Δ. In the informal discussion

below all possible cases are systematically treated, according to the relative positions of the redices R and S.

8.2. The first point is of course, that either (a) R and S are disjoint, (b) $R \equiv S$, (c) R sub S or (d) S sub R . In case (a), the contractions just commute:

$$\Sigma \equiv \ldots R \ldots S \ldots >_{1,i} \Gamma \equiv \ldots R' \ldots S \ldots >_{1,j}$$
$$\Sigma' \equiv \ldots R' \ldots S' \ldots <_{1,i} \Delta \equiv \ldots R \ldots S' \ldots <_{1,j} \Sigma \quad .$$

As for case (b), if we assume that

(*) for each definitional constant only one defining axiom is given,

then all elementary reductions are *mutually exclusive*. I.e. if R i-contracts to R' and R j-contracts to S' then i and j refer to the same kind of reduction and $R' \equiv S'$. So, under assumption (*), which is indeed fulfilled in the Automath system of abbreviations, in case (b) for a common reduct we can take $\Sigma' \blacksquare \Gamma (\equiv \Delta)$.

Case (c) is discussed in sec. 8.4 and further. Case (d) can of course be reduced to case (c) by interchanging i and j, R and S.

8.3. About expression variables in schemes for reduction

The elementary reductions are formulated in *schematic form*, i.e. with meta-variables for expressions in them. For instance, in the scheme of β-reduction "$\{A\}[x{:}B]C$ elementary reduces to C[[A]]" (in sec.3.2.1), the meta-variables A, B, C are the expression variables of the scheme.

For each of the schemes, all of its expression variables occur (of course!) at least once in the left-hand side (redex). Let X be an expression variable of a scheme for reductions. We distinguish three cases:

(i) X *disappears* in the contractum (such as B above)
(ii) X *occurs* just *once* in the contractum, possibly there is substituted in X (such as C above).
(iii) X is possibly *multiplied* by substitution (such as A above).

For all kinds of reductions, except σ and ε, the expression variables occur precisely once in the redex. To these two exceptional cases we refer as the *twin reductions* (because of the *twin occurrences*

of the meta-variable, e.g. of X in $\langle X_{(1)}, X_{(2)} \rangle$).

8.4. Case (c). Let R sub S, S j-contracts to S'. Distinguish the following cases:

(c1) $R \subset X$ for some instance X of a meta-variable of the j-redex

(c2) not (c1), so R forms an *essential part* of S (such as $[x{:}B]C$ in $\{A\}[x{:}B]C$).

Now, unless j refers to a twin reduction and $R \subset X$ for some instance X of a twin occurrence, in case (c1) the j-redex is not spoilt by the i-contraction. For common reduct Σ' we take the result of simply contracting the modified (by the internal i-contraction) j-redex in Γ. From Δ we can reach Σ' by i-contracting nothing (if X disappears, i.e. case (i), sec. 8.3), i-contracting one possibly modified (by substitution) occurrence of R (if X occurs once, i.e. case (ii), sec. 8.3) or i-contracting possibly more disjoint occurrences of R (if X multiplies, case (iii), sec. 8.3). So $\Sigma >_{1,i} \Gamma >_{1,j} \Sigma' \bar{<}_{1,i} \Delta <_{1,j} \Sigma$ (where $\bar{>}_{1,i}$ is disjoint one-step i-reduction).

Examples:

(1) j is β, X "occurs once", use substitution property I, sec. 3.8:

$$\Sigma \equiv S \equiv \{A\}[x{:}B]R >_{1,i} \Gamma \equiv \{A[\![x{:}B]\!]R' >_{1,\beta}$$

$$\Sigma' \equiv R'[\![A]\!] <_{1,i} \Delta \equiv S' \equiv R[\![A]\!] <_{1,\beta} \Sigma$$

(2) j is β, X "multiplies",

$$\Sigma \equiv S \equiv \{R\}[x{:}B]\ldots x\ldots x\ldots >_{1,i} \Gamma \equiv \{R'\}[x{:}B]\ldots x\ldots x\ldots >_{1,\beta}$$

$$\Sigma' \equiv \ldots R'\ldots R'\ldots \bar{<}_{1,i} \Delta \equiv S' \equiv \ldots R\ldots R\ldots <_{1,\beta} \Sigma$$

In contrast with this, if j refers to a twin case and $R \subset X$ for some "twin variable" X, then the j-redex is spoilt by the i-contraction indeed - but can be restored by i-contracting the other twin as well. So, since twin variables occur just once in the contractum (case (ii), sec. 8.3), for some Γ', Σ', $\Sigma >_{1,i} \Gamma >_{1,i} \Gamma' >_{1,j} \Sigma' <_{1,i} \Delta <_{1,j} \Sigma$. Hence, in this case i and j *do not commute*. An example (where j refers to σ-reduction): $\Sigma \equiv S \equiv \langle R_{(1)}, R_{(2)} \rangle >_{1,i} \Gamma \equiv \langle R_{(1)}, R'_{(2)} \rangle >_{1,i} \Gamma' \equiv \langle R'_{(1)}, R'_{(2)} \rangle >_{1,\sigma} \Sigma' \equiv R' <_{1,i} R \equiv \Delta <_{1,\sigma} \Sigma$.

8.5. Case (c2): R is an essential part of S. Notice that there are two possibilities:

(1) j is an I.E.-reduction, i is the *corresponding* ext-reduction.
(2) i is an I.E.-reduction, j is an ext-reduction.

Case (c21). Here are three cases, η v. β, σ v. π and ε v. +. In the first two cases there is no problem, even if type-labels are present: $\Gamma \equiv \Delta$, so we can take $\Sigma' \equiv \Gamma$ too.

($\eta\beta$) $\quad \{A\}C <_{1,\eta} \{A\}[x{:}B]\{x\}C >_{1,\beta} \{A\}C \ , \qquad (x \notin FV(C))$

($\sigma\pi$) $\quad <Q,A,B>_{(p)} <_{1,\sigma} <P,<Q,A,B>_{(1)},<Q,A,B>_{(2)}>_{(p)} >_{1,\pi} <Q,A,B>_{(p)}$

$(p = 1 \text{ or } p = 2).$

The case of ε v. + is more complicated. First, there is an additional β-reduction needed. Secondly, there are problems with the type-labels.

(ε+) $\quad R \equiv ([x{:}B_1]\{i_1(x,D_1)\}C) \oplus ([x{:}B_2]\{i_2(x,D_2)\}C)$, $R' \equiv C$,

$S \equiv \{i_p(A,D_3)\}R$, $S' \equiv \{A\}[x{:}B_p]\{i_p(x,D_p)\}C$,

$(p = 1 \text{ or } p = 2, x \notin FV(C))$,

$\{i_p(A,D_3)\}C <_{1,\varepsilon} S >_{1,+} S' >_{1,\beta} \{i_p(A,D_p[\![A]\!])\}C$

So, in this case, $\Gamma <_{1,\varepsilon} \Sigma >_{1,+} \Delta >_{1,\beta} \Delta'$ with $\Gamma \equiv \Delta'$ *but for the type labels*. Hence, without type-labels, $\Sigma' \equiv \Gamma \equiv \Delta'$ can serve as a common reduct. But with type-labels type-restrictions have to be imposed in order to guarantee that $D_p[\![A]\!]$ and D_3 are definitionally equal (and may have a common reduct).

8.6. Case (c22) covers β v. η, π v.σ, + v. ε and β v. ε. In the first two cases CR_1 holds but for the type-labels. In the third case additional η-contractions are needed (compare with 8.5, ε v. +), but in the fourth case CR_1 (so CR) simply does not hold at all.

($\beta\eta$) $\quad [x{:}A]C \ <_{1,\beta} \ [x{:}A]\{x\}[x{:}B]C >_{1,\eta} [x{:}B]C \qquad x \notin FV(B)$.

So here, $\Gamma \equiv \Delta$ but for the type-labels. Regarding π v. σ, the situation compares with the twincase in 8.4:an additional π-reduction is needed.

($\pi\sigma$) $\quad <P,A,B> <_{1,\pi} <P,A,<Q,A,B>_{(2)}>$

$<_{1,\pi} S \equiv <P,<Q,A,B>_{(1)} \ , \ <Q,A,B>_{(2)}> >_{1,\sigma} S' \equiv <Q,A,B>$.

In the third place, we mention de Vrijer's definition method of $\lambda\lambda$ in [70]. He starts with the simultaneous introduction of the correct E- and Q-formulas, and after that defines correctness of expressions in terms of E, Q and typ.

1.2. Some general points on the language theory

A priori it is not clear that the various definition methods generate the same structure (of correct expressions, with typing and equality). So one might think that the language theory has two aims, viz. (1) proving the equivalence of the various formulations, and (2) proving that the generated structures satisfy some specific desirable properties (sec. 1.3).

However these aims can hardly be separated: properties are first proved for one formulation, then the equivalence is established and finally the properties are transferred to the other formulation, via the equivalence.

A simple example of this situation: for the system given by the algorithmic definition, decidability is just a matter of termination of the algorithm, i.e. normalization (as Nederpelt points out [51]). So, by the results in Chapter IV, if a system can be proved to be equivalent to the "algorithmic one", it is decidable.

As a second illustration, we sketch roughly how the development below is organized. For the terminology see II.4.7 and for the kind of reasoning see II.5.4, where for A we take $\vdash$ now.

We work with three systems: I and II are given by an E-definition and III is the algorithmic definition. The three systems essentially just differ as regards their Q-rules. In system I, Q is defined to be the equivalence relation generated by $>_{\vdash}$ (but realize that Q and $\vdash$ are introduced simultaneously). This is the restricted "technical" version of the E-definition, which we present in section 2, and take as the starting point for the development in section 3. In system II, Q is $\sim_{\vdash}$, i.e. the transitive closure of $\downarrow_{\vdash}$. This is the liberal form of the E-definition, which we think is most suitable for practical purposes, as a reference manual, say.

In system III, the algorithmic definition, which we give in section 4, Q is defined to be just $\downarrow_{\vdash}$.

We say that a system satisfies CL if its correct expressions remain correct under reduction, and that it satisfies CR if its correct expressions are CR. Clearly, both I and III are contained in II, since II has more liberal rules for Q. Further, if I satisfies CL then I and II are equivalent, as is proved by induction on the definition of correctness in system II (see sec. 2.11.2). Also by induction on II-correctness it is proved that II and III are equivalent, if III satisfies CR. Now, in section 3 we prove that I satisfies CL, and in Chapter VI we prove (roughly) CL ⇒ βηδ-CR (for the βδ-case we know CR already). This gives CR for II, so CR for III, so it shows that all the three systems are equivalent, and satisfy CL and CR.

An approach, alternative to the one sketched above, is given in Chapter VII. There the algorithmic definition serves as a starting point and CL and CR are proved simultaneously, using induction on socalled *big trees*.

1.3. What are the desirable properties?

As desirable properties for the structures of correct expressions generated, we mention:

(i) *substitutivity*: correctness of expressions and formulas is preserved under substitution with correct expressions of the right types.

(ii) *closure* (CL) and *preservation of types* (PT): correctness of expressions and formulas is preserved under reduction.

(iii) the *Church-Rosser property* CR, and the *weak Church-Rosser theorem* (see Chapter II, sec.5.4): $A\ Q\ B \Rightarrow A \downarrow B$

(iv) (*strong*)*normalization* (S)N and *decidability*

(v) properties for Q, which show that Q behaves as an equality, such as:
- the *lefthand-equality rule* LQ: $A\ E\ B,\ A\ Q\ C \Rightarrow C\ E\ B$
  (the righthand-equality rule is included in the definition)
- *monotonicity rules*: $A\ Q\ B,\ C\ Q\ D \Rightarrow \{A\}C\ Q\ \{B\}D$, etc.

(vi) uniqueness properties
- *uniqueness of types*: $A\ E\ B,\ A\ E\ C \Rightarrow B\ Q\ C$
- *uniqueness of domains* UD: $[x{:}A]B\ Q\ [x{:}C]D \Rightarrow A\ Q\ C$ (and $B\ Q\ D$)
- *extended uniqueness of domains* EUD: $[x{:}A]B\ E\ [x{:}C]D \Rightarrow A\ Q\ C$ (and $B\ E\ D$).

Of course in the presence of type-inclusion (in AUT-QE), only restricted forms of uniqueness of types and property LQ (see sec. 1.7) are valid.

It depends on the choice of a definition method and on the language defined, which of the above properties are basic and which can be derived from these basic ones. Anyhow, SN, $\delta\eta$-CR and $\delta\beta$-CR we know already. The discussion below starts with substitutivity (sec. 2.9) and ends with $\beta\eta$-CR (Chapter VI) and decidability (section 4, as sketched in 1.2). In between, (ii) and (v) and (vi), which turn out to be connected, are considered more or less simultaneously. In fact, first PT, LQ and UD and the property of

(vii) *sound applicability* SA: $\{A\}[x{:}B]C$ correct $\Rightarrow A \; E \; B$

are proved simultaneously, by a careful induction on degree. Then follows one-step closure CL_1 by induction on correctness, and finally CL, by induction on $\geq$.

1.4. Some points on closure

Apart from the specific role which closure plays in our discussion, it is of course important as a technical property, in view of II.5-6. Compare, e.g. IV.2: the point of the generalization from the correct expressions to the normable expressions, lies precisely in the fact that the normable system is "large enough" to prove closure for it in a relatively easy fashion (in contrast with closure for the correct expressions), and small enough to prove (strong) normalization for it, with the help of closure.

The normalization properties and CR are nicely preserved under certain forms of taking subsystems (II.5.2.2 and II.5.3.4). So it is sufficient to prove these properties for some "large" systems: normalization for the normable expressions, $\beta\delta$- and $\eta\delta$-CR for all the expressions, and $\beta\eta\delta$-CR under fairly general conditions in Chapter VI.

The closure property however, in spite of II.5.2.2, poses a separate problem for each particular language, because correctness is defined *in terms of* reduction.

Further we must stick to a particular definition, since in the proof of closure we often apply induction on the definition of correctness. Only after closure has been proved, some important derived rules follow and equivalence with the alternative definitions can be estab-

lished.

Nevertheless, we try and give a uniform treatment of the various languages here, by splitting up the closure proof in the parts, common to all the languages (e.g. substitutivity, $CL_1 \Rightarrow CL$, etc.), and the part specific for each particular language, i.e. the proof of SA, UD, PT and LQ. The specific part is given quite elaborately for the "worst case", βη-AUT-QE (and its extensions), in sec. 3.2 and 3.3, and just sketched for the simpler languages, such as βδ-AUT-QE, βη-AUT-68 etc. (sec. 3.4). In fact, for the simpler languages the specific part simply vanishes, in which case the whole closure proof boils down to the simple closure proofs in Girard [31] and Martin-Löf [45].

1.5. Summary

Section 2 starts with a list of inductive clauses for establishing correctness of expressions, E- and Q-formulas, relative to correct book and context, as in the previous chapter. E-definitions for particular languages are specified by indicating (1) a reduction relation (β-reduction with or without δ and η), (2) possible degree restrictions, (3) a particular set of rules from the list. In order to avoid confusion we restrict ourselves here to the regular languages (i.e. degrees only 1, 2 and 3), from β-AUT-68 to βηδ-AUT-QE+. Then we prove some simple properties (renaming of contexts, substitutivity, correctness of categories) and give a short discussion of some of the rules.

Section 3 deals with the actual proof of closure and the connected properties (i.e. (ii), (v), (vi) and (vii) above) for the whole range of regular languages, as far as these properties are valid (in view of type-inclusion). First, heuristic considerations (sec. 3.1) point out how the connections can be, and how the proof might be organized in the more complicated cases (such as βη-AUT-QE). Secondly, the proof is actually carried out for βη-AUT-QE (sec. 3.2). After that, via an *unessential extension* result, all the properties are transferred to βηδ-AUT-QE+ (sec. 3.3). Finally, it is shown, that for all the simpler languages (βη-AUT-68, βδ-AUT-QE(+), etc.) easier proofs can be given, which use the more liberal E-definition II (see 1.2) instead of I as a starting point (sec. 3.4).

We claim that the restriction to degrees 1, 2 and 3 in the closure proof of βη-AUT-QE is not essential, and that this proof can be easily

adapted for $\Lambda(+)$, using the results on norm-degree-correctness in VII.2.2.

Section 4 contains the details of the equivalence proof sketched in 1.2 above. First it is shown how, in principle, the verification of correctness can be reduced to the verification of equality. Typ-functions for the various languages are discussed. Then we present the algorithmic system (like system III above) and an "intermediate" system (like system II). However, the situation is more complicated than sketched above, because the equivalence proofs in 4.3.2 and 4.3.3 are also used for proving the socalled *strengthening rule* superfluous (see below). Finally some remarks on the actual verification are made (sec. 4.4).

1.6. Complication 1: the strengthening rule

Of course, if an expression or a formula is correct relative to a book and a context, its constants are in the book and its free variables are in the context. The strengthening rule is connected with the converse question: In systems such as I, II above, which have rules for the transitivity of Q, it is a priori not clear that a correct equality $A\ Q\ B$ can be established via expressions containing only variables and constants occurring in A or in B. So it might be possible that a proof of correctness of A, or of $A\ E\ B$ *needs* correctness of expressions containing variables and constants outside A (and B).

Now for the sake of proving η- one-step-closure we have included a postulate, the strengthening rule, in our definition, which allows to skip "redundant" variables from the context. This appears to be a nasty rule because it might spoil the nice order on the correct expressions induced by the definition of correctness. See, e.g., sec. 2.10.3 and 2.14.1.

The proof that the rule is superfluous, runs roughly as follows: let $\vdash_I$, $\vdash_{II}$ and $\vdash_{III}$ stand for the correctness predicate in system I (as in 1.2, *with* strengthening rule), system II (as in 1.2, without strengthening rule), and the algorithmic system III (without strengthening rule), respectively. As in 1.2, $\vdash_{III} \Rightarrow \vdash_{II}$ (sec. 4.3.2). By CL for system I (sec. 3), we have $\vdash_{II} \Rightarrow \vdash_I$.

Since in the algorithmic definition strengthening is *provable* as in Nederpelt [51]), by CR (for I, so for II, so for III, in Chapter VI) we can conclude $\vdash_I \Rightarrow \vdash_{III}$, which closes the circle (sec. 4.3.3).

1.7. Complication 2: definitional 2-constants in the presence of type-inclusion.

The rule of type-inclusion in AUT-QE allows us to infer $A \mathsf{E} \tau$ from $A \mathsf{E} [x{:}\alpha]\tau$. This shows how uniqueness of types gets lost in AUT-QE (but only for 2-expressions A). For the restricted form which we can prove instead we refer to sec. 3.2.6.1.

A peculiarity, due to the combination of definitional 2-constants and type-inclusion, is that rule LQ is violated too in AUT-QE. Example: if $\alpha \mathsf{E} \tau$, $A \mathsf{E} [x{:}\alpha]\tau$ (relative to empty context, say), then the scheme

$$d := A * d \mathsf{E} \tau \qquad \text{(also with empty context)}$$

is correct in AUT-QE. Now $d \mathsf{Q} A$, $A \mathsf{E} [x{:}\alpha]\tau$ but *not* $d \mathsf{E} [x{:}\alpha]\tau$. So, in AUT-QE, definitional 2-constants are not only used as abbreviations but also for cutting down the type of the expression abbreviated. As a consequence of this, definitional 2-constants in AUT-QE can lead to *unessential extensions*, which are not *definitional* extensions (sec. 3.3.2).

One might wonder why we do not take more liberal variants of AUT-QE, which allows $d \mathsf{E} [x{:}\alpha]\tau$ as well. In fact, we mention such a variant AUT-QE* somewhere for technical reasons (sec.3.3.11), but we do not think that this way of ignoring the typ of a definitional constant is suitable for practical purposes.

Part of our motivation runs as follows:
First, we do not want it for definitional 3-constants, where the definition part can stand for a long proof, and the typ represents a short theorem (I.5.2). So, we do not like it for 2-constants, for the sake of uniformity.

Notice, however, that the definition of μ for the weakly normable expressions (IV.4.4.1) actually ignores the typ of the defined constants and only takes the def into account (otherwise μ could change by reduction).

V. 2. On the E-definition

2.1. The book-and-context part of the E-definition

2.1.1. The correct expressions with respect to a book and a context

form a system of admissible expressions, i.e. a restricted pretyped system, in the sense of IV.3. The correctness of books, contexts and expressions is defined simultaneously with the correctness of E-formulas $A\ \mathsf{E}\ B$ and Q-formulas $A\ \mathsf{Q}\ B$.

The symbol $\vdash$ stands for correctness; the notation for the correctness of contexts (w.r.t. $\mathcal{B}$), expressions, E- and Q-formulas (w.r.t. $\mathcal{B}$ and ξ) is respectively $\mathcal{B};\xi\vdash$, $\mathcal{B};\xi\vdash A$, $\mathcal{B};\xi\vdash A\ \mathsf{E}\ B$ and $\mathcal{B};\xi\vdash A\ \mathsf{Q}\ B$. The symbols E and Q are assumed to bind tighter than $\vdash$.

2.1.2. For brevity we sometimes write "$\mathcal{B};\xi\vdash A\ \mathsf{E/Q}\ B$" instead of "$\mathcal{B};\xi\vdash A\ \mathsf{E}\ B$ respectively $\mathcal{B};\xi\vdash A\ \mathsf{Q}\ B$", and "$\mathcal{B};\xi\vdash A(\mathsf{E/Q}\ B)$" instead of "$\mathcal{B};\xi\vdash A$ respectively $\mathcal{B};\xi\vdash A\ \mathsf{E}\ B$ resp. $\mathcal{B};\xi\vdash A\ \mathsf{Q}\ B$". So statements containing this kind of shorthand have to be read two or three times, each time with a different interpretation.

2.1.3. As in IV.3, if $\mathcal{B};\xi\vdash A$ then A is a $\mathcal{B};\xi$-expression and hence has a degree. If $\mathcal{B};\xi\vdash A\ \mathsf{E}\ B$ or $\mathcal{B};\xi\vdash A\ \mathsf{Q}\ B$ then B is a $\mathcal{B};\xi$-expression and has a degree, too. The rules for the formation of books and contexts are precisely as in IV.3.3.2. The two additional restrictions (see IV.3.3.3) are as follows:

(1) (*inhabitable degree condition*) an expression α can only act as the typ of a constant in a scheme or as the typ of a variable in a context, if its degree is 1 or 2.

(2) (*compatibility of* def *and* typ) in a scheme $\xi * d(\bar{x}) := \Delta * d(\bar{x})\ \mathsf{E}\ \Gamma$ it is required that $\mathcal{B};\xi\vdash\Delta\ \mathsf{E}\ \Gamma$, where $\mathcal{B}$ is the preceding book.

2.2. Some notational conventions

2.2.1. We often assume implicitly a fixed correct book $\mathcal{B}$ and a fixed context ξ, correct w.r.t. $\mathcal{B}$. I.e., if $\mathcal{B};\xi,\eta\vdash$ then we write

$$\eta\vdash A(\mathsf{E/Q}\ B) \quad \text{for} \quad \mathcal{B};\xi,\eta\vdash A(\mathsf{E/Q}\ B)$$

and just

$$A\ \mathsf{E/Q}\ B \quad \text{for} \quad \mathcal{B};\xi\vdash A\ \mathsf{E/Q}\ B$$

(so for formulas we omit the $\vdash$-symbol in this case).

2.2.2. At some places in the definition the degree of expressions is explicitly displayed as a superscript:

$$\vdash^{i} A\,(\mathsf{E}/\mathsf{Q}\ B) \ \Leftrightarrow \vdash A\,(\mathsf{E}/\mathsf{Q}\ B) \text{ and degree } (A) = i$$

2.2.3. Formulas like $A_1\ \mathsf{E}\ A_2\ \mathsf{Q}\ A_3\ \mathsf{E}\ A_4$ are used as abbreviation for

$$A_1\ \mathsf{E}\ A_2 \quad \text{and} \quad A_2\ \mathsf{Q}\ A_3 \quad \text{and} \quad A_3\ \mathsf{E}\ A_4 \text{ etc.}$$

2.3. The expression-and-formula part of the definition: expressions

The rules for the correctness of expressions and formulas fall apart in six groups labeled I to VI. We start with group I (correctness of 1-expressions) and group II (correctness of non 1-expressions).

I. correctness of 1-expressions

I.1. τ-rule: $\vdash^{1}\tau$

I.2. abstraction rule: $\vdash^{2}\alpha,\ x\ \mathsf{E}\ \alpha \vdash^{1} A \Rightarrow \vdash^{1}[x{:}\alpha]A$

I.3. application rule: $A\ \mathsf{E}\ \alpha,\ \vdash^{1} B\ \mathsf{Q}\ [x{:}\alpha]C \Rightarrow \vdash^{1}\{A\}B$

I.4. instantiation rule: if the scheme of d is in $\bar{B}$, with context $\bar{y}\ \mathsf{E}\ \bar{\Xi}$, and d is a 1-constant then $\bar{B}\ \mathsf{E}\ \bar{\beta}[\bar{y}/\bar{B}] \Rightarrow \vdash^{1} d(\bar{B})$

Notice, that the degree of A is indeed 1, if $\vdash^{1} A$ is derived by the above rules.

II. correctness of non-1-expressions

II. $\qquad A\ \mathsf{E}\ B \Rightarrow \vdash A$

2.4. The expression-and-formula part: E-formulas

The rules of group III, below, in combination with rule II, also serve as the formation rules for the non-1-expressions. Group IV contains the type modification rules.

III. Formation of non-1-expressions

III.1. copy rule: $\xi \equiv \ldots,\ x\ \mathsf{E}\ \alpha, \ldots \Rightarrow x\ \mathsf{E}\ \alpha$

III.2. abstraction rules: if $\vdash^{2}\alpha$ then

III.2.A. $x\ \mathsf{E}\ \alpha \vdash B\ \mathsf{E}\ \tau \Rightarrow \vdash[x{:}\alpha]B\ \mathsf{E}\ \tau$

III.2.B^{i}. $x\ \mathsf{E}\ \alpha \vdash^{i+1} B\ \mathsf{E}\ C \Rightarrow \vdash^{i+1}[x{:}\alpha]B\ \mathsf{E}\ [x{:}\alpha]C$

So of the latter are two versions, III.2.B^{1} and III.2.B^{2}.

III.3. application rules: if A E α then

III.3.A. B E $[x{:}\alpha]C \Rightarrow \{A\}B$ E $C[\![x/A]\!]$

III.3.B. B E C E $[x{:}\alpha]D \Rightarrow \{A\}B$ E $\{A\}C$

III.4. instantiation rule: if the scheme of c is in $\mathcal{B}$, with context $\bar{y}$ E $\bar{\beta}$, then

$$\bar{B} \text{ E } \bar{\beta}[\![\bar{y}/\bar{B}]\!] \Rightarrow c(\bar{B}) \text{ E } \mathrm{typ}(c)[\![\bar{y}/\bar{B}]\!]$$

Note: Below we shall prove A E $B \Rightarrow \vdash B$ (*correctness of categories*), which is not explicitly required here.

IV. Type modification rules

IV.1. type conversion: B E C, C Q $D \Rightarrow B$ E D

IV.2. type-inclusion: B E $[\bar{x}{:}\bar{\alpha}][y{:}\beta]\tau \Rightarrow B$ E $[\bar{x}{:}\bar{\alpha}]\tau$
(where $[\bar{x}{:}\bar{\alpha}]$ stands for $[x_1{:}\alpha_1]\ldots[x_k{:}\alpha_k]$)

2.5. The expression-and-formula part: Q-formulas

The rules for the correctness of Q-formulas form group V.

V. Correctness of Q-formulas

V.1. reflexivity: $\vdash A \Rightarrow A$ Q A

V.2. Q-propagation: A Q B, $\vdash C$, ($B > C$ or $C > B$) $\Rightarrow A$ Q C

Note: this is indeed the most restricted version of Q, see sec. 1.2.

2.6. The strengthening rule

This is a technical rule, which we use in the proof of η-CL, but afterwards, i.e. after having proved CL and (with help of CL)CR, as in sec. 1.6, prove superfluous. It is called *strengthening* rule because it permits to remove assumptions from the context. We say that η is a *subcontext* of ξ, for short η sub ξ, if the sequence of E-formulas of η is a subsequence of the sequence of E-formulas of ξ. So,

$$\eta \text{ sub } \xi \Rightarrow \eta \text{ sub } (\xi, x \text{ E } \alpha) \text{ and } (\eta, x \text{ E } \alpha) \text{ sub } (\xi, x \text{ E } \alpha)$$

VI. The strengthening rule

If $\mathcal{B};\xi_0$, ξ_0 sub ξ, $\xi_0 \equiv \bar{x}$ E $\bar{\alpha}$ and $\forall_y (y \in \mathrm{FV}(A) \Rightarrow y \in \bar{x})$, then

VI.1. $\mathcal{B};\ \xi \vdash A \Rightarrow \mathcal{B};\xi_0 \vdash A$

If, furthermore, $\forall_y (y \in \mathrm{FV}(B) \Rightarrow y \in \bar{x})$, then

VI.2. $\mathcal{B};\ \xi \vdash A\ \mathsf{E/Q}\ B \Rightarrow \mathcal{B};\xi_0 \vdash A\ \mathsf{E/Q}\ B$

2.7. Degree considerations

2.7.1. Degree restrictions play a minor role in the E-definition. It is rather intended that the degree specifications of the various languages (see below) are satisfied automatically by a suitable choice of the rules of the E-definitions.

We define (the notion of being a domain degree, etc.):
$\vdash [x:\alpha]B \Rightarrow \alpha$ has *domain degree* and B has *value degree*.
$\vdash \{A\}B \Rightarrow A$ has *argument degree* and B has *function degree*.

2.7.2. The degree specifications for the *regular languages* AUT-68, AUT-QE and AUT-QE+ are:

(1) degrees admitted 1, 2 and 3, inhabitable degrees 1 and 2, domain degree 2 and argument degree 3

(2) value and function degree are as in the following scheme

	AUT-68	AUT-QE	AUT-QE+
function degree	3	2,3	1,2,3
value degree	2,3	1,2,3	1,2,3

Languages where all value degrees are also function degrees, are said to be *+-languages*: AUT-QE+ (and AUT-68+, AUT-QE*, to be defined later). Consequently AUT-68 and AUT-QE are *non-+-languages*.

2.7.3. No matter what rules are chosen, by induction on $\vdash$ (i.e. on the definition of correctness) it follows that

$$A\ \mathsf{E}\ B \Rightarrow A \text{ not of degree } 1$$

So no application expressions $\{C\}\mathcal{D}$ with degree $(C) = 1$ and no instantiation expressions $c(\bar{C})$ where some C_j has degree 1, are formed, and the rules III.4 and III.3.A. do not give rise to substitution with 1-expressions (in the categories). Hence, also by induction on $\vdash$,

(1) $\quad A\ \mathsf{Q}\ B \Rightarrow (\text{degree } (A) = 1 \leftrightarrow \text{degree } (B) = 1)$

(2) $\quad A\ \mathsf{E}\ B \Rightarrow (\text{degree } (A) = 2 \leftrightarrow \text{degree } (B) = 1)$

2.7.4. This shows, together with the explicit degree restriction in the rules I.2 and III.2, that the expressions formed and the substitutions involved are weakly degree correct (cf. Ch. IV.4.4.2). The inhabitable degree restriction guarantees that only expressions of degrees 1, 2 and 3 are formed. So, the specifications of 2.7.2.(1) are fulfilled and

$$A \in B \Rightarrow \text{degree } (A) = \text{degree } (B) + 1$$

$$A \mathbin{Q} B \Rightarrow \text{degree } (A) = \text{degree } (B)$$

and all the substitutions generated by the rules are degree correct: If $\bar{A}$ is substituted for $\bar{x}$ then, for all i, degree (A_i) = degree (x_i).

2.8. Specification of the languages

2.8.1. The rules

The difference between the definitions of the various regular languages only concerns the rules of abstraction, application and type-inclusion. All the other rules, and also III.2.B^2 (for abstraction expressions of degree 3) and III.3.A (application) are present in each of the definitions.

For the rest the situation is as follows

	AUT-68	AUT-QE	AUT-QE+
abstraction	III.2.A	III.2.B^1, I.2	III.2.B^1, I.2
application		III.3.B	III.3.B, I.3
type incl. rule	no	yes	yes

Note: Below it will turn out that

(1) III.2.A is a derived rule of AUT-QE and AUT-QE+.
(2) III.3.B and IV.2 (type-inclusion) are derived rules of AUT-68.

So, after all, in AUT-68 all the rules except III.2.B^1, I.2 and I.3 are valid; AUT-QE and AUT-QE+ have additionally III.2.B^1 and I.2 and, besides, AUT-QE+ has I.3.

2.8.2. The reduction relation

For definiteness we agree that > in the Q-rule V.2 stands for disjoint one-step reduction $>_1$. So it satisfies the monotonicity conditions, e.g.

$$A > A',\ B > B' \Rightarrow \{A\}B > \{A'\}B'$$

with the important consequence that

$$\bar{A} > \bar{A}' \Rightarrow B[\![\bar{A}]\!] > B[\![\bar{A}']\!]$$

In any case the reduction relation includes β-reduction, but we leave open the presence of η- and δ-reduction. Of course, if no definitional constants are in the book then there is no δ-reduction.

We assume that AUT-68 has no definitional 1-constants (because, modulo the elimination of abbreviations, the only 1-expression in AUT-68 is τ).

The rules of strengthening will only be present in languages with η-reduction.

2.9. The substitution theorem

2.9.1. For the E-definition (in contrast with the algorithmic definition it is easy to show the substitutivity: correctness of expressions and formulas is preserved under correct substitutions, i.e. substitution with correct expressions of the right types.

For technical reasons we start with a weak form of substitution, compare α-reduction.

2.9.2. Theorem (*renaming of contexts*): If $\xi \equiv \bar{x}\ \mathsf{E}\ \bar{\alpha}$ and $\Sigma' \equiv \Sigma[\![\bar{x}/\bar{x}']\!]$, all x_i' are mutually distinct, then (with $\xi' :\equiv \bar{x}'\ \mathsf{E}\ \bar{\alpha}'$)

$$\xi \vdash A\,(\mathsf{E}/Q\ B) \Rightarrow \xi' \vdash A'\,(\mathsf{E}/Q\ B')$$

and the correctness proofs of both sides of the implication sign are equally long.

Proof: induction on $\vdash$. □

2.9.3. An easy corollary of this is the *weakening theorem*, the converse of strengthening: if ξ_0 sub ξ then

$$\xi \vdash, \ \xi_0 \vdash A\,(\mathsf{E}/Q\ B) \Rightarrow \xi \vdash A\,(\mathsf{E}/Q\ B)$$

Proof: induction on $\xi_0 \vdash A\ (\mathsf{E}/Q\ B)$. □

As a corollary of this we can prove that in a derivation of correctness the application of strengthening can be *postponed* to the end of the derivation.

2.9.4. Now we come to the *simultaneous substitution theorem*: if $\eta \equiv \bar{y}\ \mathsf{E}\ \bar{\beta}$, then

$$\bar{B}\ \mathsf{E}\ \bar{\beta}[\![\bar{y}/\bar{B}]\!],\quad \eta\vdash C(\mathsf{E}/\mathsf{Q}\ D) \Rightarrow C[\![\bar{y}/\bar{B}]\!](\mathsf{E}/\mathsf{Q}\ D[\![\bar{y}/\bar{B}]\!])$$

Proof: By induction on $\eta \vdash C(\mathsf{E}/\mathsf{Q}\ D)$. We treat just some of the cases, distinguished according to the last rule applied in the derivation. Abbreviate $\Sigma[\![\bar{y}/\bar{B}]\!]$ to Σ^*.

Last rule is III.2.B^i: Assume $\eta\vdash^2 C_1$ and $\eta,\ z\ \mathsf{E}\ C_1 \vdash^{i+1} C_2\ \mathsf{E}\ D_2$. By the ind. hyp. and by 2.7.4, $\vdash^2 C_1^*$. By the copy rule $z\ \mathsf{E}\ C_1^* \vdash z\ \mathsf{E}\ C_1^*$ (if necessary, i.e. if z in ξ, rename the implicit context ξ to ξ'). Now, by weakening, we can apply the ind. hyp. with the extended substitution $[\![\bar{y},z/\bar{B},z]\!]$ to $\eta,z\ \mathsf{E}\ C_1 \vdash^{i+1} C_2\ \mathsf{E}\ D_2$. This gives $z\ \mathsf{E}\ C_1^* \vdash^{i+1} C_2^*\ \mathsf{E}\ D_2^*$ and, by III.2.B^i, $\vdash[z{:}C_1^*]C_2^*\ \mathsf{E}\ [z{:}C_1^*]D_2^*$, q.e.d. Possibly one must first rename ξ' back to ξ again.

Last rule in V.2: Assume $\eta\vdash C_1\ \mathsf{Q}\ C_2$, $\eta\vdash C_3$, $C_2 > C_3$. By the ind. hyp. $\vdash C_1^*\ \mathsf{Q}\ C_2^*$ and $\vdash C_3^*$. Since $C_2^* > C_3^*$, $\vdash C_1^*\ \mathsf{Q}\ C_3^*$, q.e.d.

2.9.5. Corollary (*single substitution theorem*):

$$A\ \mathsf{E}\ \alpha,\quad x\ \mathsf{E}\ \alpha\vdash B(\mathsf{E}/\mathsf{Q}\ C) \Rightarrow B[\![x/A]\!](\mathsf{E}/\mathsf{Q}\ C[\![x/A]\!])$$

2.10. Some easy properties

2.10.1. On abstraction

In addition to the remark in 2.3, after rule I.4, we can say that the last inference in a proof of $\vdash^1 A$ must be rule VI.1 or one of the rules I. In particular, if $\xi_0\vdash^1[x{:}\alpha]A$, this can only follow from $\xi,x\ \mathsf{E}\ \alpha\ \vdash^1 A$ for some ξ with ξ_0 sub ξ (since sub is transitive). So application of VI.1 gives $\xi_0,\ x\ \mathsf{E}\ \alpha\vdash^1 A$. Similarly, if $\vdash^{i+1} A$, the last rule in proving this is VI.1 or II. So in the proof of correctness of $\xi_0\vdash^{i+1} A$ we can retrace some $\xi\vdash A\ \mathsf{E}\ B$, where ξ_0 sub ξ. Hence if $\xi_0\vdash^{i+1}[x{:}\alpha]A$, in its derivation we can find $\xi,\ x\ \mathsf{E}\ \alpha\vdash^{i+1} A\ \mathsf{E}\ B$ for some B and ξ, with ξ_0 sub ξ. By application of II and VI.1 we get $\xi_0,\ x\ \mathsf{E}\ \alpha\vdash A$. Resuming we have

$$\xi\vdash[x{:}\alpha]A \Rightarrow \xi,\ x\ \mathsf{E}\ \alpha\vdash A$$

2.10.2. Correctness of categories

In the rules of the definition, having $A\ \mathsf{E}\ B$ as their consequence,

it is not explicitly required that $\vdash B$. For the copy rule this *correctness of categories* follows from weakening, for III.2.A from the τ-rule, for III.3.A from the single substitution theorem (use induction on $\vdash$), for III.4 from the simultaneous substitution theorem etc. So, we have *correctness of categories*

$$A \; E \; B \Rightarrow \vdash B$$

2.10.3. Abstraction again

Assume that ξ_0, $x \; E \; \alpha \vdash^i A$, A of value degree, degree(α) = 2. If $i = 1$ then from I.2 we infer $\xi_0 \vdash [x:\alpha]A$. If $i > 1$ then, as above, we can retrace some ξ_1, $x \; E \; \alpha$, $\xi_2 \vdash^i A \; E \; B$ with ξ_0 sub ξ_1 and the transition from ξ_1, $x \; E \; \alpha$, $\xi_2 \vdash A$ to ξ_0, $x \; E \; \alpha \vdash A$ follows from applications of strengthening. By the weakening theorem, we can extend the context to ξ_1, $x \; E \; \alpha$, $\xi_2, x' \; E \; \alpha$, with some new x'. By the substitution theorem we can infer ξ_1, $x \; E \; \alpha$, ξ_2, $x' \; E \; \alpha \vdash A[\![x/x']\!] \; E \; B[\![x/x']\!]$. In case we can apply III.2.B (this depends on the language under consideration) we get ξ_1, $x \; E \; \alpha$, $\xi_2 \vdash [x:\alpha]A \; E \; [x:\alpha]B$. Otherwise the language is AUT-68, $i = 2$, $B \equiv \tau$ and application of III.2.A gives ξ_1, $x \; E \; \alpha$, $\xi_2 \vdash [x:\alpha]A \; E \; \tau$. Anyhow, rule II and iterated use of strengthening give $\xi_0 \vdash [x:\alpha]A$. Resuming,

$$(\text{degree}(\alpha) = 2, \; A \text{ of value degree}, \; x \; E \; \alpha \vdash A) \Leftrightarrow \vdash [x:\alpha]A \; .$$

Note: the results in 2.9 and 2.10 are also valid, and simpler to prove, if η-reduction (and strengthening) is not present.

2.11. On the Q-rules

2.11.1. Clearly Q is the equivalence relation generated by $>_{\vdash}$, i.e. the restriction of $>$ to the correct expressions. So $A \; Q \; B$ means precisely that

$\vdash A$ and $\vdash B$ and there are correct $C_1, \ldots, C_k$ such that

$$A > C_1 > \ldots < C_{i-1} < C_i > C_{i+1} > \ldots < C_{j-1} < C_j > C_{j+1} > \ldots < C_k < B$$

(where possibly, in view of strengthening, the C_i in between are correct w.r.t. extended contexts).

2.11.2. An alternative rule of Q-propagation is

V.2' $\quad A\ Q\ B,\ \vdash C,\ B \vdash C \Rightarrow A\ Q\ C$

If the language definition has this rule, Q becomes $\sim_{\vdash}$, i.e. $(\vdash_{\vdash})^*$ (sec.1.2,II.4.7), i.e. the transitive closure of the restriction of $\vdash$ to the correct expressions.

So, no matter what other rules there are in the definition of correctness,

V.2' ⇒ V.2

and

CL, V.2 ⇒ V.2'

2.11.3. An even stronger rule for Q, also including reflexivity is

V.2" $\quad \vdash A,\ \vdash B,\ A = B \Rightarrow A\ Q\ B$

Assuming the (full) CR-theorem, i.e. CR for all, not just the correct expressions, which is the case if η-reduction is not present, we get:

(V.1, V.2') ⇒ V.2"

2.12. On type-conversion

2.12.1. The Q-formulas (and the Q-rules, see below) can be avoided completely by reformulating IV.1, the type-conversion rule to

IV.1': $A\ E\ B,\ \vdash C,\ (B > C$ or $C > B) \Rightarrow\ A\ E\ C$

And, corresponding to V.2' rather than to V.2,

IV.1": $A\ E\ B, \vdash C,\ B \vdash C \Rightarrow A\ E\ C$

As in 2.11.2, IV.1" ⇒ IV.1' and CL, IV.1' ⇒ IV.1".

Corresponding to V.2" is the alternative rule

IV.1"' : $A\ E\ B, B = C, \vdash C \Rightarrow A\ E\ C$

2.12.2. The system with Q-formulas, Q-rules V.1 and V.2, and rule IV.1 is indeed a conservative extension of the system without Q but with the corresponding type-conversion rule instead. First we have

IV.1, V.1, V.2 ⇒ IV.1',

respectively

IV.1, V.1, V.2' ⇒ IV.1",

respectively

IV.1, V.2" ⇒ IV.1"' ,

so the Q-system is an extension of the Q-less one.

Secondly, the expressions and E-formulas, correct in a Q-system are also correct in the corresponding Q-less system.

2.12.3. Notice, that in the presence of η, rule IV.1"' (so rule V.2" too!) is inconsistent in the sense that it gives rise to anomalies such as self-application. This fact is connected with the βη-CR-problem, solved in Chapter VI.
Example: if $\alpha\ E\ \tau$ then $\vdash[x{:}\alpha]\alpha$ and $\vdash[y{:}[x{:}\alpha]\alpha]\alpha$. Further
$[x{:}\alpha]\alpha$ = (by β) $[x{:}\alpha]\{x\}[y{:}[x{:}\alpha]\alpha]\alpha$ = (by η) $[y{:}[x{:}\alpha]\alpha]\alpha$.
So, if $f\ E\ [x{:}\alpha]\alpha$ then $\{f\}f\ E\ \alpha$.

2.13. On type-inclusion

2.13.1. Iterated use of the rule of type-inclusion gives

$$A\ E\ [\bar{x}{:}\bar{\alpha}][\bar{y}{:}\bar{\beta}]\tau \Rightarrow A\ E\ [\bar{x}{:}\bar{\alpha}]\tau$$

so

$$A\ E\ [\bar{x}{:}\bar{\alpha}]\tau \Rightarrow A\ E\ \tau.$$

This shows that AUT-68 is a sublanguage of AUT-QE: all the correct books, contexts, expressions and formulas of AUT-68 are also correct in AUT-QE.
Proof: Rule III.2.A, not in the definition of AUT-QE, can be derived from III.2.B[1] and IV.2. For, let $x\ E\ \alpha \vdash B\ E\ \tau$. Then $\vdash[x{:}\alpha]B\ E\ [x{:}\alpha]\tau$ so $\vdash[x{:}\alpha]B\ E\ \tau$, q.e.d.

2.13.2. Conversely, rule IV.2 is (vacuously) a derived rule of AUT-68, because all the correct AUT-68 1-expressions δ-reduce to τ.

2.14. The form of derivations

2.14.1. We called the rules III the formation rules of non-1-expressions. This is because, in a proof of $\xi_0 \vdash^{i+1} A$, we can retrace some $\xi \vdash A \mathrel{E} B$ and $\xi_1 \vdash A \mathrel{E} C$, such that (i) the last rule applied in proving $\xi_1 \vdash A \mathrel{E} C$ is the formation rule of A, i.e. one of the rules III, (ii) the transition from $\xi_1 \vdash A \mathrel{E} C$ to $\xi \vdash A \mathrel{E} B$ is by iterated use of VI.2 and type conversion, (iii) the transition from $\xi \vdash A \mathrel{E} B$ to $\xi_0 \vdash A$ is by using VI.2, II, and VI.1. So, in case there is no type-inclusion applied, e.g. if $i > 1$, we have (use weakening) $\xi_1 \vdash B \mathrel{Q} C$. Below we introduce a symbol covering the relation between B and C in case type-inclusion is involved.

2.14.2. The new relation $\sqsubset$ can be defined as follows

(i) $\vdash [x{:}\alpha]A,\ x \mathrel{E} \alpha \vdash A \sqsubset B \Rightarrow [x{:}\alpha]A \sqsubset [x{:}\alpha]B$

(ii) $A \mathrel{Q} B \Rightarrow A \sqsubset B$

(iii) $\sqsubset$ is transitive

(iv) $\vdash^1 \alpha \Rightarrow \alpha \sqsubset \tau$ □

Clearly, $\sqsubset$ is a reflexive and transitive relation on the correct expressions, including Q and type-inclusion, which on the non-1-expressions coincides with Q (use 2.10.3). The type modification rule can now be contracted to one rule

IV. $A \mathrel{E} B,\ B \sqsubset C \Rightarrow A \mathrel{E} C$

And, for ξ_1, B and C as in 2.14.1 we have $\xi_1 \vdash C \sqsubset B$ now.

2.14.3. So, in a proof of $[x{:}\alpha]B \mathrel{E} D$ we can retrace $x \mathrel{E} \alpha \vdash B \mathrel{E} C$ with $[x{:}\alpha]C \sqsubset D$.

Similarly, in a proof of $\{A\}B \mathrel{E} D$ we can retrace either

(i) $B \mathrel{E} [x{:}\alpha]C$ with $C[\![A]\!] \sqsubset D$, $A \mathrel{E} \alpha$, or

(ii) $B \mathrel{E} C \mathrel{E} [x{:}\alpha]E$ with $\{A\}C \sqsubset D$, $A \mathrel{E} \alpha$.

And, in a proof of $c(\bar{C}) \sqsubset D$ we can retrace some

$c(\bar{C}) \mathrel{E} \mathrm{typ}(c)[\![\bar{C}]\!] \sqsubset D$.

2.14.4. Above, we used already

$$\vdash [x:\alpha]A,\ x\ \mathsf{E}\ \alpha \vdash A\ \mathsf{Q}\ B \Rightarrow [x:\alpha]A\ \mathsf{Q}\ [x:\alpha]B$$

The other monotonicity rule

$$\alpha\ \mathsf{Q}\ \beta,\ \vdash [x:\alpha]A \Rightarrow [x:\alpha]A\ \mathsf{Q}\ [x:\beta]A$$

follows by induction on Q, using the substitution theorem.
However, we do not know yet

$$A\ \mathsf{Q}\ B\ ,\ C\ \mathsf{Q}\ D \Rightarrow \{A\}C\ \mathsf{Q}\ \{B\}D$$

and consequently, it is a priori not clear that (uniqueness of types for 3-expressions)

$$\vdash^3 A\ \mathsf{E}\ \alpha,\ A\ \mathsf{E}\ \beta \Rightarrow \alpha\ \mathsf{Q}\ \beta\ .$$

This (and its weaker counterpart for 2-expressions) will not be proved before the next section (3.2.4, 3.2.6).

2.15. On the application rules

2.15.1. In AUT-68, where no 1-abstraction expressions are formed, the rule III.3.B is vacuously a derived rule, viz. there are no B with $B\ \mathsf{E}\ C\ \mathsf{E}\ [x:\alpha]D$.

Since, in AUT-QE and AUT-QE+,

$$\vdash^2 [x:\alpha]C \Rightarrow [x:\alpha]C\ \mathsf{E}\ [x:\alpha]D$$

we can restrict the rule III.3.A

$$A\ \mathsf{E}\ \alpha,\ B\ \mathsf{E}\ [x:\alpha]C \Rightarrow \{A\}B\ \mathsf{E}\ C[\![A]\!]$$

to the case where degree $(C) = 1$.

2.15.2. As an alternative to III.3.B (and to III.3.A if I.3 is present) we mention

III.3.B': $\vdash \{A\}C,\ B\ \mathsf{E}\ C \Rightarrow \{A\}B\ \mathsf{E}\ \{A\}C$

The following equivalences hold

(I.3, III.3.A, III.3.B) $\Leftrightarrow$ (I.3, III.3.B')

(III.3.A, III.3.B) $\Leftrightarrow$ (III.3.A, III.3.B') .

Proof: e.g. that III.3.A is a derived rule in presence of I.3 and III.3.B'. Let $A \in \alpha$, $B \in [x:\alpha]C$. By I.3 (and III.3.B', if degree$(C) = 2$), $\vdash \{A\}[x:\alpha]C$. By the single substitution theorem $\vdash C[\![A]\!]$. So by III.3.B' and type-conversion $\{A\}B \in C[\![A]\!]$.

2.15.3. Notice that in the presence of η-reduction rule III.3.A by itself is sufficient, because

$$\eta, \text{III.3.A} \Rightarrow \text{III.3.B} .$$

Proof: assume $A \in \alpha$, $B \in C \in [x:\alpha]D$. Then $x \in \alpha \vdash x \in \alpha$, so by III.3.A, $x \in \alpha \vdash \{x\}C \in D$ and by abstraction $\vdash [x:\alpha]\{x\}C \in [x:\alpha]D$. By II and type-conversion $B \in [x:\alpha]\{x\}C$ $(x \notin \mathrm{FV}(C))$, so by III.3.A. $\{A\}B \in \{A\}C$, q.e.d.

2.16. An E-definition for Λ and Λ+

2.16.1. In order to adapt the E-definition to Λ and Λ+ we must first drop the inhabitable degree condition, and the restriction to α of degree 2 in the abstraction rules I.2 and III.2. The rule of type-inclusion and rule III.2.A must be skipped but III.2.B^i is permitted for all i. A suitable combination of application rules is I.3 and III.3.B' for Λ+, and III.3.A and III.3.B' for Λ. An alternative for III.3.B' is an extended form of III.3.B

$$A \in \alpha,\ B \in C_1 \in \ldots \in C_k \in [x:\alpha]D \Rightarrow \{A\}B \in \{A\}C_1 .$$

2.16.2. Degree considerations for Λ and Λ+ are indeed more involved than those in 2.7. Of course we can show weak degree correctness, as in 2.7, but we must know more in order to establish degree correctness. See Ch. VII, sec. 2.2.
The various properties proved above, such as substitutivity, correctness of categories, etc. etc. simply go through for the E-versions of Λ and Λ+.

V.3. The actual closure proof

3.1. Heuristics

3.1.1. The first idea which comes to mind about proving *closure*, CL

CL: $\vdash A,\ A \geq B \Rightarrow \vdash B$

is simply to prove one-step closure, CL1

CL1: $\vdash A,\ A > B \Rightarrow \vdash B$

by induction on $\vdash A$ and then use induction on $\geq$.
Among the possible ways of one-step reduction we distinguish the main or "outside" reductions

(β) $\{A\}[x{:}B]C > C[\![A]\!]$

(η) $x \notin \mathrm{FV}(A) \Rightarrow [x{:}\alpha]\{x\}A > A$

(δ) $d(\bar{A}) > \mathrm{def}(d)[\![A]\!]$

and the "inside" reductions which follow by the monotonicity rules

(appl) $A > A',\ B > B' \Rightarrow \{A\}B > \{A'\}B'$

(abstr) $\alpha > \alpha',\ A > A' \Rightarrow [x{:}\alpha]A > [x{:}\alpha']A'$

(const) $\bar{A} > \bar{A}' \Rightarrow c(\bar{A}) > c(\bar{A}')$.

So we assume that > stands for disjoint one-step reduction. Now consider, e.g., the appl-case where the correctness of $\{A\}C$ follows from $A\ \mathsf{E}\ \alpha$, $B\ \mathsf{E}\ [x{:}\alpha]C$. Here the induction hypothesis, CL1 applied to A and to B, just tells us that $\vdash A'$ and $\vdash B'$ (where $A > A'$, $B > B'$), which is of course not enough to conclude $\vdash \{A'\}B'$. This suggests that we need *preservation of types*, PT

PT: $A\ \mathsf{E}\ \alpha,\ \vdash B,\ A \geq B \Rightarrow B\ \mathsf{E}\ \alpha$

or at least *one-step preservation of types*, PT1

PT1: $A\ \mathsf{E}\ \alpha,\ \vdash B,\ A > B \Rightarrow B\ \mathsf{E}\ \alpha$

additionally. Similarly with the const-case of one-step reduction.

3.1.2. So the next idea is to combine CL and PT to

CLPT: $\vdash A(\mathsf{E}\,\alpha),\ A \geq B \Rightarrow \vdash B(\mathsf{E}\ \alpha)$

(as the conjunction of the version with and the version without parentheses) and to use the same induction. I.e. first prove

CLPT1: $\vdash A\ (\mathsf{E}\ \alpha),\ A > B \Rightarrow \vdash B(\mathsf{E}\ \alpha)$

by induction on correctness and then use induction on $\geq$.
This works fine with all the inside reductions. E.g., consider once more the appl-case: $A\ \mathsf{E}\ \alpha$, $B\ \mathsf{E}\ [x{:}\alpha]C$, $A > A'$, $B > B'$. Now the induction hypothesis gives us $A'\ \mathsf{E}\ \alpha$, $B'\ \mathsf{E}\ [x{:}\alpha]C$ and $\{A'\}B'\ \mathsf{E}\ C[\![A']\!]$. Since $>$ is disjoint one-step reduction, $C[\![A]\!] > C[\![A']\!]$ so $C[\![A]\!]\ \mathsf{Q}\ C[\![A']\!]$ so $\{A'\}B'\ \mathsf{E}\ C[\![A]\!]$, q.e.d. The other cases of inside reductions are treated similarly, using some facts from the previous sections.
Then the outside reductions: δ and η do not cause major difficulties either. For δ use the simultaneous substitution theorem and the compatibility of *def* and *typ*, for η use the strengthening rule. But there is a problem with β-outside reduction. For, in order to conclude $\vdash C[\![A]\!]$ from $\vdash \{A\}[x{:}B]C$, we seem to need *soundness of applicability*, SA

SA $\vdash \{A\}[x{:}B]C \Rightarrow A\ \mathsf{E}\ B$

which would allow us to use the single substitution theorem.

3.1.3. Let us try to find out about SA. So consider the assumptions which can lead to the correctness of $\{A\}[x{:}B]C$.
E.g. $A\ \mathsf{E}\ \alpha$, $[x{:}B]C\ \mathsf{Q}\ [x{:}\alpha]D$ (resp. $[x{:}B]C\ \mathsf{E}\ [x{:}\alpha]D$). Then SA amounts to *uniqueness of domains*, UD

UD $[x{:}B]C\ \mathsf{Q}\ [x{:}\alpha]D \Rightarrow B\ \mathsf{Q}\ \alpha$

resp. *extended uniqueness of domains*, EUD

EUD $[x{:}B]C\ \mathsf{E}\ [x{:}\alpha]D \Rightarrow B\ \mathsf{Q}\ \alpha$

or: $A\ \mathsf{E}\ \alpha$, $[x{:}B]C\ \mathsf{E}\ D\ \mathsf{E}\ [x{:}\alpha]E$ (these are the assumptions of rule III.3.B). As in 2.14.3, for some F, $[x{:}B]C\ \mathsf{E}\ [x{:}B]F \sqsubset D$ (and in fact $[x{:}B]F\ \mathsf{Q}\ D$). So, in this case SA seems to require the *left-hand equality rule* LQ

LQ $A\ \mathsf{E}\ \alpha$, $A\ \mathsf{Q}\ B \Rightarrow B\ \mathsf{E}\ \alpha$

which would give $[x{:}B]F\ \mathsf{E}\ [x{:}\alpha]E$ and, by EUD, $A\ \mathsf{E}\ B$.
However, LQ $\Rightarrow$ PT. So, it appears that we cannot do SA separately beforehand (i.e. not if III.3.B is present) and then proceed with CLPT as sketched above.

3.1.4. In order to simplify matters, we first forget about type-inclusion. Then we may hope to be able to prove *uniqueness of types*, UT

UT: $\quad A \mathrel{E} \alpha,\ A \mathrel{E} \beta \Rightarrow \alpha \mathrel{Q} \beta$

If we assume UT then UD $\Rightarrow$ EUD and, besides, LQ and PT turn out to be equivalent. This may suggest us to incorporate the proof of SA in the proof of CLPT

But we do not have UT yet. If we try to prove UT by induction on the length of A, we come again in trouble with rule III.3.B. For, let $A_1 \mathrel{E} \alpha$, $A_2 \mathrel{E} B \mathrel{E} [x{:}\alpha]D$, $A_2 \mathrel{E} C \mathrel{E} [x{:}\alpha]E$. The ind. hyp. just gives us $B \mathrel{Q} C$ here, but we need more, viz. something like

(*) $\quad \vdash \{A\}B,\ B \mathrel{Q} C \Rightarrow \{A\}B \mathrel{Q} \{A\}C.$

(this is one half of the third monotonicity formula of sec. 2.14.4). Since a proof of (*) requires LQ in turn, UT cannot be isolated either. We might try to combine SA, UT and CLPT, i.e. to prove the necessary instances of SA and UT in the course of the proof of CLPT_1. A proof along these lines is indeed possible even if type-inclusion is present, but it has a complicated structure and it cannot easily be extended to languages with higher function degrees, such as Λ and $\Lambda+$.

3.1.5. Thus we prefer the alternative approach sketched below, which essentially runs as follows: first prove PT_1, UT and LQ by induction on degree, then prove SA and UD, and afterwards prove CL as indicated in 3.1.1. To this end we distinguish degree-i-versions of the various properties

$\mathrm{PT}_1^i \quad \vdash^i A \mathrel{E} \alpha,\ A > B,\ \vdash^i B \Rightarrow B \mathrel{E} \alpha$

$\mathrm{LQ}^i \quad \vdash^i A \mathrel{E} \alpha,\ A \mathrel{Q} B \Rightarrow B \mathrel{E} \alpha$

$\mathrm{UT}^i \quad \vdash^i A \mathrel{E} \alpha,\ A \mathrel{E} \beta \Rightarrow \alpha \mathrel{Q} \beta$

$(*^i) \quad \vdash^i B \mathrel{Q} C,\ \vdash \{A\}B \Rightarrow \{A\}B \mathrel{Q} \{A\}C$

$\mathrm{UD}^i \quad \vdash^i [x{:}\alpha]A \mathrel{Q} [x{:}\beta]B \Rightarrow \alpha \mathrel{Q} \beta$

$\mathrm{SA}^i \quad \vdash^i \{A\}[x{:}B]C \Rightarrow A \mathrel{E} B$

First notice that: $\quad \mathrm{PT}_1^i,\ \mathrm{UT}^i \Rightarrow \mathrm{LQ}^i$

and that: $\quad \mathrm{LQ}^i \Rightarrow (*^i)$

hence: $\quad \mathrm{LQ}^i \Rightarrow \mathrm{UT}^{i+1}$

We assume that the language under consideration is a non-+-language (see sec. 2.7). Then it is relatively easy to show UD^k and UT^{k+1} (ignoring type-inclusion), where k is the lowest value degree. Now let us try to prove PT_1^{i+1} by induction on correctness, where we assume PT_1^j, LQ^j and UT^{j+1} for $j \leq i$. An instructive example is the appl-case of inside reduction: $A > A'$, $B > B'$, $\vdash^{i+1}\{A\}B$, $\vdash^{i+1}\{A'\}B'$. It is no restriction to assume that both $\{A\}B$ and $\{A'\}B'$ originate from the extended application rule of 2.16.1: $A \in \alpha$, $A' \in \alpha'$, $B \in C_1 \in \ldots \in C_\ell \in [x{:}\alpha]D$, $B' \in C_1' \in \ldots \in C_{\ell'}' \in [x{:}\alpha']D'$ with degree (D) = degree (D') = k and $\ell = \ell'$. Then by the ind. hyp. we have $B' \in C_1$, so by UT^{i+1} $C_1 \mathrel{Q} C_1'$ and by LQ^i $C_1' \in C_2$. Then follows $C_2 \mathrel{Q} C_2'$ and $C_2' \in C_3$ etc. Finally we have $[x{:}\alpha]D \mathrel{Q} [x{:}\alpha']D'$ and by UD^k $\alpha \mathrel{Q} \alpha'$ so $A' \in \alpha$. Hence $\{A'\}B' \in \{A'\}C_1 < \{A\}C_1$, so $\{A'\}B' \in \{A\}C_1$, q.e.d.

From PT_1^{i+1} and UT^{i+1} we get LQ^{i+1}, and UT^{i+2}. So by induction, we get PT_1, LQ, (*) and UT.

3.1.6. It is clear that SA^{i+1} can be distilled from the proof of PT_1^{i+1}, but it can alternatively be given as follows. First, we have

$$LQ^{i+1},\ UD^i \Rightarrow UD^{i+1}$$

so we have UD. Now let $\vdash^{i+1}\{A\}[x{:}B]C$. Then (see sec. 2.15.2) either $A \in \alpha$, $[x{:}B]C \in [x{:}\alpha]D$, or $[x{:}B]C \in E$, $\vdash\{A\}E$. Further $[x{:}B]C \in [x{:}B]F$. So by UT we have either $[x{:}B]F \mathrel{Q} [x{:}\alpha]D$, or $[x{:}B]F \mathrel{Q} E$. Hence, either by UD we have $\alpha \mathrel{Q} B$, or by (*) we have $\vdash\{A\}[x{:}B]F$. So from LQ, UD and UT we get

$$SA^i \Rightarrow SA^{i+1}$$

and by induction SA.

3.2. Closure for βη-AUT-QE

3.2.1. For definiteness we present a rather detailed version of our closure proof here for βη-AUT-QE, i.e. AUT-QE without definitional constants and without δ-reduction. So the admitted degrees are 1, 2 and 3, the value degrees are 1, 2 and 3, the domain degree is 2 and the argument degree is 3.

The function degrees are just 2 and 3, so βη-AUT-QE is a non-+-language. So the reasoning of sec. 3.1.5 is valid, but for additional problems due to the presence of type-inclusion (viz. that UT is not

true and that not immediately ($PT_1 \Rightarrow LQ$) and ($UD \Rightarrow EUD$)). These problems are overcome by the introduction of a "canonical type" in sec. 3.2.4. below.

This canonical type also plays a role in the η-case of PT_1. Later we include definitional constants and δ-reduction, and application expressions of degree 1, thus extending our result to βηδ-AUT-QE + (in section 3.3).

A closure proof of βη-AUT-68 can easily be imitated from the proof below and is in fact somewhat easier because there is no type-inclusion.

3.2.2. We specify a set of rules (in shorthand, omitting contexts) for βη-AUT-QE, which according to the properties in 2.10-2.15 are equivalent to the rules indicated previously.

(i) $\vdash \tau$

(ii) $\ldots, x \mathsf{E} \alpha, \ldots \vdash x \ (\mathsf{E} \ \alpha)$

(iii) $x \mathsf{E} \alpha \vdash A \ (\mathsf{E} \ B) \Rightarrow \vdash [x{:}\alpha]A \ (\mathsf{E} \ [x{:}\alpha]B)$

(iv) $A \mathsf{E} \alpha, \vdash^2 B \mathsf{E} [x{:}\alpha]C \Rightarrow \vdash \{A\}B \ (\mathsf{E} \ C[\![A]\!])$

(v) $A \mathsf{E} \alpha, B \mathsf{E} C \mathsf{E} [x{:}\alpha]D \Rightarrow \vdash \{A\}B \ (\mathsf{E} \ \{A\}C)$

(vi) $\bar{A} \mathsf{E} \bar{\alpha}[\![\bar{A}]\!], \quad \bar{x} \mathsf{E} \bar{\alpha} * p(\bar{x}) \mathsf{E} P$ is a scheme $\Rightarrow \vdash p(\bar{A}) \ (\mathsf{E} \ P \ [\![\bar{A}]\!])$

(vii) $A \mathsf{E} B \sqsubset C \Rightarrow A \mathsf{E} C$

(viii) $\vdash A$, $A > B$ or $B > A$, $\vdash B \Rightarrow A \ \mathsf{Q} \ B$ (where $>$ is disjoint one step βη-reduction)

(ix) $A \ \mathsf{Q} \ B \ \mathsf{Q} \ C \Rightarrow A \ \mathsf{Q} \ C$

(x) $A \ \mathsf{Q} \ B \Rightarrow A \sqsubset B$

(xi) $\vdash^1 A \Rightarrow A \sqsubset \tau$

(xii) $x \mathsf{E} \alpha \vdash A \sqsubset B \Rightarrow [x{:}\alpha]A \sqsubset [x{:}\alpha]B$

(xiii) $A \sqsubset B \sqsubset C \Rightarrow A \sqsubset C$

(xiv) strengthening

3.2.3. On 1- expressions and type-inclusion

3.2.3.1. Since there are no 1-application expressions and no definitional constants all 1-expressions are of the form $[\bar{x}{:}\bar{\alpha}]\tau$, with $\bar{x}$

possibly empty. And, if $\vdash^1[x:\alpha]A$, $\vdash^1[x:\beta]B$, $[x:\alpha]A > [x:\beta]B$, then $\alpha > \beta$, A > B so α Q β and x E $\alpha \vdash A$ Q B. So, by induction on Q, we can show UD^1

$$\vdash^1[x:\alpha]A \ Q \ [x:\beta]B \Rightarrow \ \alpha \ Q \ \beta \ (\text{and } x \ E \ \alpha \vdash A \ Q \ B).$$

Then, by induction on $\sqsubset$, we get

$$\vdash^1[x:\alpha]A \sqsubset [x:\beta]B \Rightarrow \ \alpha \ Q \ \beta \ (\text{and } x \ E \ \alpha \vdash A \sqsubset B).$$

3.2.3.2. We introduced UT^i, uniqueness of types for expressions of degree i (i > 1),

UT^i $\qquad \vdash^i A \ E \ B, A \ E \ C \ \Rightarrow B \ Q \ C$

For i=3 this will be proved below, but for i=2 it is simply false in view of type-inclusion. Now we define

$$B \square C : \Leftrightarrow \ B \sqsubset C \text{ or } C \sqsubset B$$

Below we shall prove that the new symbol covers the relationship between B and C whenever A E B and A E C.

Clearly on the non-1-expressions $\square$ is just Q. We have

$$\vdash^1[x:\alpha]A \sqsubset [x:\beta]B \Leftrightarrow \ \alpha \ Q \ \beta, \ (x \ E \ \alpha \vdash A \square B)$$

Further $\square$ satisfies a strengthening rule, and is substitutive:

$$A \ E \ \alpha, \ x \ E \ \alpha \vdash B \square C \Rightarrow \ B \ [\![A]\!] \square C \ [\![A]\!]$$

3.2.3.3. We also want to show

$$\vdash^1 B \ \square \ C \ \Leftrightarrow \ \text{for some } A, A \sqsubset B \text{ and } A \sqsubset C$$

Proof: $\Rightarrow$ is trivial. So let $B \sqsupset A \sqsubset C$. Then $A \equiv [\bar{z}:\bar{\gamma}][\bar{y}:\bar{\beta}][\bar{x}:\bar{\alpha}]\tau$, $B \equiv [\bar{z}:\bar{\gamma}_1][\bar{y}:\bar{\beta}_1]\tau$, $C \equiv [\bar{z}:\bar{\gamma}_2]\tau$ (or similar with B and C interchanged), with "$\bar{\gamma}$ Q $\bar{\gamma}_1$ Q $\bar{\gamma}_2$", "$\bar{z}$ E $\bar{\gamma} \vdash \bar{\beta}$ Q $\bar{\beta}_1$". So $B \sqsubset C$ (or $C \sqsubset B$).

3.2.4. The canonical type

3.2.4.1. It is possible, for each A with $\vdash^{i+1} A$ to indicate an α_o such that

(1) α_o is a minimal representative - w.r.t. $\sqsubset$ - of the categories of A, i.e.
$A \ E \ \alpha_o$ and: $(A \ E \ \alpha \Rightarrow \ \alpha_o \sqsubset \alpha)$

(2) $FV(\alpha_o) \subset FV(A)$

We call this α_o the cantyp of A (with respect to a context). The definition of cantyp is like the definition of typ given previously (sec. IV.3.2), but slightly modified in order to stay in the correct fragment, as follows:

(i) $\mathsf{cantyp}(x) \equiv \mathsf{typ}(x)$

(ii) $\mathsf{cantyp}(p(\bar{A})) \equiv \mathsf{typ}(p)[\![\bar{A}]\!]$

(iii) $\mathsf{cantyp}([x{:}\alpha]B) \equiv [x{:}\alpha]\mathsf{cantyp}(B)$ - w.r.t. to extended context-

(iv) $\mathsf{cantyp}(\{A\}B) \equiv \{A\}\mathsf{cantyp}(B)$ if degree $(B)=3$

(v) $\mathsf{cantyp}(\{A\}B) \equiv C[\![A]\!]$ if degree $(B)=2$ and $\mathsf{cantyp}(B) \equiv [x{:}\alpha]C$

3.2.4.2. Clearly, $\mathsf{typ}(A) \geq \mathsf{cantyp}(A)$ so property (2) above is immediate.

Now we prove a lemma corresponding to property (1).

Lemma: if LQ^i and $\vdash^{i+1} A \ E \ \alpha$ then $A \ E \ \mathsf{cantyp}(A) \sqsubset \alpha$

Proof: By induction on the length of A. The more interesting cases are

(i) $A \equiv [x{:}\alpha_1]A_1, x \ E \ \alpha_1 \vdash A_1 \ E \ \alpha_2, [x{:}\alpha_1]\alpha_2 \sqsubset \alpha$. By the ind. hyp., $x \ E \ \alpha_1 \vdash A_1 \ E \ \mathsf{cantyp}(A_1) \sqsubset \alpha_2$, so $[x{:}\alpha_1]A \ E \ [x{:}\alpha_1]\mathsf{cantyp}(A_1) \equiv \mathsf{cantyp}(A) \sqsubset [x{:}\alpha_1]\alpha_2 \sqsubset \alpha$, q.e.d.

(ii) $A \equiv \{A_1\}A_2$, $A_1 \ E \ \alpha_1$, $\vdash^2 A_2 \ E \ [x{:}\alpha_1]C$, $C[\![A_1]\!] \sqsubset \alpha$. By the ind. hyp., $A_2 \ E \ \mathsf{cantyp}(A_2) \sqsubset [x{:}\alpha_1]C$ so $\mathsf{cantyp}(A_2) \equiv [x{:}\alpha_1']C'$. Hence $\mathsf{cantyp}(A)$ is indeed defined, $\alpha_1 \ Q \ \alpha_1'$, $x \ E \ \alpha_1 \vdash C' \sqsubset C$, so $\{A_1\}A_2 \ E \ C'[\![A_1]\!] \sqsubset \alpha$, q.e.d.

(iii) $A \equiv \{A_1\}A_2$, $A_1 \ E \ \alpha_1$, $\vdash^3 A_2 \ E \ B \ E \ [x{:}\alpha_1]C$, $\{A_1\}B \ Q \ \alpha$. By the ind. hyp. $A_2 \ E \ \mathsf{cantyp}(A_2) \ Q \ B$. By LQ^i we can use property $(*^i)$ of sec. 3.1.5 and get $\mathsf{cantyp}(A) \ Q \ \{A_1\}B \ Q \ \alpha$, q.e.d.

3.2.4.3. Corollary: (i) $\vdash^2 A \ E \ B$, $A \ E \ C \Rightarrow B \sqsupset C$ (this is, for A of degree 2, the desired property of $\sqsupset$).

(ii) $\vdash^2 [x{:}\alpha]A \ E \ [x{:}\beta]B \Rightarrow \alpha \ Q \ \beta$, $x \ E \ \alpha \vdash A \ E \ B$ (this includes EUD^2)

(iii) SA^2

Proof: (i) LQ^1 is vacuously fulfilled, so $B \sqsupset \mathsf{cantyp}(A) \sqsubset C$, so by 3.2.3.3. $B \sqsupset C$. (ii) and (iii) are immediate.

3.2.5.1. Now that we have introduced cantyp we can use it in the proof of PT. We define the property of *preservation of* cantyp.

PCT^i: $\quad \vdash^i A,\ A \geq A',\ \vdash A' \Rightarrow \mathsf{cantyp}(A)\ \mathsf{Q}\ \mathsf{cantyp}(A')$

Similarly PCT_1^i; PCT is the conjunction of all the PCT^i.

We first prove some lemmas for PCT^2.

3.2.5.2. Lemma (substitution lemma for cantyp): let B^* stand for $B[\![x/A]\!]$. Then $x\ \mathsf{E}\ \alpha,\ \bar{y}\ \mathsf{E}\ \bar{\beta} \vdash^2 C,\ \vdash^3 A\ \mathsf{E}\ \alpha \Rightarrow \mathsf{cantyp}(C)^* \equiv \mathsf{cantyp}(C^*)$ where the cantyp's are taken w.r.t. ($x\ \mathsf{E}\ \alpha,\ \bar{y}\ \mathsf{E}\ \bar{\beta}$) and ($\bar{y}\ \mathsf{E}\ \bar{\beta}^*$) resp.
Proof: Induction on C. Note that $C \sharp x$, because degree(x)=3. Some cases are: (i) $C \equiv [z{:}C_1]C_2$, $\mathsf{cantyp}(C)^* \equiv [z{:}C_1^*]\mathsf{cantyp}(C_2)^*$ (w.r.t. $x\ \mathsf{E}\ \alpha,\ \bar{y}\ \mathsf{E}\ \bar{\beta},\ z\ \mathsf{E}\ C_1$) $\equiv$ (by ind. hyp.) $[z{:}C_1^*]\mathsf{cantyp}(C_2^*)$ (w.r.t. $\bar{y}\ \mathsf{E}\ \bar{\beta}^*,\ z\ \mathsf{E}\ C_1^*$) $\equiv \mathsf{cantyp}(C^*)$, q.e.d.

(ii) $C \equiv \{C_1\}C_2$, $\mathsf{cantyp}(C)^* \equiv D[\![C_1]\!]^* \equiv D^*[\![C_1^*]\!]$ where $\mathsf{cantyp}(C_2) \equiv [z{:}\gamma]D$ and, by ind. hyp., $[z{:}\gamma^*]D^* \equiv \mathsf{cantyp}(C_2^*)$, so $\mathsf{cantyp}(C^*) \equiv D^*[\![C_1^*]\!]$ as well, q.e.d.

3.2.5.3. Corollary: $x\ \mathsf{E}\ \alpha \vdash^2 C,\ \vdash^3 A\ \mathsf{E}\ \alpha \Rightarrow \mathsf{cantyp}(C)[\![A]\!] \equiv \mathsf{cantyp}(C[\![A]\!])$.

3.2.5.4. Corollary (β-PCT_1^2):
$\vdash^2 \{A\}[x{:}B]C \Rightarrow \mathsf{cantyp}(\{A\}[x{:}B]C)\ \mathsf{Q}\ \mathsf{cantyp}(C[\![A]\!])$.
Proof: By SA^2 we have $A\ \mathsf{E}\ B$, so even $\mathsf{cantyp}(\{A\}[x{:}B]C) \equiv \mathsf{cantyp}(C)[\![A]\!] \equiv \mathsf{cantyp}(C[\![A]\!])$.

3.2.5.5. Lemma (η-PCT_1^2):

$\vdash^2 [x{:}\alpha]\{x\}A,\ x \notin \mathrm{FV}(A) \Rightarrow \mathsf{cantyp}([x{:}\alpha]\{x\}A)\ \mathsf{Q}\ \mathsf{cantyp}(A)$

Proof: Let $\mathsf{cantyp}(A) \equiv [y{:}\beta]D$ and let $\vdash^2 [x{:}\alpha]\{x\}A$ be based upon $x\ \mathsf{E}\ \alpha',\ A\ \mathsf{E}\ [y{:}\alpha']D'$. By 3.2.4.2 $[y{:}\beta]D \sqsubset [y{:}\alpha']D'$ and $x \notin \mathrm{FV}([y{:}\beta]D)$, so $\alpha\ \mathsf{Q}\ \alpha'\ \mathsf{Q}\ \beta$ and $\mathsf{cantyp}(A) \equiv [x{:}\beta]D[\![y/x]\!]\ \mathsf{Q}\ [x{:}\alpha]D[\![y/x]\!] \equiv \mathsf{cantyp}([x{:}\alpha]\{x\}A)$.

3.2.5.6. Theorem: PCT_1^2
Proof: let $\vdash^2 A$, $\vdash A'$, $A > A'$. For a main reduction use 3.2.5.4 or 3.2.5.5. For inside reductions use induction on the length of A. Some cases are:

(i) $A \equiv [x{:}A_1]A_2$, $A' \equiv [x{:}A_1']A_2'$, $A_1 > A_1'$, $A_2 > A_2'$. By ind. hyp. $\mathsf{cantyp}([x{:}A_1]A_2)$ Q $\mathsf{cantyp}([x{:}A_1]A_2') \equiv [x{:}A_1]\mathsf{cantyp}(A_2')$ Q $[x{:}A_1']\mathsf{cantyp}(A_2')$, by the substitution property 3.2.5.3.

(ii) $A \equiv \{A_1\}A_2$, $A' \equiv \{A_1'\}A_2'$, $A_1 > A_1'$, $A_2 > A_2'$. Since $\{A_1\}A_2$ is oorrect, A_1 E α_1, A_2 E $\mathsf{cantyp}(A_2) \equiv [x{:}\beta]C \sqsubset [x{:}\alpha_1]D$. So α_1 Q β. Similarly A_1' E α_1', A_2' E $\mathsf{cantyp}(A_2') \equiv [x{:}\beta']C' \sqsubset [x{:}\alpha_1']D'$. So α_1' Q β'. By the ind. hyp. $[x{:}\beta]C$ Q $[x{:}\beta']C'$, so $C[\![A_1]\!]$ Q $C'[\![A_1]\!]$ Q $C'[\![A_1']\!]$, q.e.d.

3.2.5.7. Corollary: (i) PT_1^2, (ii) LQ^2, (iii) UD^2.

3.2.6.1. By LQ^2 we can apply 3.2.4.2 to expressions of degree 3 now. We get: (i) $\vdash^3 A$ E $\alpha \Rightarrow A$ E $\mathsf{cantyp}(A)$ Q α

(ii) UT^3: $\vdash^3 A$ E α, A E $\beta \Rightarrow \alpha \sqsupset \beta$ (i.e. α Q β)

(this is the announced property of $\sqsupset$ for A of degree 3).

(iii) SA^3 (e.g. as in 3.1.6)

Notice that by UT^3 the properties PCT^3 and PT^3 are equivalent.

3.2.6.2. We introduce CLPT^i:

$$\vdash^i A(\mathsf{E}\ \alpha),\ A \geq A' \Rightarrow \vdash^i A'(\mathsf{E}\ \alpha)$$

and similarly CLPT_1^i.

Here follow some lemmas for CLPT_1^3.

3.2.6.3. Lemma (β-CLPT_1^3): $\vdash^3 \{A\}[x{:}B]C$ E $D \Rightarrow C[\![A]\!]$ E D

Proof: Let A E α, $[x{:}B]C$ E F E $[x{:}\alpha]G$, $\{A\}F$ Q D, and let x E $B \vdash C$ E H. $[x{:}B]H$ Q F. By SA^3 we have A E B and by $(*^2)$ $\{A\}[x{:}B]H$ Q $\{A\}F$. By the substitution theorem for correctness $C[\![A]\!]$ E $H[\![A]\!]$ Q D.

3.2.6.4. Lemma (η-CLPT_1^3): $\vdash^3 [x{:}\alpha]\{x\}A$ E B, $x \notin \mathrm{FV}(A) \Rightarrow A$ E B

Proof: $\mathsf{cantyp}([x{:}\alpha]\{x\}A) \equiv [x{:}\alpha]\{x\}\mathsf{cantyp}(A)$ Q $\mathsf{cantyp}(A)$ (by η-reduction), by strengthening $\vdash A$, so by 3.2.6.1 A E B.

3.2.6.5. Now we are ready for CLPT.

Theorem: (CLPT_1): $\vdash A(\mathsf{E}\ \alpha)$, $A > A' \Rightarrow \vdash A'(\mathsf{E}\ \alpha)$

Proof: If $A > A'$ is a main reduction use SA, strengthening, PT^2 and the preceding two lemmas. Otherwise use induction on the length of A.

(i) $A \equiv [x{:}\alpha_1]A_1$, $A' \equiv [x{:}\alpha_1']A_1'$, $\alpha_1 > \alpha_1'$, $A_1 > A_1'$, x E $\alpha_1 \vdash A_1(\mathsf{E}\ \alpha_2)$, $([x{:}\alpha_1]\alpha_2 \sqsubset \alpha)$. By ind. hyp. $\vdash \alpha_1'$ and x E $\alpha_1' \vdash A_1'(\mathsf{E}\ \alpha_2)$.

So $\vdash[x{:}\alpha_1']A_1'$ ($\mathsf{E}\ [x{:}\alpha_1']\alpha_2\ \mathsf{Q}\ [x{:}\alpha_1]\alpha_2 \sqsubset \alpha$) - read this twice, one time with and one time without the symbols in parentheses -.

(ii) $A \equiv \{A_1\}A_2$, $A' \equiv \{A_1'\}A_2'$, $A_1 > A_1'$, $A_2 > A_2'$, $A_1\ \mathsf{E}\ \alpha_1$, $A_2\ \mathsf{E}\ [x{:}\alpha_1]C$, $C[\![A]\!] \sqsubset \alpha$. By ind. hyp. $A_1'\ \mathsf{E}\ \alpha_1$, $A_2'\ \mathsf{E}\ [x{:}\alpha_1]C$. So $A'\ \mathsf{E}\ C[\![A_1']\!]\ \mathsf{Q}\ C[\![A_1]\!]$.

(iii) As in (ii), but $A_2\ \mathsf{E}\ B\ \mathsf{E}\ [x{:}\alpha_1]C$, $\{A_1\}B \sqsubset \alpha$. By ind. hyp. $A_1'\ \mathsf{E}\ \alpha_1$, $A_2'\ \mathsf{E}\ B$, so $A'\ \mathsf{E}\ \{A_1'\}B\ \mathsf{Q}\ \{A_1\}B$.

(iv) $A \equiv p(B_1,\ldots B_k)$, $A' \equiv p(B_1',\ldots,B_k')$, $\bar{B} > \bar{B}'$, $B_1\ \mathsf{E}\ \beta_1$, $B_2\ \mathsf{E}\ \beta_2[\![B_1]\!],\ldots,\ B_k\ \mathsf{E}\ \beta_k[\![B_1,\ldots,B_{k-1}]\!]$, $P[\![\bar{B}]\!] \sqsubset \alpha$, where $\bar{y}\ \mathsf{E}\ \bar{\beta} * p(\bar{y})\ \mathsf{E}\ P$ is a scheme. By ind. hyp. $B_1'\ \mathsf{E}\ \beta_1$, $B_2'\ \mathsf{E}\ \beta_2[\![B_1']\!]\ \mathsf{Q}\ \beta_2[\![B_1']\!],\ldots,\ B_k'\ \mathsf{E}\ \beta_k[\![\bar{B}]\!]\ \mathsf{Q}\ \beta_k[\![\bar{B}']\!]$, so $p(B_1',\ldots,B_k')\ \mathsf{E}\ P[\![B_1',\ldots,B_k']\!]\ \mathsf{Q}\ P[\![\bar{B}]\!]$.

3.2.6.6. Corollary: (i) CLPT, (ii) LQ, (iii) UD.

3.2.6.7. Corollary (Rule V.2', sec. 2.11): $\vdash A$, $\vdash B$, $A \downarrow B \Rightarrow A\ \mathsf{Q}\ B$

3.3. Extension to βηδ-AUT-QE+

3.3.1. Now we consider βηδ-AUT-QE+, i.e. βη-AUT-QE extended with 1-application expressions, with definitional constants and with definitional reduction. The additional rules are

I.3: $\quad A\ \mathsf{E}\ \alpha,\ \vdash^1 B\ \mathsf{Q}\ [x{:}\alpha]C \Rightarrow \vdash^1 \{A\}B$

(vi'): $\quad \bar{A}\ \mathsf{E}\ \bar{\alpha}[\![\bar{A}]\!],\ \bar{x}\ \mathsf{E}\ \bar{\alpha} * d(\bar{x}) := D$ ($*d(\bar{x})\ \mathsf{E}\ E$) is a scheme $\Rightarrow$

$$\vdash d(\bar{A})\ (\mathsf{E}\ E\ [\![\bar{A}]\!])$$

(cf. sec. 3.2.2 and sec. 2.3 respectively).

If we try to repeat the previously given proof, we first come in trouble because not all the compound 1-expressions are abstraction expressions anymore. This makes the proof of UD^1 from sec. 3.2.3 fail-though the property itself remains valid. Furthermore there is the problem with definitional 2-constants and type-inclusion (mentioned in sec. 1.7), which makes LQ^2 fail.

Below we give an indirect proof instead which runs as follows: first we show (secs. 3.3.3 - 3.3.8) that the indicated extension is a so-called *unessential* extension. Then we use this fact to transfer the desired properties from βη-AUT-QE to the new system (sec. 3.3.9). Finally (in sec. 3.3.11) we briefly discuss an even larger system than

AUT-QE+, which we call AUT-QE*.

3.3.2. Some terminology

Consider two systems of correct expressions with typing and equality relation, $(\vdash, E, Q)$ and $(\vdash_+, E_+, Q_+)$ respectively. $(\vdash_+, E_+, Q_+)$ is an *extension* of $(\vdash, E, Q)$ if $\vdash \Rightarrow \vdash_+$, $E \Rightarrow E_+$ and $Q \Rightarrow Q_+$, i.e.: $\mathcal{B} \vdash$ resp. $\mathcal{B}$; $\xi \vdash$ resp. $\mathcal{B}$; $\xi \vdash A\ (E/Q\ B) \Rightarrow$

$$\mathcal{B} \vdash_+ \text{resp. } \mathcal{B};\ \xi \vdash_+ \text{resp. } \mathcal{B};\ \xi \vdash_+ (E_+/Q_+\ B).$$

We further just write $\vdash_+ A\ E/Q\ B$ instead of $\vdash_+ A\ E_+/Q_+\ B$. The "new" system $\vdash_+$ is said to be *conservative* over the "old" system $\vdash$ if all new facts about old objects are old facts, i.e. if

UE0 $\qquad \vdash A,\ \vdash B,\ \vdash_+ A\ E/Q\ B \Rightarrow \vdash A\ E/Q\ B.$

An extension is *unessential* if no "essentially new" objects are formed, i.e. if all new objects are equal to old ones. This means that the new system can be *translated* into the old one by a mapping $^-$, working on expressions, books and contexts, such that

UE1 $\qquad \vdash_+ A \Rightarrow \vdash_+ A\ Q\ A^-$ and $\vdash A \Rightarrow A \equiv A^-$

UE2 $\qquad \mathcal{B} \vdash_+$ resp. $\mathcal{B};\ \xi\vdash_+$ resp. $\mathcal{B};\ \xi\vdash_+ A \Rightarrow$

$\qquad\qquad \mathcal{B}^-\vdash$ resp. $\mathcal{B}^-;\ \xi^-\vdash$ resp. $\mathcal{B}^-;\ \xi^- \vdash A^-$

UE3 $\qquad \mathcal{B};\ \xi\vdash_+ A\ E/Q\ B \Rightarrow \mathcal{B}^-;\ \xi^- \vdash A^-\ E/Q\ B^-$

Clearly unessential extensions are conservative. Property UE3 means that new formulas imply their old counterparts. Unessential extensions also satisfying UE3', the converse of UE3,

UE3' $\qquad \vdash_+ A,\ \vdash_+ B,\ \vdash A^-\ E/Q\ B^- \Rightarrow \vdash_+ A\ E/Q\ B$

are called *definitional* extensions.

In a definitional extension new formulas are equivalent to old ones. All unessential extensions satisfy the Q-part of UE3', but for the E-part we need property LQ for the larger system (at least if the smaller system satisfies LQ). For that matter, if the +-system satisfies LQ, we have

$$\text{UE1, UE2} \Rightarrow \text{UE3'}$$

and: $\qquad \text{UE0, UE1, UE2} \Rightarrow \text{UE3}$

3.3.3. The translation

Of course, we take βη-AUT-QE for our smaller system $\vdash$ and we take βηδ-AUT-QE+ as the extension $\vdash_+$. We are going to prove that $\vdash_+$ is an unessential (but not a definitional) extension.

For an expression A we intend its translation A^- to be the normal form w.r.t. a certain reduction relation $\geq_-$. In order to make A^- well-defined and in view of UE1, UE2 we require

(0) $\geq_-$ normalizes and satisfies CR

(1) $\geq_-$ just affects the new elements of expressions (1-application parts and definitional constants) and removes them

(2) $\geq_-$ is part of the reduction relation of the new system and satisfies CLPT

For contexts $\xi \equiv \bar{x}\ \mathsf{E}\ \bar{\alpha}$ the context ξ^- is simply $\bar{x}\ \mathsf{E}\ \bar{\bar{\alpha}}^-$ (where the meaning of $\bar{\bar{\alpha}}^-$ is clear). Similarly schemes for primitive constants $\xi * p(\bar{x})\ \mathsf{E}\ \beta$ are translated into $\bar{\xi}^- * p(\bar{x})\ \mathsf{E}\ \beta^-$. But schemes for definitional constants have to be omitted in the translation.

Before fixing $\geq_-$ we define *i^j-reduction* $\geq_i^j$, i-reduction of degree j (where i is β, η, δ or a combination of these). This is the reduction relation generated from *elementary i^j-reduction*, defined as follows:

A elementary i^j-reduces to A' if A elementary i-reduces to A' and degree(A)=j. The corresponding one-step reduction is denoted $>_i^j$. Notice that for degree-correct A the degree of A' above is j as well (cf. sec. 2.7).

Now, in view of requirement (1) above, we define $\geq_-$ to be the reduction relation generated from $\geq_\beta^1$ and $\geq_\delta$.

3.3.4. Notice that β^1-reductions cannot be inside reductions. Strong normalization for β^1 is easy to prove even without using normability. From Ch.III we recall δ-SN and δ-CR. As in Ch.II, secs. 6, 7, 8, we can show that β^1-CR holds, and that β^1 commutes with all other reductions (such as β^2, δ, η^2) except η^1.

So $\geq_-$ commutes with all kinds of reduction but η^1, and we have $\geq_-$-SN and $\geq_-$-CR (whence requirement (0) above).

Clearly $\geq_-$-normal forms do not contain defined constants anymore; a simple normability argument shows that $\geq_-$ removes the 1-application

parts as well.

3.3.5. A further property we want $\geq_-$ to satisfy is CLPT. Since δ-CLPT1 follows from the simultaneous substitution theorem (cf. 2.9.4) we just want to know SA^1

$$\vdash^1_+\{A\}[x{:}B]C \Rightarrow \vdash_+ A \mathsf{E} B$$

or, equivalently, UD^1

$$\vdash^1_+[x{:}B]C \mathsf{Q} [x{:}\alpha]D \Rightarrow \vdash_+ \alpha \mathsf{Q} B.$$

Here turn up the problems with 1-expressions, announced in 3.3.1. To overcome these we seemingly modify our system:

(1) we exclude η^1-reduction

(2) we change our 1-application rule into

I.3' $\quad A \mathsf{E} \alpha, \vdash^1 B \ \mathsf{red}_-[x{:}\alpha]C \Rightarrow \vdash^1_+\{A\}B$

where red_- is $\geq_-$ restricted to the correct expressions, i.e. generated by

$$\vdash_+ A, \vdash_+ A', (A \geq^1_\beta A' \text{ or } A >_\delta A') \Rightarrow \vdash_+ A \ \mathsf{red}_- A'.$$

Clearly I.3. $\Rightarrow$ I.3'., so the modification is a restriction. However, after having proved $\geq_-$-CLPT (whence UE1, see sec. 3.3.6), UE2 and UE3 (sec. 3.3.7) for the modified version, we shall be able to show that both I.3 and η^1-equality: $\vdash_+ A$, $A >^1_\eta A'$, $\vdash_+ A' \Rightarrow \vdash_+ A \mathsf{Q} A'$ are derived rules. Hence the two versions of $\vdash_+$ are equivalent, and we have the desired properties for the original +-system.

3.3.6.1. For the modified system the property SA^1 is clear, so we have the theorem ($\geq_-$-CLPT): $\vdash_+ A (\mathsf{E} \alpha)$, $A \geq_- A' \Rightarrow \vdash_+ A' (\mathsf{E} \alpha)$

Proof: Since we know δ-CLPT, and $\geq^1_\beta$ is just $\equiv$ on the non-1-expressions we only need to consider A of degree 1. Use, e.g., a double induction, viz. (1) on $\theta_-(A)$ - i.e. the length of the $\geq_-$-reduction tree of A, (2) on length(A). The only interesting case is when $A \equiv \{A_1\}A_2$, $A_1 \mathsf{E} \alpha$, $A_2 \ \mathsf{red}_-[x{:}\alpha]C$. If $A_1 \geq A_1'$ then $A_1 \geq_\delta A_1'$ so by δ-CLPT $A_1' \mathsf{E} \alpha$. If $A_2 \geq A_2'$ then by the ind. hyp. and by $\geq_-$-CR: $A_2' \ \mathsf{red}_-[x{:}\alpha']C'$, $[x{:}\alpha]C \ \mathsf{red}_- [x{:}A']C'$. So $A_1' \mathsf{E} \alpha'$ and $\vdash_+\{A_1'\}A_2'$. If $A_2 \equiv [x{:}A_3]A_4$ then $A_1 \mathsf{E} A_3$ (this is SA^1) and $\vdash_+ A_4[\![A_1]\!]$.

Since a reduction $A \geq A'$ starts with an inside or with an outside reduction, we are finished by the first ind. hypothesis.

3.3.6.2. Corollary (UE1): $\vdash_+ A \Rightarrow \vdash_+ A \; Q \; A^-$

3.3.7. Theorem (UE2 and UE3): Consider the system without η^1 and with rule I.3'. Then $\mathcal{B}\vdash_+$, resp. $\mathcal{B};\xi\vdash_+$, resp. $\mathcal{B};\xi\vdash_+ A(E/Q\ B) \Rightarrow$

$$\mathcal{B}^-\vdash, \text{ resp. } \mathcal{B}^-;\xi^-\vdash, \text{ resp. } \mathcal{B}^-;\xi^-\vdash A^-(E/Q\ B^-)$$

Proof: By induction on $\vdash_+$, using $\geq_-$-CLPT. The interesting rules are

(i) appl. rule I.3': let $\vdash_+ A \; E \; \alpha$, $\vdash_+ B \; \mathrm{red}_- \; [x{:}\alpha]C$. By ind. hyp. $\vdash A^- \; E \; \alpha^-$. Clearly $B^- \equiv [x{:}\alpha^-]C^-$ and by ind. hyp. $\vdash B^-$, so $x \; E \; \alpha^- \vdash C^-$, so $\vdash (\{A\}B)^- \equiv C^-[\![A^-]\!]$, q.e.d.

(ii) instantiation rule (vi'): let $\mathcal{B}$ contain a scheme $\bar{y} \; E \; \bar{\beta} \star d(\bar{y}) := D$ (possibly followed by $\star d(\bar{y}) \; E \; C$). Let $\mathcal{B}_1$ be the book preceding this scheme. By ind. hyp. $\mathcal{B}_1^-$; $\bar{y} \; E \; \bar{\beta}^- \vdash D^- (E \; C^-)$. Now if $\mathcal{B};\xi \vdash \bar{B} \; E \; \bar{\beta}[\![\bar{B}]\!]$, then by ind. hyp. $\mathcal{B}^-;\xi^- \vdash \bar{B}^- \; E \; (\bar{\beta}[\![\bar{B}]\!])^- \equiv \bar{\beta}^-[\![\bar{B}^-]\!]$, so $\mathcal{B}^-;\xi\vdash (d(\bar{B}))^- \equiv C^-[\![\bar{B}^-]\!](E \; (C[\![B]\!])^- \equiv C^-[\![B^-]\!])$, q.e.d.

(iii) Q-rule: let $\vdash_+ A \; Q \; B$, $\vdash_+ C$, $B > C$. By ind. hyp. $\vdash A^- \; Q \; B^-$, $\vdash C^-$. Since $\geq_-$ commutes with all other reductions, except possibly η^1 which we have forbidden, we find $B^- \geq C^-$ so by CL for $\beta\eta$-AUT-QE $\vdash B^- \; Q \; C^-$ and $\vdash A^- \; Q \; C^-$, q.e.d. The case that $C \geq B$ instead is completely similar.

3.3.8.1. Now we prove that I.3. is a derived rule in the modified system. So assume $\vdash_+ A \; E \; \alpha$, $\vdash_+ B \; Q \; [x{:}\alpha]C$. By 3.3.7 $\vdash B^- \; Q \; [x{:}\alpha^-]C^-$, whence B^- must be $[x{:}\beta]B_1$ with $\vdash \alpha^- \; Q \; \beta$ and $\vdash_+ \alpha \; Q \; \beta$. Further, by 3.3.6.1., $\vdash_+ B \; \mathrm{red}_- \; B^-$ and by I.3' $\vdash_+ \{A\}B$, q.e.d.

3.3.8.2. Similarly, η^1-equality is a derived rule. Let $\vdash_+ A$, $\vdash_+ A'$ $A \geq_\eta^1 A'$. We can assume that degree(A) = 1. By induction on length(A) we prove that $\vdash_+ A \; Q \; A'$. The interesting case is when $A \equiv [x{:}\alpha_1]\{x\}A'$, $x \notin \mathrm{FV}(A')$. As in 3.3.8.1., $x \; E \; \alpha_1 \vdash_+ A' \; \mathrm{red}_- \; [x{:}\alpha_2]A_1$ with $x \notin \mathrm{FV}(\alpha_2)$. By SA^1 $x \; E \; \alpha_1 \vdash_+ \alpha_1 \; Q \; \alpha_2$ and by strengthening $\vdash_+ \alpha_1 \; Q \; \alpha_2$. So $\vdash_+ A \; Q \; [x{:}\alpha_1]A_1 \; Q \; [x{:}\alpha_2]A_1 \; Q \; A'$, q.e.d.

3.3.8.3. Hence the system with I.3 and η^1-equality is equivalent to the system with I.3' and without η^1-equality. So we have SA^1, $\geq_-$-CLPT, UE_1, UE_2 and UE_3 for the original system of $\beta\eta\delta$-AUT-QE+ now.

3.3.9. The proof of CLPT

3.3.9.1. As in 3.2.6.5, we can prove CLPT_1 from outside-CLPT_1, by induction on correctness. Clearly δ-CLPT (and a fortiori δ-outside-CLPT_1) is included in $\geq$-CLPT, so we just need β- and η-outside-CLPT_1.

In the next section we infer PT^3 and SA from our UE-result, which leaves us to prove the β^2- and η^2-case of outside-PT_1 only. These two cases are dealt with in 3.3.9.3.

3.3.9.2. Consider the properties mentioned in 3.1.5. In this section we distinguish the two versions of a property (viz. for the smaller and the larger system) by providing the latter with a + below. It is clear that

$$\mathsf{UT}^i \Rightarrow \mathsf{UT}^i \quad \text{and} \quad \mathsf{UT}^i_+, \mathsf{PT}^i \Rightarrow \mathsf{LQ}^i_+$$

whence UT^3_+, PT^3_+ and LQ^3_+.

The property UD is also preserved in passing to the larger system, and in fact, as in 3.2.3.1,

$$\vdash_+ [x{:}\alpha]A \ \mathsf{Q} \ [x{:}\beta]B \Rightarrow \vdash_+ \alpha \ \mathsf{Q} \ \beta, \ (x \ \mathsf{E} \ \alpha \vdash_+ A \ \mathsf{Q} \ B)$$

By LQ^3_+ we have $(*^3_+)$. SA^1_+ we knew already. Now we show SA^i_+ for $i \neq 1$: let $\vdash^i_+ \{A\}[x{:}B]C$. Since $i \neq 1$, $(\{A\}[x{:}B]C)^- \equiv \{\bar{A}\}[x{:}\bar{B}]\bar{C}$, so by UE_2, $\vdash^i \{\bar{A}\}[x{:}\bar{B}]\bar{C}$ and by SA, $\vdash \bar{A} \ \mathsf{E} \ \bar{B}$. Hence by LQ^3_+ again, we have SA^i_+ for $i \neq 1$ as well.

3.3.9.3. In sec. 3.2.5 we used cantyp in proving β- and η-outside-PT^2_1. The same procedure applies in the +-system, but with typ (defined as in IV.3.2) instead of cantyp now. In particular we have

$$\text{(ii')} \quad \mathsf{typ}(d(\bar{A})) \equiv \mathsf{typ}(d)[\![\bar{A}]\!]$$

for defined constants of degree 2 and 3 now

and $$\text{(iv)} \quad \mathsf{typ}(\{A\}B) \equiv \{A\}\mathsf{typ}(B)$$

for both B of degree 2 and 3.

As in 3.2.4.2 we get

$$\vdash^2_+ A \ \mathsf{E} \ \alpha \Rightarrow \vdash_+ A \ \mathsf{E} \ \mathsf{typ}(A) \sqsubseteq \alpha$$

and,

as in 3.2.5.2.,

$$\vdash_{+}^{3} A \mathsf{E} \alpha,\ (x \mathsf{E} \alpha \vdash^{2} C) \Rightarrow \mathsf{typ}(C[\![A]\!]) \equiv \mathsf{typ}(C)[\![A]\!]$$

So, as in 3.2.5.4 and 3.2.5.5, we get

$$\vdash_{+}^{2} \{A\}[x{:}B]C \Rightarrow \mathsf{typ}(\{A\}[x{:}B]C)\ \mathsf{Q}\ \mathsf{typ}(C[\![A]\!])$$

whence β-outside-$\mathsf{PT}^{2}_{+,1}$, and

$$\vdash_{+}^{2} [x{:}\alpha]\{x\}A,\ x \notin \mathsf{FV}(A) \Rightarrow \mathsf{typ}([x{:}\alpha]\{x\}A)\ \mathsf{Q}\ \mathsf{typ}(A)$$

whence η-outside-$\mathsf{PT}^{2}_{+,1}$.

3.3.10.1. In 3.3.9.2 we have carefully avoided the properties which do not hold in the larger system, in particular LQ^{2} and $(*^{2})$. For a counterexample let $d(x)$ be defined by $x \mathsf{E} \tau * d(x) := [y{:}x]x$, with $\mathsf{typ}(d) \equiv \tau$. If $\alpha \mathsf{E} \tau$, then $d(\alpha)\ \mathsf{Q}\ [y{:}\alpha]\alpha \mathsf{E} [y{:}\alpha]\tau$, but certainly not $d(\alpha) \mathsf{E} [y{:}\alpha]\tau$, so not LQ^{2}. If, furthermore, $A \mathsf{E} \alpha$, then $\vdash \{A\}[y{:}\alpha]\alpha$ but not $\vdash \{A\}d(\alpha)$, whence not $(*^{2})$. Consequently, the +-system is not a definitional extension of the old system.

3.3.10.2. Besides, if we stick to our counterexample, $z \mathsf{E} d(\alpha) \vdash z \mathsf{E} [y{:}\alpha]\alpha$, so $z \mathsf{E} d(\alpha) \vdash \{A\}z \mathsf{E} \alpha$, but not $z \mathsf{E} d(\alpha) \vdash \{A\}d(\alpha)\ (\equiv \mathsf{typ}(\{A\}z))$. This shows that typ applied to 3-expressions can lead us out of the correct expressions (in contrast with the situation in the smaller system), and that not:

$$\vdash^{3} A \Rightarrow A \mathsf{E} \mathsf{typ}(A)$$

3.3.10.3 In the next section we restore $(*)$ and LQ^{2} by a further extension of the language. But first we give a theorem stating some very weak versions of LQ^{2} to hold in βηδ-AUT-QE+ instead of LQ^{2}. Recall the symbol $\square$ from sec. 3.2.3 and the result (sec. 3.2.4.3, 3.2.6.1) for βη-AUT-QE:

$$\vdash A \mathsf{E} B,\ \vdash A \mathsf{E} C \Rightarrow \vdash B \square C.$$

Theorem: Let $\vdash_{+} A \mathsf{E} B$, $\vdash_{+} C \mathsf{E} D$, $\vdash A\ \mathsf{Q}\ C$. Then

$$\vdash_{+} A \mathsf{E} D \quad \text{or} \quad \vdash_{+} C \mathsf{E} B$$

Proof: By UE we get $\vdash A^- \mathsf{E} B^-$, $\vdash C^- \mathsf{E} D^-$, $\vdash A^- \mathsf{Q} C^-$. By LQ for βη-AUT-QE we get $\vdash C^- \mathsf{E} B^-$ so $\vdash B^- \sqsupset D^-$, so $\vdash_+ B \mathsf{Q} B^- \sqsupset D^- \mathsf{Q} D$, i.e. $\vdash_+ B \sqsupset D$, i.e. $B \sqsubset D$ or $D \sqsubset B$, q.e.d.

3.3.11.1. The aforementioned anomalies can partially be removed by properly extending βηδ-AUT-QE+ to a language βηδ-AUT-QE*. In this new system we first replace the application rules by

(1) $B \mathsf{Q} [x{:}\alpha]C$, $A \mathsf{E} \alpha \Rightarrow \vdash \{A\}B$

(2) $B \mathsf{E} C$, $\vdash \{A\}C \Rightarrow \vdash \{A\}B \mathsf{E} \{A\}C$

Rule (1) is simply I.3 without the restriction to degree 1. Rule (2) is III.3.B' (sec. 2.15). So, indeed, AUT-QE* extends AUT-QE+.

3.3.11.2. By this modification we gain the property

$$\vdash {}^3A \Rightarrow \vdash \mathsf{typ}(A), \text{ so it is a proper extension.}$$

Furthermore, by η-reduction we get

$$B \mathsf{E} [x{:}\alpha]C \Rightarrow B \mathsf{Q} [x{:}\alpha]\{x\}B, \text{ which yields property } (*)$$

for the new system.

Our counterexample, however, shows that there are still problems: LQ^2 does not hold, so we do not yet have a definitional extension of AUT-QE. Besides, now the new 2-expressions (e.g. $\{A\}\mathsf{d}(\alpha)$ in the example, which is correct now) do not have a correct typ, and not even an E-formula.

3.3.11.3. The following theorem shows that the difference between AUT-QE+ and AUT-QE* just lies in the particular role of the definitional 2-constants, and that AUT-QE* is an unessential extension of AUT-QE+ (though it is no definitional extension).
Theorem: Let $\vdash_*$ stand for correctness in AUT-QE*, and let A' be the δ^2-normal form of A. Then $\vdash_* A(\mathsf{E}/\mathsf{Q}\ B) \Rightarrow \vdash_+ A'(\mathsf{E}/\mathsf{Q}\ B')$ (so $\vdash A^-(\mathsf{E}/\mathsf{Q}\ B^-)$).
Proof: Induction on $\vdash_*$.

3.3.11.4. A drastic way of combining 2-constants with type-inclusion and still preserve LQ, is to add LQ explicitly to the language definition, or at least something like

$$\vdash {}^2A,\ C \mathsf{E} B,\ A \geq^2_\delta C \Rightarrow A \mathsf{E} B$$

Adding this rule to βηδ-AUT-QE+ produces the smallest definitional extension of AUT-QE which includes βηδ-AUT-QE+, and it gives us AUT-QE* *plus* all the missing E-formulas.

An alternative way of defining this new system (*we still call it* AUT-QE*) is by ignoring the type-assignment part of definitonal 2-schemes, and by defining the typ of a definitional 2-constant to be the typ of its definiens (compare the definition of μ in IV.4.4).

From the latter definition of this new system it will be clear that our desirable properties (except UT^2, of course) can be proved for it by the same methods as used in the closure proof of AUT-QE+.

3.3.12.1. Up till now we have, for definiteness, just compared βη-AUT-QE with βηδ-AUT-QE+ (and βηδ-AUT-QE*), i.e. we made the extension in one step and added the definitional constants and the 1-appl-expressions simultaneously. One can as well, of course, consider intermediate languages like βη-AUT-QE+ and βηδ-AUT-QE.

Then one notices that the problems with (*), LQ^2 and typ are exclusively due to the δ (in particular δ^2) and not to the + in βηδ-AUT-QE+. Thus βη-AUT-QE+ satisfies LQ and (*), and is a neat definitional extension of βη-AUT-QE, whereas βηδ-AUT-QE has all the unpleasant features of βηδ-AUT-QE+. In fact, βηδ-AUT-QE+ is a definitional extension of βηδ-AUT-QE, and βηδ-AUT-QE can only be made into a definitional extension of βη-AUT-QE (call this new system from now on AUT-QE') by adding a rule like in sec. 3.3.11.4.

3.3.12.2. If one takes AUT-68 instead and adds an application rule:

$$A \text{ E } \alpha,\ [x{:}\alpha]C \text{ Q } B \text{ E } \tau \Rightarrow \{A\}B \text{ E } \tau$$

(compare 3.3.11.1, rule (1)) one gets the corresponding +-language. (i.e. smallest value degree = smallest function degree), AUT-68+. These systems are easier to handle than AUT-QE: both AUT-68 and AUT-68+ satisfy UT, LQ and (*), even in the presence of definitional constants, and AUT-68+ is a definitional extension of AUT-68.

Without definitional constants, AUT-68+ is already contained in AUT-QE, but βηδ-AUT-68+ is not contained in βηδ-AUT-QE. It is contained, though, in the system AUT-QE' of 3.3.12.1.

Closure for AUT-68+ can, e.g., be proved by the methods of the next section (see 3.4.5).

3.4. Some easier closure proofs (for simpler languages)

3.4.1. There are various ways of proving closure for simpler languages, such as βη-AUT-68 or βδ-AUT-QE. First, one can take the closure proof of the previous sections and adapt it to the language under consideration. Since η-reduction, type-inclusion and liberal degree specification (in particular for function degree) are responsible for many technical details in the proof, the simpler languages allow some obvious simplifications. E.g. if a language lacks η-reduction we can clearly skip the η-closure part and, besides, we can freely use CR. Or, if a language has more restricted function degrees (AUT-68 vs. AUT-QE, non-+-languages vs. +-languages), we have to push SA, LQ, UD etc. through less degree levels. And, if a language lacks type-inclusion (AUT-68 and Nederpelt's Λ), we simply have PT ⇒ LQ, and do not need to introduce something like cantyp for this purpose.

A second approach is suggested by the fact that our language definition contains some technicalities which are only introduced to make the closure proof (i.e. *this kind* of closure proof, for a complicated language like βη-AUT-QE) possible. In particular, I intend the use of the restricted Q-rule V.2 instead of the more liberal V.2', i.e. the use of the restricted system type I, instead of the liberal system type II (see sec. 1.2.). Recall that after having proved closure for I, I and II can be proved to be equivalent, and that, after all, we are more interested in system II than in system I.

Now it turns out that, for the simpler languages, the modifications in the language definition (and the detour via system I) are superfluous, and that we can give a direct closure proof for a type II language definition.

Such direct closure proofs are presented below for all the regular languages which either lack η-reduction, or have just function degree 3: β(δ)-AUT-68(+), β(δ)-AUT-QE(+) and βη(δ)-AUT-68. A mere sketch is given for βη(δ)-AUT-68+ (for the definition of AUT-68+ see sec. 3.3.12).

3.4.2. So we give these languages by an E-definition with Q-rule V.2': $A\ Q\ B,\ B \downarrow C,\ \vdash C \Rightarrow A\ Q\ C$

which a priori is stronger than V.2 but later turns out to be equivalent. The properties in secs. 2.9, 2.10 such as the *substitution theorem*, *correctness of categories*, and the property: α of domain degree, A of value degree, $x \mathsf{E} \alpha \vdash A \Leftrightarrow \vdash [x:\alpha]A$ simply go through.

As in sec. 3.1., we essentially just need SA for proving closure. So below we confine ourselves to SA and, in connection with this, UD for the various languages. We start with the η-less languages.

3.4.3.1. Theorem: UD for η-less languages
Proof: Let $[x:\alpha]B \mathsf{Q} [x:\alpha]C$. Then by CR, $[x:\alpha]B \downarrow [x:\alpha]C$ so $\alpha \downarrow \beta$ and $B \downarrow C$, whence $\alpha \mathsf{Q} \beta$ and $x \mathsf{E} \alpha \vdash B \mathsf{Q} C$.

3.4.3.2. Corollary: SA^1 for $\beta(\delta)$-AUT-QE+, SA^2 for $\beta(\delta)$-AUT-68+.
Proof: Let $A \mathsf{E} \alpha$, $[x:B]C \mathsf{Q} [x:\alpha]D$. Then $B \mathsf{Q} \alpha$ so $A \mathsf{E} B$.

3.4.3.3. Let $\sqsubset$ be defined as in sec. 2.14. We need a lemma:
$F \sqsubset G$, $G \geq [\bar{x}:\bar{\beta}]D \Rightarrow F \geq [\bar{x}:\bar{\alpha}]C$ with $|\bar{\alpha}| = |\bar{\beta}|$ and $\bar{\alpha} \downarrow \bar{\beta}$ (i.e. $\alpha_1 \downarrow \beta_1$, $\alpha_2 \downarrow \beta_2$, etc.)
Proof: Induction on $\sqsubset$.

3.4.3.4. Corollary: SA^2 for $\beta(\delta)$-AUT-QE(+), SA^3 for $\beta(\delta)$-AUT-68(+)
Proof: Let $A \mathsf{E} \alpha$, $[x:B]C \mathsf{E} [x:\alpha]D$. Then $[x:B]C \mathsf{E} [x:B]F \sqsubset [x:\alpha]D$. So by the previous lemma $B \mathsf{Q} \alpha$ and $A \mathsf{E} B$.

3.4.3.5. Now in order to get SA^3 for β-AUT-QE(+) we need a lemma again. Notice that the proof of this lemma fails when there are definitional constants.
Lemma: $\vdash^2 A \mathsf{E} B$, $B \geq [\bar{x}:\bar{\beta}]D$, $A \geq [\bar{x}:\bar{\alpha}]C$, $|\bar{\alpha}| = |\bar{\beta}| \Rightarrow \bar{\alpha} \downarrow \bar{\beta}$
Proof: Induction on the length of A. The interesting cases are:

(1) $A \equiv [x_1:\alpha_1]A_1$, $A_1 \geq [\bar{x}_2:\bar{\alpha}_2]C$, $x_1 \mathsf{E} \alpha_1 \vdash A_1 \mathsf{E} B_1$, $[x_1:\alpha_1]B_1 \sqsubset B \geq [x_1:\beta_1][\bar{x}_2:\bar{\beta}_2]D$, $|\bar{\alpha}_2| = |\bar{\beta}_2|$. By 3.4.3.3 $\alpha_1 \downarrow \beta_1$ and $B_1 \geq [\bar{x}_2:\bar{\beta}_2']B_2'$ with $\bar{\beta}_2 \downarrow \bar{\beta}_2'$. By the ind. hyp. $\bar{\alpha}_2 \downarrow \bar{\beta}_2'$ so $\bar{\alpha}_2 \downarrow \bar{\beta}_2$ and $\bar{\alpha} \equiv (\alpha_1,\bar{\alpha}_2) \downarrow (\beta_1,\bar{\beta}_2) \equiv \bar{\beta}$, q.e.d.

(2) $A \equiv \{A_1\}A_2$, $A_1 \mathsf{E} \gamma$, $A_2 \mathsf{E} [z:\gamma]B_1$, $B_1[\![A_1]\!] \sqsubset B \geq [\bar{x}:\bar{\beta}]D$.

By 3.4.3.3 again, $B_1[\![A_1]\!] \geq [\bar{x}:\bar{\beta}']D_1$ with $\bar{\beta} \downarrow \bar{\beta}'$. Because B_1 has degree 1 and A_1 has degree 3, $B_1 \geq [\bar{x}:\bar{\beta}_0]D_0$ with $\bar{\beta}_0[\![A_1]\!] \geq \bar{\beta}'$.

Similarly, since A_2 has degree 2, if $\{A_1\}A_2 \geq [\bar{x}:\bar{\alpha}]C$ then $A_2 \geq [x:\gamma'][x:\bar{\alpha}_0]C_0$ *with* $\bar{\alpha}_0[\![A_1]\!] \geq \bar{\alpha}$, $C_0[\![A_1]\!] \geq C$. By the ind. hyp. $\bar{\alpha}_0 \downarrow \bar{\beta}_0$ so $\bar{\alpha} \leq \bar{\alpha}_0[\![A_1]\!] \downarrow \beta_0[\![A_1]\!] \geq \bar{\beta}$ and by CR $\bar{\alpha} \downarrow \bar{\beta}$, q.e.d.

3.4.3.6. Corollary: SA^3 for β-AUT-QE(+)
Proof: Let $A \in \alpha$, $[x:B]C \in D \in [x:\alpha]F$. Then $[x:B]C \in [x:B]G \mathbin{Q} D$ whence $C \geq [x:B']G'$ with $B \geq B'$. By the lemma $B \downarrow \alpha$, so $B \mathbin{Q} \alpha$ and $A \in B$.

3.4.3.7. So we have SA for β(δ)-AUT-68(+) and β-AUT-QE(+). In order to tackle the βδ-case of AUT-QE we first prove δ-CLPT, which give us an unessential extension result. Then we can either extend SA directly, or first extend the lemma 3.4.3.5 to βδ-AUT-QE+ and proceed as before.

3.4.4.1 Now consider βη-AUT-68. We cannot use CR anymore.
Theorem: UD^2 for βη-AUT-68.
Proof: All 2-expressions are of the form $[\bar{x}:\bar{\alpha}]y$ or $[\bar{x}:\bar{\alpha}]p(\bar{C})$. So if $\vdash^2 A \geq [x:\beta]B$, then $A \equiv [x:\alpha]A_1$ with $\alpha \geq \beta$. By ind. on Q we can prove: if $\vdash^2 A \mathbin{Q} [x:\beta]B$ then $A \equiv [x:\alpha]A_1$ with $\alpha \mathbin{Q} \beta$. This gives UD^2.

3.4.4.2. Corollary: SA for βη-AUT-68
Proof: Immediate.

3.4.4.3. The same proof works as well for βηδ-AUT-68, as follows.
Lemma: $\vdash^2 A \geq_\delta [x:\alpha]A_1$, $\vdash^2 B$, $A \downarrow B \Rightarrow B \geq_\delta [x:\beta]B_1$, $\alpha \downarrow \beta$.
Proof: Since $\geq_\delta$ commutes with $\geq$, $[x:\alpha]A_1 \geq [x:\alpha']A_1' \leq_\delta E \leq B$. By δ-advancement (**sec.** II.9.3), $B \geq_\delta C \geq [x:\alpha'']A_1'' \leq_\delta [x:\alpha']A_1'$. Here the reduction $C \geq [x:\alpha'']A_1''$ does not contain δ-reductions so $C \equiv [x:\beta]B_1$ with $\beta \geq \alpha'' \leq \alpha' \leq \alpha$, q.e.d.
3.4.4.4. By the simultaneous substitution theorem we have δ-CLPT again. Then by induction on Q we can prove:

$$\vdash^2 F \mathbin{Q} [x:\beta]B \Rightarrow F \geq_\delta [x:\alpha]A, \ \alpha \mathbin{Q} \beta.$$

This gives us UD^2 whence SA, as before.

3.4.5. It is possible to extend these results (for βη(δ)-AUT-68) to the corresponding +-language βη(δ)-AUT-68+, but it is rather complicated. We can use a mixture of the methods in 3.4.4.3 and 3.4.4.4 and the methods in sec. 3.3. Thus we start with leaving η^2-

reduction out of consideration, and restricting the appl-rule of degree 2 to: $A \mathsf{E} \alpha$, $\vdash^2 B \geq [x{:}\beta]C$, $\alpha \geq \beta \Rightarrow \vdash \{A\}B$.

Later on these two restrictions prove to be immaterial. For the restricted system SA^2 is immediate and β^2-closure is guaranteed. Then we need δ-β^2-advancement and the fact that $\delta\beta^2$-reduction commutes with $\geq$, and get:

$$\vdash^2 F \mathsf{Q} [x{:}\beta]B \Rightarrow F \geq_{\delta\beta^2} [x{:}\alpha]A, \alpha \mathsf{Q} \beta.$$

This yields UD^2, and SA^3 and we are finished.

V.4. The equivalence of the E-definition with the algorithmic definition

4.1. Introduction

4.1.1. Since in the E-definition the correctness of expressions and formulas (relative to a correct book and a correct context) was given by an ordinary inductive definition, the correctness relation is a priori just recursively enumerable and not necessarily recursive i.e. effectively decidable.

In this section V.4, though, we prove the decidability and discuss some related topics. First we give some introductory considerations leading to a sketch of a decision procedure (secs. 4.1.3-4.1.6). The whole verification process is, in principle, reduced to the verification of Q-formulas, for which the decidability follows from the normalization property N and the Church-Rosser property (compare sec. II.5.4). We can use normalization freely because we proved N for a very large system in IV.4.5, but $\beta\eta$-CR we do not know yet. Therefore *we assume throughout* V.4 *property* CR *for the correct expressions, for the proof* of which *we refer to Ch.* VI.

4.1.2. Then (sec. 4.2.2) we present the actual algorithmic definition, to be adapted for the various languages by a suitable choice of a reduction relation, of a *typing function* cantyp and of a domain function dom for the computation of *domains* (sec. 4.2.3., 4.2.4).

The equivalence proof in sec. 4.3 is organized as sketched in sec. 1.2 and 1.6, with the following effects:

(1) the strengthening rule can be skipped from the E-definition
(2) the E-systems are decidable
(3) the algorithmic system satisfies the nice properties of the E-system: closure etc.

The final sections concern the verification of Automath languages *in practice*. This is a matter completely different from the *theoretical* decision procedure discussed before. Particularly some remarks are made on suitable reduction strategies for deciding Q-formulas.

4.1.3. Deciding Q and $\sqsubset$

No matter whether a system has Q-rule V.2 or Q-rule V.2', there holds

$$A\ Q\ B \Leftrightarrow \vdash A,\ \vdash B,\ A \neq B$$

Proof: $\Rightarrow$. By induction on Q, using CR.

$\Leftarrow$. This is precisely rule V.2' so either it holds by definition or it follows from CL. □

So, by N (as in II.5.4), for correct A and B, $A\ Q\ B$ is decidable.

In $\beta(\eta)$-AUT-QE all 1-expressions are of the form $[\bar{x}:\bar{\alpha}]\tau$.

We have $\vdash^1 A \sqsubset \tau \Leftrightarrow \vdash^1 A$

and (sec. 3.2.3.1).

$$\vdash^1 A \sqsubset [x:\beta]B_1 \Leftrightarrow A \equiv [x:\alpha]A_1,\ \alpha\ Q\ \beta \text{ and } x\ E\ \alpha\ \vdash A_1 \sqsubset B_1.$$

So, for correct 1-expressions A and B, $A \sqsubset B$ is decidable (use induction on the length of B). Since on non-1-expressions $\sqsubset$ is just Q, this is true for A and B of other degrees as well.

Let $\vdash$ stand for correctness in $\beta(\eta)$-AUT-QE, $\vdash_+$ for some larger system, like $\beta\eta\delta$-AUT-QE+ or $\beta\eta\delta$-AUT-QE* and let $^-$ denote the $\beta^1\delta$-normal form. By UE (secs. 3.3.2, 3.3.3) we have.

$$\vdash_+ A \sqsubset B \Leftrightarrow \vdash_+ A,\ \vdash_+ B,\ \vdash A^- \sqsubset B^-$$

So, in the larger systems, too, $A \sqsubset B$ is decidable, for correct A and B.

4.1.4. Deciding E-formulas

In principle, E-formulas $A\ E\ B$, for correct A and B are going to be decided by the equivalence

$$A\ E\ B \Leftrightarrow \mathsf{typ}(A) \sqsubset B$$

which reduces the E-formula to a $\sqsubset$-formula.

However, there is some trouble with typ. First, typ can lead us out of the correct expressions of the language we consider. There are two ways to solve this problem: first one can introduce for each language a specific modified type-function cantyp (for: *canonical type*) which does not suffer from this defect. Then we get what we want (as in 3.2.4 for AUT-QE)

$$A \mathsf{E} B \Leftrightarrow \vdash A,\ \vdash B,\ \mathsf{cantyp}(A) \sqsubset B$$

Alternatively, one can use the fact that the new, possibly incorrect expressions created by typ in general are correct in some larger system (e.g. the corresponding +-system). Then one can decide the E-formula in the larger system:

$$A \mathsf{E} B \Leftrightarrow \vdash A,\ \vdash B,\ \vdash_+ \mathsf{typ}(A) \sqsubset B$$

where $\vdash_+$ stands for correctness in the larger system.

If we make sure that $\vdash_+ \mathsf{cantyp}(A)\ \mathsf{Q}\ \mathsf{typ}(A)$ then, by conservativity, the two approaches are clearly equivalent.

A second difficulty with typ occurs exclusively in AUT-QE' and AUT-QE*. These languages have the rule: $\vdash^2 B,\ \vdash C \mathsf{E} D,\ B \geq^2_\delta C \Rightarrow \vdash B \mathsf{E} D$, and for the new category D of B the property $\mathsf{typ}(B) \sqsubset D$ (even if $\mathsf{typ}(B)$ is correct) is not necessarily true anymore.

This problem can be solved by taking a type-function which first eliminates all the δ^2-constants. For a δ^2-constant d we have then $\mathsf{cantyp}(d(\bar{A})) \equiv \mathsf{cantyp}(\delta^2\text{-}\mathsf{nf}(d(\bar{A})))$.

4.1.5. Deciding correctness of expressions

All correct expressions relative to a correct $\mathcal{B}$ and a correct ξ have to be $\mathcal{B}; \xi$-expressions, i.e. the constants have to be in $\mathcal{B}$ and the free variables have to be in ξ. The verification of compound expressions can roughly be described as: verify the subexpressions, plus their possible type- and degree-restrictions. E.g. for abstr-expressions use the equivalence

$\vdash [x{:}\alpha]A \Leftrightarrow \vdash \alpha$, α of domain degree, $x \mathsf{E} \alpha \vdash A$, A of value degree.

For the subexpressions $\bar{B}$ in $c(\bar{B})$ there are type-restrictions prescribed in the scheme of c, viz. if the context of the scheme is $\bar{y} \mathsf{E} \bar{\beta}$ then

$$\vdash c(\bar{B}) \Leftrightarrow \bar{B} \mathsf{E} \bar{\beta}[\![\bar{B}]\!] \quad \text{(i.e. } B_1 \mathsf{E} \beta_1,\ B_2 \mathsf{E} \beta_2[\![B_1]\!] \text{ etc.)}$$

To verify the right hand-side first verify $\vdash B_1$. Since $\vdash \beta_1$ (it occurs in $\mathcal{B}$), we can decide $B_1 \mathsf{E} \beta_1$ as indicated above. Then check $\vdash B_2$. Since $B_1 \mathsf{E} \beta_1$ and $y_1 \mathsf{E} \beta_1 \vdash \beta_2$ we know $\vdash \beta_2[\![B_1]\!]$ so we can tackle the next E-formula etc.

4.1.6. Verification of application expressions

Now we discuss the type-restriction implied in the correctness of $\{A\}B$. We restrict ourselves to AUT-68 and AUT-QE here.

Define α to be *a domain* of B if

(i) $B \mathsf{E} [x{:}\alpha]C$ for some C, or (ii) $B \mathsf{E} C \mathsf{E} [x{:}\alpha]D$ for some C, D.

Then, in view of the formation rules for appl-expressions, we have the equivalence:

$$\vdash \{A\}B \Leftrightarrow \vdash B,\ B \text{ has a domain } \alpha,\ A \mathsf{E} \alpha$$

The arbitrariness w.r.t. the domain can be somewhat reduced by another property of uniqueness of domains, viz.

$$\text{if } \alpha_1 \text{ and } \alpha_2 \text{ are domains of } B \text{ then } \alpha_1 \mathsf{Q} \alpha_2$$

(which will be proved below, 4.2.4.2). This allows us to modify the equivalence:

$$\vdash \{A\}B \iff \vdash B,\ B \text{ has a domain, and } \forall_\alpha\ (B \text{ has a domain } \alpha \Rightarrow A \mathsf{E} \alpha)$$

i.e. we need just one domain to check the type-restriction.

If one fixes a particular procedure for the computation of *some* domain of an expression, one can define a *domain function* dom (specific for each language). E.g. for AUT-68 one might inductively define

$$\delta^2\text{-nf}(\mathsf{cantyp}(B)) \equiv [x{:}\alpha]C \Rightarrow \mathsf{dom}(B) \equiv \alpha.$$

Now define an *extended reduction* relation $\twoheadrightarrow$, as follows:

(i) $A \geq B \Rightarrow A \twoheadrightarrow B$

(ii) $A \twoheadrightarrow \mathsf{typ}(A)$

(iii) $\twoheadrightarrow$ is transitive.

Then, an alternative way to compute a domain of an expression B, is to perform a more or less specified search through the $\twoheadrightarrow$-reduction tree of B until one possibly encounters an abstraction expression, say $[x{:}\alpha]C$; if so, this α is some domain of B. Certain restrictions (specific for each language) have to be imposed upon the search in order to guarantee that not too many expressions get a domain in this way.

Just like property N (at least δ^2-N) is crucial in the definition of dom above, the well-foundedness (i.e. property SN) of $\twoheadrightarrow$ is needed

for the termination of the second procedure. This will indeed be proved below (4.4.11).

As a whole, the situation with the two possible ways of finding a domain can be very well compared with the two ways of deciding a Q-formula: either one can compare normal forms (use N) or one can search for a common reduct in the respective reduction trees (use SN).

4.2. The algorithmic definition

4.2.1. Now we give, guided by the considerations in the preceding sections, the algorithmic definition of correctness. Apart from the compatibility condition of def and typ (see below), the book-and-context part of the definition is as usual (see IV.3) and will be omitted. So we just define the correctness of expressions and formulas (new notations $\vdash_a$, E_a, Q_a and $\sqsubset_a$, with the subscript for "algorithmic") in terms of reduction, dom and cantyp (sec. 4.2). Later we discuss the choice of cantyp and dom for the various regular languages (4.2.3, 4.2.4).

4.2.2.1. Let $\mathcal{B};\xi \vdash_a$. The conventions for omitting $\mathcal{B}$ and ξ in $\mathcal{B};\ \xi \vdash_a A$ are as in V.2.1.. Degrees are indicated as superscripts and defined as usual. The compatibility condition reads: $\mathsf{def}(d)\ \mathsf{E}_a\ \mathsf{typ}(d)$.

4.2.2.2. Formula part of the definition

Let A and B be $\mathcal{B};\ \xi$-expressions (so not necessarily correct). We define: (i) $A\ \mathsf{Q}_a\ B :\Leftrightarrow A \downarrow B$
with the straight forward extension to strings: $\bar{A}\ \mathsf{Q}_a\ \bar{B}$.

(ii) $A \sqsubset_a B$, if degree$(B) = 1 :\Leftrightarrow$
$\beta^1\delta^1\text{-nf}(A) \equiv [\bar{x}:\bar{\alpha}]A_1$, $\beta^1\delta^1\text{-nf}(B) \equiv [x:\beta]\tau$, $\bar{\alpha}\ \mathsf{Q}_a\ \bar{\beta}$.

(iii) $A \sqsubset_a B$, if degree$(B) \neq 1 :\Leftrightarrow A\ \mathsf{Q}_a\ B$

(iv) $A\ \mathsf{E}_a\ B :\Leftrightarrow \mathsf{cantyp}(A) \sqsubset_a B$

with a straightforward extension to strings $A\ \mathsf{E}_a\ B$.

4.2.2.3. Expression part of the definition

(i) $\vdash_a^1 \tau$

(ii) $\vdash_a x : \Leftrightarrow x$ occurs in ξ

(iii) $\vdash_a c(B_1,\ldots,B_m) : \Leftrightarrow \vdash_a B_1, \ldots, \vdash_a B_m$, c occurs in $\mathcal{B}$ and, if the scheme of c has context $\bar{y} \mathsf{E} \bar{\beta}$ then $\bar{B} \mathsf{E}_a \bar{\beta}[\![\bar{B}]\!]$.

(iv) $A \mathsf{E}_a B : \Leftrightarrow \mathsf{cantyp}(A) \sqsubset_a B$
with a straightforward extension to strings $\bar{A} \mathsf{E}_a \bar{B}$

4.2.2.3. Expression part of the definition

(i) $\vdash^1_a \tau$

(ii) $\vdash_a x: \Leftrightarrow x$ occurs in ξ

(iii) $\vdash_a c(B_1,\ldots,B_m) :\Leftrightarrow \vdash_a B_1, \ldots, \vdash_a B_m$, c occurs in $\mathcal{B}$ and, if the scheme of c has context $\bar{y} \mathsf{E} \bar{\beta}$ then $\bar{B} \mathsf{E}_a \beta[\![\bar{B}]\!]$.

(iv) $\xi \vdash_a [x:\alpha]A :\Leftrightarrow \xi \vdash^2_a \alpha$ and $\xi, x \mathsf{E} \alpha \vdash A$ and A has value degree.

(v) $\vdash_a \{A\}B :\Leftrightarrow \vdash^3_a A, \vdash_a B$, B has function degree, $A \mathsf{E}_a \mathsf{dom}(B)$

4.2.3. The choice of cantyp

4.2.3.1. For our purposes (see 4.1.4) we require that, for correct A, $\mathsf{cantyp}(A)$ is as well correct, is a category of A, i.e. $A \mathsf{E} \mathsf{cantyp}(A)$, and is minimal with respect to $\sqsubset$: $A \mathsf{E} B \Rightarrow \mathsf{cantyp}(A) \sqsubset B$.

This leaves us still a lot of freedom for our choice of cantyp: e.g., as long as different definitions of cantyp yield definitionally equal results, they are equally good to us. In some languages typ itself meets the requirements mentioned above, viz. βη-AUT-QE+ and Nederpelt's Λ. In most languages, however, typ causes some problems, e.g. there are correct expressions with incorrect typ; then we choose cantyp to be some suitable modification of typ.

Below we give a survey of the difficulties with typ, and how these can be solved by cantyp.

4.2.3.2. We start with the languages where the trouble with typ is due to mere degree restrictions.

(1) βη-AUT-68: if $\vdash^2 [x:\alpha]\beta$ then its typ is not correct in AUT-68, but is a typical AUT-QE-expression. Then cantyp of this expression has to be τ. Further, $\mathsf{typ}(\{A\}B)$ where $\mathsf{degree}(B) = 3$, is incorrect in AUT-68 but correct in AUT-68+ (so, see 3.3.11.2, in AUT-QE). In cantyp $(\{A\}B)$ we have to remove the applicator $\{A\}$, so we can define $\mathsf{cantyp}(\{A\}B) \equiv C[\![A]\!]$, where $\mathsf{cantyp}(B) \equiv [x:\alpha]C$. This is the same idea as in 3.2.4, but

now for B of degree 3.

(2) $\beta\eta$-AUT-QE and $\beta\eta$-AUT-68+: Application of typ to $\{A\}B$ of degree 2 yields AUT-QE+ expressions. For AUT-68+ cantyp of these expressions has to be τ. For AUT-QE we remove $\{A\}$ from cantyp, by β-reduction as in 3.2.4 (and in (1)).

4.2.3.3 Now we add definitional constants. This gives rise to the interference of δ^2-constants and type-inclusion, discussed before in 3.3.10-3.3.12.

(3) $\beta\eta\delta$-AUT-68: Consider the example of 3.3.10 which is also correct in AUT-68. There occurs an $\{A\}B$ of degree 3 such that typ($\{A\}B$) does not belong to AUT-68 (of course not, as in (1)), does not even belong to AUT-QE and AUT-QE+, but does belong to AUT-68+, AUT-QE' (3.3.12.1) and AUT-QE* (3.3.11). Again, we must remove the applicator in cantyp, but we cannot be certain anymore that cantyp(B) is an abstr-expression. Therefore we define cantyp($\{A\}B$) $\equiv C[\![A]\!]$, where δ^2-nf(cantyp(B)) $\equiv [x{:}\alpha]C$.

(4) $\beta\eta\delta$-AUT-QE(+): The same expression typ($\{A\}B$) of (3) is again incorrect here. Now the applicator is allowed in cantyp, but we need the δ^2-reduction in order to remove the effect of the type-inclusion: cantyp($\{A\}B$) $\equiv \{A\}(\delta^2$-nf(cantyp(B))).

(5) $\beta\eta\delta$-AUT-68+: This language has 2-expressions $\{A\}B$ (see 3.3.11.2), the typ of which is incorrect in all the languages, and even not normable, e.g. $\{A\}\tau$. The cantyp of such $\{A\}B$ must be τ.

(6) $\beta\eta\delta$-QUT-QE' and $\beta\eta\delta$-AUT-QE*: Here we have the same $\{A\}B$ of degree 2 of AUT-68+. Besides, the typ of a degree 2 definitional const-expression (even if typ is correct) need not be a minimal category anymore. Therefore we define cantyp($d(\bar{A})$) :$\equiv$cantyp(δ^2-nf($d(\bar{A})$)). Then for the cantyp of $\{A\}B$ of degree 2 we can simply take $\{A\}$cantyp(B) in AUT-QE*, whereas in AUT-QE' we must take $C[\![A]\!]$ where δ^1-nf(cantyp(B)) $\equiv [x{:}\alpha]C$.

4.2.3.4. Resuming: we have three types of difficulties, viz.

(i) In AUT-68(+) the only 2-expression is τ, so the typ of 2-expressions can be incorrect. Remedy: define cantyp to be τ.

(ii) In non-+-languages (AUT-68, AUT-QE and AUT-QE') the typ of $\{A\}B$ of minimal function degree (say: i) is incorrect. Remedy: create an abstr. expression by taking the $(\beta\delta)^{i-1}$-normal form of cantyp(B) and remove $\{A\}$ by another β^{i-1}-reduction.

(iii) In languages with δ^2-constants and type-inclusion typ produces incorrect appl-2-expressions (AUT-QE(+)) or appl-1-expressions (AUT-QE' and AUT-QE*). Besides, in AUT-QE' and AUT-QE* the typ of a δ^2-const-expression is not necessarily a minimal category. Remedy: remove the δ^2-constants after (AUT-QE(+)) or before (AUT-QE' and AUT-QE*) taking cantyp.

4.2.3.5. In view of the arbitrariness of cantyp (4.2.3.1) we need only three different definitions of cantyp, one for the AUT-68-family, one for the restricted AUT-QE languages AUT-QE and AUT-QE+, and one for the liberal AUT-QE branch (AUT-QE' and AUT-QE*). Since the above list of difficulties is exhaustive, for the rest (e.g. for variables and const-expressions) the definition of cantyp differs only as regards the following clauses:

(1) for AUT-68 and AUT-68+

(i) $\mathrm{degree}(B) = 2 \Rightarrow \mathrm{cantyp}(B) :\equiv \tau$

(ii) $\mathrm{degree}(B) = 3,\ \beta^2\delta^2\text{-nf}(\mathrm{cantyp}(B)) \equiv [x{:}\alpha]C \Rightarrow \mathrm{cantyp}(\{A\}B) :\equiv C[\![A]\!]$

(2) for AUT-QE and AUT-QE+

(i) $\mathrm{degree}(B) = 2,\ \beta^1\delta^1\text{-nf}(\mathrm{cantyp}(B)) \equiv [x{:}\alpha]C \Rightarrow \mathrm{cantyp}(\{A\}B) :\equiv C[\![A]\!]$

(ii) $\mathrm{degree}(B) = 3, \Rightarrow \mathrm{cantyp}(\{A\}B) :\equiv \{A\}(\delta^2\text{-nf}(\mathrm{cantyp}(B)))$

(3) for AUT-QE' and AUT-QE*

(i) $\mathrm{degree}(d) = 2 \Rightarrow \mathrm{cantyp}(d(\bar{A})) :\equiv \mathrm{cantyp}(\delta^2\text{-nf}(d(\bar{A})))$

(ii) $\mathrm{degree}(B) = 2,\ \beta^1\delta^1\text{-nf}(\mathrm{cantyp}(B)) \equiv [x{:}\alpha]C \Rightarrow \mathrm{cantyp}(\{A\}B) :\equiv C[\![A]\!]$

4.2.3.6. That the proposed definitions of cantyp actually satisfy the requirements of 4.2.3.1 can be proved directly for the E-systems using the results (CLPT, LQ, UE etc.) from section 3, but will become clear as well in the course of the equivalence proof, below.

4.2.4. The choice of dom

4.2.4.1. We start with a recapitulation of the appl-rules for the

various languages. First, the appl-rules of AUT-68 ((1) A E α, B E $[x{:}\alpha]C \Rightarrow \vdash\{A\}B$) and of AUT-QE ((2) A E α, B E C E $[x{:}\alpha]D \Rightarrow \vdash\{A\}B$) are simply valid in all the languages (though rule (2) is vacuously so in AUT-68(+)). Then, additionally, rule (3^i) (A E α, $\vdash^i B$ Q $[x{:}\alpha]C \Rightarrow \vdash\{A\}B$); this rule is with i = minimal value degree necessary for defining the +-languages AUT-68+ (i=2), AUT-QE+ and AUT-QE* (i=1). For languages satisfying LQ^i, where i is not the minimal value degree, rule (3^i) is a derived rule. Indeed, for such i is $\vdash^i[x{:}\alpha]C$ E $[x{:}\alpha]D$ so by LQ^i B E $[x{:}\alpha]D$. Hence, rule (3^3) is anyhow valid, rule (3^2) is valid in the AUT-QE languages without δ^2-constants, further in AUT-68+, AUT-QE' and AUT-QE*, and rule (3^1) is valid in AUT-68(+)(vacuously), AUT-QE+ and AUT-QE*. Alternatively formulated, rule (3^i) is always valid but for: rule (3^2) in AUT-68 and AUT-QE(+) with δ^2-constants, and: rule (3^1) in AUT-QE and AUT-QE'.

4.2.4.2. So, for certain languages we must extend the definition of domain from 4.1.6 with the clause: (iii) B Q $[x{:}\alpha]C \Rightarrow \alpha$ is a domain of B. The set of domains of an expression is clearly closed under Q:

$$\alpha_1 \text{ a domain of } B,\ \alpha_1\ Q\ \alpha_2 \Rightarrow \alpha_2 \text{ a domain of } B.$$

The converse of this is the announced uniqueness property, which we prove here for the enlarged notion of domain:

$$\alpha_1 \text{ and } \alpha_2 \text{ both domains of } B \Rightarrow \alpha_1\ Q\ \alpha_2.$$

Proof: From 3.2.3.2, 3.2.4.3, 3.2.5.7 we recall the properties of βη-AUT-QE

$$\vdash^1[x{:}\alpha_1]C\ \square\ [x{:}\alpha_2]D \Rightarrow \alpha_1\ Q\ \alpha_2 \quad \text{(this includes } UD^1\text{)}$$

$$\vdash^2[x{:}\alpha_1]C\ E\ [x{:}\alpha_2]D \Rightarrow \alpha_1\ Q\ \alpha_2 \quad (EUD^2)$$

$$\vdash^2[x{:}\alpha_1]C\ Q\ [x{:}\alpha_2]D \Rightarrow \alpha_1\ Q\ \alpha_2 \quad (UD^2)$$

Now let $\vdash^3[x{:}\alpha_1]C$ E $[x{:}\alpha_2]D$. Then also $\vdash^3[x{:}\alpha_1]C$ E $[x{:}\alpha_1]F$. By UT^2 we get $[x{:}\alpha_2]D$ Q $[x{:}\alpha_1]F$ and by UD^2: α_1 Q α_2. So we have EUD^3 as well. Further $\vdash^3[x{:}\alpha_1]C$ Q $[x{:}\alpha_2]D$. Then also $\vdash^3[x{:}\alpha_1]C$ E $[x{:}\alpha_1]F$ and by LQ^3 $[x{:}\alpha_2]D$ E $[x{:}\alpha_1]F$. So by EUD^3 : α_1 Q α_2. This amounts to UD^3. These results can all be extended to the extensions of βη-AUT-QE by translation (e.g. $\beta^1\delta$-reduction) into βη-AUT-QE, as follows: let $\vdash_+[x{:}\alpha_1]C$ E/□ $[x{:}\alpha_2]D$, where $\vdash_+$ stands for correctness in the larger system. By UE, $\vdash[x{:}\alpha_1^-]C^-$ E/□ $[x{:}\alpha_2^-]D^-$, correct in βη-AUT-QE,

so by one of our (E)UD results: α_1 Q α_1^- Q α_2^- Q α_2. Of course, in AUT-68(+) these (E)UD results are also valid.

Now we treat the various possibilities for α_1 and α_2 to be a domain of B.

(1) $[x{:}\alpha_1]C$ Q B Q $[x{:}\alpha_2]D$. Use UD.

(2) $[x{:}\alpha_1]C$ Q B E $[x{:}\alpha_2]D$. If necessary, translate (e.g. by δ^2-reduction) into a language satisfying LQ: $[x{:}\alpha_1^-]C^-$ Q B^- E $[x{:}\alpha_2^-]D^-$. Then by LQ we get $[x{:}\alpha_1^-]C^-$ E $[x{:}\alpha_2^-]D^-$, and can use EUD.

(3) $[x{:}\alpha_1]C$ Q B E D E $[x{:}\alpha_2]F$. Use LQ: $[x{:}\alpha_1]C$ E D E $[x{:}\alpha_2]F$. But also $[x{:}\alpha_1]C$ E $[x{:}\alpha_1]G$ and by UT^3: $[x{:}\alpha_1]G$ Q D we arrive in case (2) again.

(4) B E $[x{:}\alpha_1]C$, B E $[x{:}\alpha_2]D$. Then $[x{:}\alpha_1]C$ □ $[x{:}\alpha_2]D$ so α_1 Q α_2.

(5) B E $[x{:}\alpha_1]C$, B E D E $[x{:}\alpha_2]F$. By UT^3: $[x{:}\alpha_1]C$ Q D we are again in case (2).

(6) B E C E $[x{:}\alpha_1]D$, B E F E $[x{:}\alpha_2]G$. By UT^3 we get C Q F. Translate into a language satisfying LQ. This gives C^- Q F^- E $[x{:}\alpha_2^-]G^-$ and by LQ C^- E $[x{:}\alpha_2^-]G^-$. It also gives C^- E $[x{:}\alpha_1^-]D^-$, and case (4) applies.

4.2.4.3. It would be nice if the notation of domain of an expression was preserved under Q: B Q C, α a domain of B ⇒ α a domain of C. This is indeed true for languages satisfying LQ, but not for the others, viz. βηδ-AUT-QE and βηδ-AUT-QE+. By CLPT, there holds

$$B \geq C,\ \alpha \text{ a domain of } B \Rightarrow \alpha \text{ a domain of } C$$

i.e. the notion of domain is preserved under ≥. So the converse direction ($C \geq B$, in particular with δ^2-reduction), fails in βηδ-AUT-QE(+).

For all the languages we have

$$B \text{ Q } C,\ \alpha \text{ a domain of } B \Rightarrow \alpha \text{ a domain of } C^-$$

where C^- is the δ^2-normal form of B.

Proof: By the translation $^-$ we arrive in a language satisfying LQ, so from B^- Q C^-, α a domain of B^- we get the desired result.

As a corollary of this, we get

$$B \text{ Q } C,\ \alpha \text{ a domain of } B,\ C \text{ has a domain} \Rightarrow \alpha \text{ domain of } C.$$

4.2.4.4. In view of the above remarks we still have a lot of freedom in defining a domain function dom which picks some expression from the set of domains. Dom is going to be defined in terms of cantyp and, just like cantyp, in terms of δ^2-reduction and $(\beta\delta)^i$-reduction, where i is the minimal value degree. I.e. by application of cantyp and these reductions we arrive at an expression which we call the *domain normal form*, dnf. If the dnf is an abstr-expression then we read off the domain dom from it:

$$\mathsf{dnf}(B) \equiv [x{:}\alpha]C \Rightarrow \mathsf{dom}(B) :\equiv \alpha.$$

Otherwise, dom is simply not defined.

The rules for computing dnf are for the non-+-languages:

(1) AUT-68: $\mathsf{dnf}(B) :\equiv \beta^2\delta^2\text{-}\mathsf{cantyp}(B)$

(2) AUT-QE('): (i) $\mathsf{degree}(B) = 3 \Rightarrow \mathsf{dnf}(B) :\equiv \beta^1\delta^1\text{-}\mathsf{nf}(\mathsf{cantyp}(\delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(B))))$.

(ii) $\mathsf{degree}(B) = 2 \Rightarrow \mathsf{dnf}(B) :\equiv \beta^1\delta^1\text{-}\mathsf{nf}(\mathsf{cantyp}(B))$

The β^2 of AUT-68 and the β^1 of AUT-QE(') were only added in order to cover the corresponding +-languages too. Now, we can deal with the +-languages by simply adding a rule for B of minimal value degree: $\mathsf{degree}(B) = i$, i is minimal value degree $\Rightarrow \mathsf{dnf}(B) :\equiv (\beta\delta)^i\text{-}\mathsf{nf}(B)$. This rule gives us AUT-68+ from AUT-68, AUT-QE+ from AUT-QE and AUT-QE* from AUT-QE'.

4.2.4.5. That $\mathsf{dom}(B)$, as defined above, gives us a domain if B has one, and gives us nothing otherwise, can be proved directly, but will also become clear in the course of the equivalence proof.

4.3. The equivalence proof

4.3.1. As announced before, the equivalence of the algorithmic definition with the E-definition will also prove the superfluity of the strengthening rule. To this end we use, along with the algorithmic definition system III, two distinct versions of the E-definition, system I and system II. Here, system I is the system of sec. 2: it has the strengthening rule and it has Q-rule V.2. System II, however, lacks the strengthening rule and has Q-rule V.2' instead.

By CL for system I, we have: str., V.2 $\Longleftrightarrow$ (str., V.2') $\Rightarrow$ V.2', so

system II is clearly included in system I.

Below we denote correctness in I, II and III respectively by $\vdash$, $\vdash_0$ and $\vdash_a$; hence the inclusion of II in I becomes: $\vdash_0 \Rightarrow \vdash$.

Now the equivalence of the three systems is shown by additionally proving $\vdash_a \Rightarrow \vdash_0$ (sec. 4.3.2) and $\vdash \Rightarrow \vdash_a$ (sec. 4.3.3).

4.3.2. The $\vdash_a \Rightarrow \vdash_0$ -part.

4.3.2.1. We first formulate the theorem, which we want to prove. Theorem: If $\mathcal{B} \vdash_a$ resp. $\mathcal{B};\xi \vdash_a$ resp. $\mathcal{B};\xi \vdash_a^1 A$ resp. $\mathcal{B};\xi \vdash_a^{i+1} A$ then $\mathcal{B} \vdash_0$ resp. $\mathcal{B};\xi \vdash_0^1 A$ resp. $\mathcal{B};\xi \vdash_0^{i+1} A$ E cantyp(A). So the theorem implies that cantyp is well-defined on the non-1-expressions of the algorithmic definition. The proof of the theorem is by induction on $\vdash_a$ and depends of course on dom and cantyp, i.e. on the language we consider. However, large parts of the proof can be done for all or some of the languages together.

4.3.2.2. Some properties

(1) $\vdash_0 A$, $\vdash_0 B$, $A\ Q_a\ B \Rightarrow \vdash_0 A\ Q\ B$

Proof: this is simply rule V.2'.

(2) $\vdash_0 A \Rightarrow \vdash_0 \beta^1\delta^1\text{-nf}(A)\ Q\ A$

Proof: By the simultaneous subst. theorem δ-CLPT holds. Further SA^1 can be proved as in 3.3.6.1-3.3.8.2, or holds vacuously so β^1-CL. By $\beta\delta$-CR and $\beta\delta$-N the $\beta^1\delta^1$-nf is well-defined.

(3) Let $\vdash_0 A$, $\vdash_0 B$, $A \sqsubset_a B$. Then $\vdash_0 A \sqsubset B$

Proof: For $\mathcal{B}$ of degree 1, by (2) $\vdash_0 A\ Q\ \beta^1\delta^1\text{-nf}(A) \equiv [\bar{x}:\bar{\alpha}]A_1 \sqsubset [\bar{x}:\bar{\alpha}]\tau \downarrow [\bar{x}:\bar{\beta}]\tau \equiv \beta^1\delta^1\text{-nf}(B)\ Q\ B$ so $\vdash_0 A \sqsubset B$. If degree$(B) \neq 1$ this is (1) again.

(4) $\vdash_0 A\ E$ cantyp(A), $\vdash_0 B \Rightarrow \vdash_0 A\ E\ B$

Proof: apply (3).

(5) The $\vdash_0$-system satisfies CR

Proof: $\vdash_0 \Rightarrow \vdash$ and we assumed CR for $\vdash$.

(6) Strengthening for Q:

$\xi \vdash_0 A\ Q\ B$, ξ_1 sub ξ, $\xi_1 \vdash_0 A$, $\xi_1 \vdash_0 B \Rightarrow \xi_1 \vdash_0 A\ Q\ B$

Proof: By ind. on Q we get $A \downarrow B$ so $\xi_1 \vdash_0 A\ Q\ B$.

4.3.2.3. Proof of the theorem, part 1

We only need to give the inductionstep for those clauses in the definition of $\vdash_a$ which differ from the corresponding clauses in the definition of $\vdash_0$. We start with the easy cases.

(1) the compatibility condition
let $\xi * d(\bar{x}) := A * d(\bar{x})$ E B be a correct scheme according to the algorithmic definition, i.e. $\xi \vdash_\alpha A$, $\xi \vdash_a B$ and A E$_a$ B. By the ind. hyp. $\xi \vdash_0 A$ E cantyp(A), $\vdash_0 B$, so by (4) above $\xi \vdash_0 A$ E B, q.e.d.

(2) expressions (easy cases)
(i) τ: trivial
(ii) variables: let $\xi \vdash_a$ then by the ind. hyp. $\xi \vdash_0$, so for x in ξ, $\xi \vdash_0 x$ E typ(x) $\equiv$ cantyp(x).
(iii) const-expressions, except δ^2-const-expressions in AUT-QE' and AUT-QE*: let the scheme of c be in B with context $\bar{y}$ E $\bar{\beta}$. Let $\vdash_a B_1, \ldots, \vdash_a B_m$ and $\bar{B}$ E$_a$ $\bar{\beta}[\![\bar{B}]\!]$. By the ind. hyp. $\vdash_0 B_1$ E cantyp(B_1), $\vdash_0 B_2$ E cantyp(B_2) etc. Further $\bar{y}$ E $\bar{\beta} \vdash_a$ so $\bar{y}$ E $\bar{\beta} \vdash_0$ so $\vdash_0 \beta_1$, y_1 E $\beta_1 \vdash_0 \beta_2$ etc. So $\vdash_0 B_1$ E β_1 and by the subst. theorem $\vdash_0 \beta_2[\![B_1]\!]$, so $\vdash_0 B_2$ E $\beta_2[\![B_1]\!]$ etc. up to $\vdash_0 B_m$ E $\beta_m[\![\bar{B}]\!]$. The conclusion is $\vdash_0 c(\bar{B})$(E typ(c)$[\![\bar{B}]\!]$ $\equiv$ cantyp($c(\bar{B})$)).
(iv) abstr-expressions: let $\xi \vdash_a^2 \alpha$ and ξ, x E $\alpha \vdash_a A$, A of value degree. By the ind. hyp. $\xi \vdash_0^2 \alpha$ and ξ, x E $\alpha \vdash_0 A$(E cantyp(A)). For A of degree 2 in AUT-68(+) this is ξ, x E $\alpha \vdash_0 A$ E τ which yields $\xi \vdash_0 [x{:}\alpha]A$ E $\tau \equiv$ cantyp($[x{:}\alpha]A$). Otherwise, we get $\xi \vdash_0 [x{:}\alpha]A$ (E $[x{:}\alpha]$cantyp(A) ◘ cantyp($[x{:}\alpha]A$)).

4.3.2.4. Some more properties

Before discussing the remaining clauses we prove some more properties of $\vdash_0$. First something about $\sqsubset$. Of course, the $\beta^1\delta^1$-nf's of 1-expressions are of the form $[\bar{x}{:}\bar{\alpha}]\tau$. As in 3.3.6-3.3.8 (leave η^1 out of consideration, restrict the appl-1-rule) we can prove, even without using CR

$$\vdash_0^1 A \; Q \; B \Rightarrow \beta^1\delta^1\text{-nf}(A) \equiv [\bar{x}{:}\bar{\alpha}]\tau,\ \beta^1\delta^1\text{-nf}(B) \equiv [\bar{x}{:}\bar{\beta}]\tau,\ \vdash_0 \bar{\alpha} \; Q \; \bar{\beta},$$

and, by induction on $\sqsubset$,

$\vdash_0^1 A \sqsubset B \Rightarrow \beta^1\delta^1\text{-nf}(A) \equiv [\bar{x}:\bar{\alpha}][\bar{y}:\bar{\gamma}]\tau$, $\beta^1\delta^1\text{-nf}(B) \equiv [\bar{x}:\bar{\beta}]\tau$, $\vdash_0 \bar{\alpha}\ Q\ \bar{\beta}$.
So we get: $\vdash_0^1 A \sqsubset [x:\beta]B_1 \Rightarrow \beta^1\delta^1\text{-nf}(A) \equiv [x:\alpha]A_1$, $\vdash_0 \alpha\ Q\ \beta$, $x\ E\ \alpha\ \vdash A_1 \sqsubset B_1$
Now we prove a lemma: $\vdash_0^2 A\ (E\ B) \Rightarrow \vdash_0^2 A\ E\ \mathsf{cantyp}(A)\ (\sqsubset B)$.
Proof: E.g. in AUT-68(+) there is nothing to prove. Anyhow, the cases $A \equiv \tau$, A a variable or A an easy const-expression (i.e. not a δ^2-const-expression in AUT-QE' or AUT-QE*) are immediate. For the rest we proceed by induction on (1) the length of δ^2-reduction tree of A, (2) the length of A.

Abstraction expressions are easy. If A is a δ^2-const-expression in AUT-QE' or AUT-QE*, by δ-CLPT and the first ind. hyp. $\vdash_0^2 \delta^2\text{-nf}(A)\ E\ \mathsf{cantyp}(\delta^2\text{-nf}(A)) \equiv \mathsf{cantyp}(A)\ (\sqsubset B)$. Then by the extra type modification rule of these languages we get $\vdash_0^2 A\ E\ \mathsf{cantyp}(A)\ (\sqsubset B)$, q.e.d. Now let $A \equiv \{A_1\}A_2$. We have $\vdash_0 A_1\ E\ \alpha$, $_0A_2\ E\ \mathsf{cantyp}(A_2) \sqsubset [x:\alpha]C$. So $\beta^1\delta^1\text{-nf}(\mathsf{cantyp}(A_2)) \equiv [x:\alpha_1]C_1$ with $\alpha_1\ Q\ \alpha$, $x\ E\ \alpha_1\ \vdash C_1 \sqsubset C$. We want $\vdash_0 A\ E\ \mathsf{cantyp}(A) \equiv C_1[\![A_1]\!]\ (\sqsubset B)$. If the formula $A\ E\ B$ in the assumption comes directly from $C[\![A_1]\!] \sqsubset B$ we get $C_1[\![A_1]\!] \sqsubset C[\![A_1]\!] \sqsubset B$ q.e.d.. Otherwise $A \geq_\delta^2 D$, $\vdash_0 D\ E\ B$ (i.e. the extra rule of AUT-QE' and AUT-QE* has been used). This $D \equiv \{D_1\}D_2$ with $A_1 \geq_\delta^2 D_1$, $A_2 \geq_\delta^2 D_2$, so $\vdash_0 D_1\ E\ \alpha$, and $\vdash_0 D_2\ E\ \mathsf{cantyp}(D_2)\ Q\ \beta^1\delta^1\text{-nf}(\mathsf{cantyp}(D_2)) \equiv [x:\alpha_2]C_2 \sqsubset [x:\alpha_1]C_1$ (apply one of the ind. hypotheses to D_2), and by the first ind. hyp. $\vdash_0 D\ E\ \mathsf{cantyp}(D) \equiv C_2[\![D_1]\!]\ Q\ C_2[\![A_1]\!] \sqsubset C_1[\![A_1]\!]$. So, by the type mod. rule, $\vdash_0 A\ E\ C_1[\![A_1]\!]$, q.e.d.

4.3.2.5. Proof of the theorem, part 2

Now we prove the induction step for the two remaining cases.

(1) δ^2-const-expressions in AUT-QE' or AUT-QE*
As in 4.3.2.3.(iii) we can get $\vdash_0^2 d(\bar{B})$ from $\vdash_a d(\bar{B})$. Then by the lemma $\vdash_0 d(\bar{B})\ E\ \mathsf{cantyp}(d(\bar{B}))$.

(2) appl-expressions
Let $\vdash_a^3 A$, $\vdash_a B$, B of function degree, $A\ E_a\ \mathsf{dom}(B)$. By the ind. hyp. $\vdash_0^3 A\ E\ \mathsf{cantyp}(A) \downarrow \mathsf{dom}(B)$, $\vdash B_0(E\ \mathsf{cantyp}(B))$. For the computation of cantyp and dom in the various languages see 4.2.3.5 and 4.2.4.4 respectively.
(i) AUT-68(+), $\vdash_0^3 B$: $\beta^2\delta^2\text{-nf}(\mathsf{cantyp}(B)) \equiv [x:\alpha]C$, $\mathsf{dom}(B) \equiv \alpha$.
By δ-CLPT $\vdash_0 B\ E\ [x:\alpha]C$ and $\vdash_0 \alpha$, so $\vdash_0 A\ E\ \alpha$ and

$\vdash_0 \{A\}B \ \mathsf{E}\ C[\![A]\!] \equiv \mathsf{cantyp}(\{A\}B)$

(ii) AUT-68+, $\vdash_0^2 \& \beta^2\delta^2\text{-}\mathsf{nf}(B) \equiv [x{:}\alpha]C$. We have SA^2 (see e.g. 3.4.5) so β^2-CL so $\vdash_0 B\ \mathsf{Q}\ [x{:}\alpha]C$ and $\vdash_0 \{A\}B\ \mathsf{E}\ \tau \equiv \mathsf{cantyp}(\{A\}B)$

(iii) AUT-QE(+), $\vdash_0^3 B$: $\beta^1\delta^1\text{-}\mathsf{nf}(\mathsf{cantyp}(\delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(B)))) \equiv [x{:}\alpha]C$, $\mathsf{dom}(B) \equiv \alpha$. By δ-CL and the lemma in 4.3.2.4 $\vdash_0 B\ \mathsf{E}\ \delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(B))\ \mathsf{E}\ [x{:}\alpha]C$ so $\vdash_0 \{A\}B\ \mathsf{E}\ \{A\}(\delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(B))) \equiv \mathsf{cantyp}(\{A\}B)$.

(iv) AUT-QE' and AUT-QE*, $\vdash_0^3 B$: As (iii), but from $\vdash_0 \delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(B))\ \mathsf{E}\ [x{:}\alpha]C$ we infer now $\vdash_0 \mathsf{cantyp}(B)\ \mathsf{E}\ [x{:}\alpha]C$ so $\vdash_0 \{A\}B\ \mathsf{E}\ \{A\}\mathsf{cantyp}(B) \equiv \mathsf{cantyp}(\{A\}B)$

(v) AUT-QE, $\vdash_0^2 B$: Like (i) but decrease the degrees by 1

(vi) AUT-QE+ and AUT-QE*, $\vdash_0^1 B$: like (ii), but decrease the degrees by 1.

This finishes the proof of the theorem in 4.3.2.1.

4.3.3. The $\vdash \Rightarrow \vdash_a$-part

4.3.3.1. We formulate our theorem.

Theorem: If $\mathcal{B} \vdash$ resp. $\mathcal{B};\xi \vdash$ resp. $\mathcal{B};\xi \vdash A$ then $\mathcal{B} \vdash_a$ resp. $\mathcal{B};\xi \vdash_a$ resp. $\mathcal{B};\xi \vdash_a A$. Further, if $\mathcal{B};\xi \vdash A\ \mathsf{E}\ B$ then $A\ \mathsf{E}_a\ B$.

The proof will be by induction on $\vdash$. We just discuss AUT-QE, because with AUT-68 everything is completely similar or somewhat easier.

4.3.3.2. First, we need some properties

(1) Strengthening holds in the $\vdash_a$-system
Proof: notice that the definition of cantyp only refers to the relevant parts of the context, i.e. to assumptions concerning actually occurring free variables, and that the other notions in the definition of correctness do not refer to the context at all. Hence, strengthening can be proved by a simple induction on $\vdash_a$.

(2) on PCT^2 (preservation of cantyp): In 3.2.5, we proved $\beta\eta$-outside-PCT_1^2 for $\beta\eta$-AUT-QE. However δ-outside-PCT_1^2 is wrong, so for AUT-

QE(+) with δ^2-constants we can only get *restricted* PCT^2:

if $\vdash^2 A$, $A \geq B$ *not using* δ^2*-reduction* then $\mathsf{cantyp}(A)\ \mathsf{Q}\ \mathsf{cantyp}(B)$

In order to prove this, start with $\vdash^2 A\ \mathsf{E}\ \alpha \Rightarrow \vdash A\ \mathsf{E}\ \mathsf{cantyp}(A) \sqsubset \alpha$ (e.g. as in 4.3.2.4). Then, as in 3.2.5, one can prove:
$\vdash^2 A$, $A \geq B$ not by δ^2-reduction $\Rightarrow \mathsf{cantyp}(A)\ \mathsf{Q}\ \mathsf{cantyp}(B)$.
Restricted PCT^2 gives us restricted LQ^2 for AUT-QE(+):

if $\vdash^2 A$, $B\ \mathsf{E}\ C$, $A\ \mathsf{Q}\ B$ without using δ^2-reduction then $A\ \mathsf{E}\ C$

(3) However, in AUT-QE' and AUT-QE*, full PCT^2 is still valid and hence LQ^2 holds (this was already implicitly claimed in 3.3.11.4).

Proof: In AUT-QE' and AUT-QE* we have

$$\delta^2\text{-}\mathsf{nf}(\mathsf{cantyp}(A)) \leq^2_\delta \mathsf{cantyp}(\delta^2\text{-}\mathsf{nf}(A))$$

So, let $A \geq B$. Then $\delta^2\text{-}\mathsf{nf}(A) \geq \delta^2\text{-}\mathsf{nf}(B)$ without using δ^2-reduction, so by restricted PCT^2 we have $\mathsf{cantyp}(\delta^2\text{-}\mathsf{nf}(B))$.

(4) By CR we have $\vdash A\ \mathsf{Q}\ B \Rightarrow A\ \mathsf{Q}_a\ B$. As in 4.3.2.4 we have
$\vdash^1 A \quad B \Rightarrow \beta^1\delta^1\text{-}\mathsf{nf}(A) \equiv [\bar{x}{:}\bar{\alpha}][\bar{y}{:}\bar{\gamma}]\tau$, $\beta^1\delta^1\text{-}\mathsf{nf}(B) \equiv [\bar{x}{:}\bar{\beta}]\tau$, $\vdash \bar{\alpha}\ \mathsf{Q}\ \bar{\beta}$

So $\qquad \vdash A \sqsubset B \Rightarrow A \sqsubset_a B$

4.3.3.3. Proof of the theorem

Note that the $\vdash A\ \mathsf{E}\ B \Rightarrow A\ \mathsf{E}_a\ B$ part of the theorem, for A of degree 2 follows from $\vdash^2 A\ \mathsf{E}\ B \Rightarrow \vdash A\ \mathsf{E}\ \mathsf{cantyp}(A) \sqsubset B$ (in 4.3.3.2(2) and 4.3.3.2.(4)). The proof is by induction on $\vdash$. We first discuss some of the clauses for the formation of expressions:

(i) abstr-expressions: let $\vdash^2 \alpha$, $x\ \mathsf{E}\ \alpha \vdash A_1(\mathsf{E}\ B_1)$. By the ind. hyp. $\vdash^2_a \alpha$, $x\ \mathsf{E}\ \alpha \vdash_a A_1$, $(A_1\ \mathsf{E}_a\ B_1$, i.e. $\mathsf{cantyp}(A_1) \sqsubset_a B_1)$, so $\vdash_a [x{:}\alpha]A_1$, $(\mathsf{cantyp}([x{:}\alpha]A_1) \equiv [x{:}\alpha]\mathsf{cantyp}(A_1) \sqsubset [x{:}\alpha]B_1$, so $[x{:}\alpha]A_1\ \mathsf{E}_a\ [x{:}\alpha]B_1)$, q.e.d.

(ii) const-expressions: let $\bar{y}\ \mathsf{E}\ \bar{\beta}$ be the context of the scheme of c, $\vdash \bar{B}\ \mathsf{E}\ \bar{\beta}[\![\bar{B}]\!]$. By the ind. hyp. $\vdash_a \bar{B}$, $\bar{B}\ \mathsf{E}_a\ \bar{\beta}[\![\bar{B}]\!]$, so $\vdash_a c(\bar{B})$. If c is not a δ^2-constant in AUT-QE' or AUT-QE* then $\mathsf{cantyp}(c(\bar{B})) \equiv \mathsf{typ}(c)[\![\bar{B}]\!]$ so certainly $\mathsf{cantyp}(c(\bar{B})) \sqsubset_a \mathsf{typ}(c)[\![\bar{B}]\!]$, q.e.d. Otherwise use the remark above.

(iii) 2-appl-expressions: let $\vdash^3 A\ \mathsf{E}\ \alpha$, $\vdash B\ \mathsf{E}\ [x{:}\alpha]C$. By ind. hyp.

$\vdash^3 A$, $\vdash_a B$, cantyp$(A) \downarrow \alpha$, cantyp$(B) \sqsubset_a [x:\alpha]C$.
So $\beta^1\delta^1$-nf(cantyp$(B)) \equiv [x:\alpha']C'$, dom$(B) \equiv \alpha' \downarrow \alpha$. By CR, cantyp$(A) \downarrow$ dom(B) so $\vdash_a \{A\}B$. Further, by the remark above, $\{A\}B$ E_a $C[\![A]\!]$, q.e.d.

(iv) 3-appl-expressions: let $\vdash^3 A$ E α, $\vdash C$ E $[x:\alpha]D$. By the ind. hyp. $\vdash^3_a A$, cantyp$(A) \downarrow \alpha$, $\vdash_a B$, cantyp$(B) \downarrow C$. By δ^2-CLPT, $\vdash \delta^2$-nf(C) E $[x:\alpha]D$. By the $\vdash_a \Rightarrow \vdash_0$-part, $\vdash_0 B$ E cantyp(B) so $\vdash B$ E cantyp(B), so $\vdash$cantyp(B) so $\vdash \delta^2$-nf(cantyp$(B))$. Further δ^2-nf(cantyp$(B)) \downarrow \delta^2$-nf(C) without using δ^2-reduction, so by restricted LQ, $\vdash \delta^2$-nf(cantyp$(B))$ E $[x:\alpha]D$ and cantyp $(\delta^2$-nf(cantyp$(B)))$ $\sqsubset_a [x:\alpha]D$. I.e. $\beta^1\delta^1$-nf(cantyp$(\delta^2$-nf(cantyp$(B))))\equiv [x:\alpha']D'$, $\alpha \downarrow \alpha' \equiv$ dom(B). Hence $\vdash_a \{A\}B$. Further $\{A\}$cantyp$(B) \downarrow \{A\}C$ and $\{A\}(\delta^2$-nf(cantyp$(B)) \downarrow \{A\}C$ so anyhow cantyp$(\{A\}B) \downarrow \{A\}C$, q.e.d. Finally we discuss the type modification rules and the strengthening rule.

(v) Type modification: let $\vdash A$ E B, $B \sqsubset C$. By the ind. hyp. $\vdash_a A$, A E_a B, i.e. cantyp(A) $\sqsubset_a B$ and by 4.3.3.2.(4) $B \sqsubset_a C$. Use CR to get A E_a C q.e.d.

(vi) Strengthening: Use 4.3.3.2.(1).

This finishes the proof of the theorem $\vdash \Rightarrow \vdash_a$ and the proof of the equivalence of the three systems $\vdash$, $\vdash_0$, $\vdash_a$. So we do not distinguish between $\vdash$, $\vdash_0$ and $\vdash_a$ any more and have

$$\vdash A(\mathsf{E}\ \alpha) \Rightarrow \vdash A(\mathsf{E}\ \text{cantyp}(A) \sqsubset \alpha)$$

and $$\vdash \{A\}B \Rightarrow \text{cantyp}(A) \downarrow \text{dom}(B).$$

4.4. The actual verification

4.4.1. Before discussing the *actual* verification we make some concluding remarks on the *formal* decidability of the Automath languages. First, on the *well-definedness* of the decision algorithm suggested by the definition of $\vdash_a$ in sec. 4.2, in particular the well-definedness of cantyp and dom. Cantyp and dom are partial functions, so by well-definedness we understand: (1) it is decidable whether an expression *has a* cantyp (or a dom) (2) *if* it has one, this is effectively computable. All this is already implicitly included in the equivalence proof. E.g. the $\vdash_a \Rightarrow \vdash_0$-part states that cantyp on the correct non-

1-expressions delivers a correct expression again. In the course of the decision process cantyp and dom are required of correct expressions only. E.g. before settling cantyp(A) Q B (in the verification of A E F) we first check $\vdash A$, and before settling A E dom(B) (in the verification of $\{A\}B$) we first check $\vdash B$. The definitions of cantyp and dom just computation of degrees, and computation of $\beta^i\delta$-normal forms where i is the minimal value degree. Notice that β^i-N in this case, and in fact for all i < 3, can even be proved without using normability.

4.4.2. Our second remark concerns the normability. Below we make sure the normability result of sec. IV.4.4., as we claimed already several times, actually covers the regular languages, viz. by proving that the system of sec. IV.4.5 contains our most liberal language AUT-QE*. Let us abbreviate the system of sec. IV.4.5 by system IV. Theorem: System IV contains AUT-QE*.
Proof: This system avoids Q-formulas as indicated in 2.12. For the rest it is like our system $\vdash_0$, with type-modification rule V.2' (sec. 2.11) and without strengthening, but of course with much weaker degree restrictions. The expression formation rules are the familiar rules of AUT-68 and AUT-QE, except perhaps for the appl-rules which are most similar to the rules in 3.3.11 for the first version of AUT-QE*. We only consider the 1-appl-expressions. Let (in AUT-QE*) A E α, $\vdash^1 B$ Q $[x{:}\alpha]C$. By $\beta^1\delta$-reduction we get $B \geq [x{:}\alpha']C'$ which α Q α'. The substitution theorem and SA^1 (and hence $\beta^1\delta$-CL) are as usual valid in system IV, so using induction on AUT-QE*-correctness we get (in system IV) A E α', $\vdash B \geq [x{:}\alpha']C'$ so $\vdash \{A\}B$, q.e.d.

4.4.3. From our axiomatic introduction in sec. II.1.3 the actual nature of expressions does not become very clear, viz. that they are just some well-structured *symbol-strings*. In view of this fact, a verification process for the correctness of expressions must be able to perform the following task: given a correct book and a correct context (mere symbolstrings as well), each symbol-string must, in a finite amount of time, either be recognized as a correct expression (relative to book and context) or be rejected.

The verification of such a string can be analyzed in several stages, e.g.: (1) bracket structure has to be correct, (2) the free

variables have to occur in the context and the constants have to occur in the book (after this stage the constants in the string can be assigned an arity, variables and constants get a degree and possibly a typ and a def), (3) the arity of each constant has to fit the arity of the argument string going with it (only after this stage we can speak of expressions in the sense of sec. II.1), (4) degree restrictions (and possibly norm restrictions) must be satisfied, (5) the type restrictions have to be fulfilled (i.e. of the argument A in $\{A\}B$ and of the argument string $\bar{C}$ in $c(\bar{C})$.

Here it is just stage (1) which represents the context-free part of the verification. The stages (2)-(4) are literally context-dependent, but still trivially recursive. After passing stage (3) an expression is pretyped. From our point of view stage (5) is the interesting part of the verification.

The actually running verification program for Automath languages at Eindhoven University has indeed been organized along this lines (see Zandleven [75], Jutting [37]). There is a first pass with a *"syntax-checker"* covering stages (1) and (2). This pass is optional since there is a next pass with a "translator" covering stages (1)-(4) (but without checking norm-restrictions). And finally there is the "processor", operating on the result of the translator, which covers stage (5).

4.4.4. First we discuss the verification of definitional equalities $A \downarrow B$. As in the case of δ-equality (sec. III.6.2) we do not want to compute normal forms but rather design a *strategy* which after a few reduction steps in A or B either results in common reduct of A and B (if this exists), or enables one to conclude that it does *not* exist.

As explained in sec. III.6.3, when confronted with certain A and B during the decision process, we have to answer the following questions: (1) shall we do an outside reduction, (2) if so, on which of the expressions? The form (or: *shape*) of A and B (i.e. whether they are abstr-, or appl-expressions etc.) plays a crucial role here. E.g. if A and B are both in *immune form* (see II.4.9) then there is no choice: there is simply no outside reduction possible. So either we can immediately decide our definitional equality (if A and B are of different shape, or if A and B are atomic), or we have to *split up*

(or: *decompose*) the equality into the equalities of the corresponding subexpressions of A and B. But if A and B have different form, not both immune, then an outside reduction is required.

The basic construction aim for a decision strategy is of course to minimize in most of the cases the total number of reduction steps required for a conclusion: A is equal to B or not. There is of course uncertainty about what happens in most of the cases, but the intuitive (and possibly questionable) ideas on this subject, underlying the algorithm in the next sections, can be summarized as follows: generally, the definitional equalities arising in the course of the verification and offered to the decision process, *are true*, and a common reduct can be reached in *relatively few* steps.

4.4.5. We define new, restricted relations $>_h$, $\geq_h$ (h for *head reduction*) and $>_h^-$, $\geq_h^-$ which precisely cover: (1) outside reduction steps, (2) the reduction steps needed in order to make new outside steps possible. The relations are given by a simultaneous inductive definition:

(i) $B \geq_h^- [x{:}\alpha]C \Rightarrow \{A\}B >_h^- C[\![A]\!]$

(ii) $d(\bar{C}) >_h^- \mathrm{def}(d)[\![\bar{C}]\!]$

(iii) $A \geq_h \{B\}D$, $B \geq_h x$, $D \geq C$, $x \notin \mathrm{Fv}(C) \Rightarrow [x{:}\alpha]A >_h C$

(iv) $A >_h^- B \Rightarrow A >_h B$

(v) $\geq_h^-$ (resp. $\geq_h$) is the reflexive and transitive closure of $>_h^-$ (resp. $>_h$)

I.e. $>_h^-$ and $\geq_h^-$ are just η-less versions of $>_h$ and $\geq_h$. Clearly $A \geq_h B \Rightarrow A \geq B$, and if $A >_h^- B$ (or $A >_h B$) then B is a first main reduct (see sec. II.4.9) of A.

Remark: This reduction does correspond to the head reduction common in the literature [4] , i.e. to the "first half of" the so-called *normal* reduction [25]. A reduction $A \geq_h^- B$ consists of mere *simple head contractions*, i.e. $\{A_1\}\ldots\{A_k\}B > \{A_1\}\ldots\{A_k\}C$ where $B > C$ is an elementary $\beta\delta$-reduction, and even only such of these that their reduct eventually becomes a new simple head redex.

The unrestricted reduction $D \geq C$ in clause (iii) is put there on purpose: it is of course possible that *internal contractions* are

needed in order to remove free variables from an expression.

The main property of $\geq_h$ (or $\geq_h^-$, depending on whether η-reduction is present) is: if $A \geq B$ then $A \geq_h C \geq B$ where the reduction from C to B consists solely of internal reductions. So if $A \geq B$ and A, B have different shapes, then $A >_h A' \geq B$.

4.4.6. The intuition formulated in 4.4.4. leads us to the idea that a sensible decision process for definitional equalities must search for a common reduct (i.e. an affirmative answer) rather than normalize, by means of $\geq_h$ (in order to get a negative answer), and that during the reduction process the definitional constants must be saved, i.e. left intact, as much as possible.

The strategy presented below (corresponding to what is actually implemented in Eindhoven [75]) can indeed be characterized by the following principles:

(1) decomposition is preferred above main reduction
(2) β-reduction is preferred above δ-reduction (is preferred above η-reduction)
(3) reduction of a "younger" definitional constant is preferred above reduction of the "older" one (see sec. III.6.3).

For example, if there is to be decided whether $\{A\}B \downarrow \{C\}D$, the process first tries decomposition: $B \downarrow D$ and $A \downarrow C$. If this succeeds, i.e. $B \geq F \leq D$, $A \geq G \leq C$ then we have a common reduct $\{G\}F$. Only after this has failed, an outside reduction is attempted on one of the expressions: e.g. $\{A\}B >_h E$, i.e. $B \geq [x{:}\alpha]F$, $E \equiv F[\![A]\!]$, and the new question to be decided is $E \downarrow \{C\}D$. Was no outside reduction possible, then the other expression is tackled: $\{C\}D >_h E$ is tried, possibly resulting in a new question $\{A\}B \downarrow E$. And, when confronted with the question $\{A\}B \downarrow d(\bar{C})$, the process tries to main reduce the appl-expression rather than the other one.

4.4.7. The inductive *definition* of $>_h$ and $\geq_h$ can be read as a recursive algorithm for deciding questions of the form $A \geq_h B$, $\exists_B (A >_h B)$, $\exists_{B_1} \exists_{B_2} (A \geq_h [x{:}B_1]B_2)$ etc. We give our algorithm for deciding $\downarrow$ also in the form of an inductive definition. Here are the rules:

(0) Exchange: $B \downarrow A \Rightarrow: A \downarrow B$

(i) Variable, τ: $A \geq_h x \Leftrightarrow : A \downarrow x$, and $A \geq_h \tau \Leftrightarrow: A \downarrow \tau$

(ii) Prim: $(A \geq_h p(\bar{C}), \bar{C} \downarrow \bar{B}) \Leftrightarrow: A \downarrow p(\bar{B})$

(iii) Appl-appl, decompose: $B \downarrow D,\ A \downarrow C \Rightarrow: \{A\}B \downarrow \{C\}D$

(iv) Appl, β-red: $\{A\}B >_h C \Rightarrow (C \downarrow D \Leftrightarrow: \{A\}B \downarrow D)$

(v) Def-def, decompose: $\bar{B} \downarrow \bar{C} \Rightarrow: d(\bar{B}) \downarrow d(\bar{C})$

(vi) Def, δ-red: $d(\bar{B}) >_h C \Rightarrow (C \downarrow D \Leftrightarrow: d(\bar{B}) \downarrow D)$

(vii) Abstr-abstr, decompose: $\alpha \downarrow \beta,\ A \downarrow B \Leftrightarrow: [x{:}\alpha]A \downarrow [x{:}\beta]B$

(viii) Abstr, η-red: $[x{:}\alpha]A >_h B \Rightarrow (B \downarrow C \Leftrightarrow: [x{:}\alpha]A \downarrow C)$

The notation $\bar{B} \downarrow \bar{C}$ is used in the ordinary sense, i.e. $B_1 \downarrow C_1$, $B_2 \downarrow C_2$ etc. The clauses (i)-(viii) are given in their order of priority, they have to be tried successively until a clause applies. Clause (0) must only be applied, and of course only once: (1) if none of the rules (i)-(viii) applies, (2) if by the exchange a rule of higher priority among (i)-(viii) can be made to apply, (3) in case the question $d(\bar{A}) \downarrow e(\bar{B})$ is presented, where e is a "younger" definitional constant than d. The clauses containing a bi-implication ((i), (ii), (vii)) are *terminal*: if application of one of these rules does not lead to an affirmative answer, a negative conclusion about the presented definitional equality can be drawn. In contrast with the other clauses, e.g. clause (iii): if not $(A \downarrow C)$, so not $(A \downarrow C$ and $B \downarrow D)$ then it is of course very well possible that rule (iv) produces a common result of $\{A\}B$ and $\{C\}D$. Further, a negative conclusion can be drawn if after exchanging still no clause applies at all. If η-reduction is not allowed then one has to read $>_h^-$ and $\geq_h^-$ instead of $>_h$ and $\geq_h$, and rule (viii) has to be skipped.

4.4.8. It should be clear that the algorithm above on the correct expressions indeed corresponds with $\downarrow$. The only interesting point is the bi-implication in clause (vii), which makes that clause (viii) never has to be applied to a pair of abstr-expressions. This is justified by our property UD (for correct expressions only) from the previous sections.

We also have to show the termination of the algorithm (this shows the decidibility of $\downarrow$ once more). First, the questions concerning $>_h$ and $\geq_h$ (e.g. whether $A \geq_h [x{:}B_1]B_2$ for certain B_1, B_2) are decidable on behalf of SN. Secondly, the procedure sketched above (for deciding $A \downarrow B$) is easily shown to terminate by induction on (1) $\theta(A) + \theta(B)$, (2) $\ell(A) + \ell(B)$ - where θ stands for length of reduction tree and ℓ stand for length of expression -.

Clearly the η-rule (viii) is equivalent to:

$$A \downarrow \{x\}B,\ B \geq C,\ x \notin \mathrm{FV}(C) \Rightarrow [x:\alpha]A \downarrow B$$

By a careful implementation of the handling of bound variables - this falls outside the scope of my thesis - there can be guaranteed that whenever during actual verification an equality $[x:\alpha]A \downarrow B$ is offered to the decision procedure, B does not contain free occurrences of the same free variable x. This enables us to modify (viii) into the simpler rule (viii'): $A \downarrow \{x\}B \Rightarrow [x:\alpha]A \downarrow B$, which avoids the nasty internal reductions in the course of an outside η-reduction completely. The termination of the algorithm is still guaranteed with this new rule; we can even use the same induction as before, because it can be shown that rule (viii') never will be applied with a B such that $B \geq_h [y:\beta]C$.

4.4.9. In accordance with our views on the actual verification process it may be sensible to provide the decision procedure with a device which gives a warning in the following cases: (1) if the decision process requires too much time, or rather: too many reduction steps (2) if a question $d(\bar{B}) \downarrow d(\bar{C})$ or $\{A\}D \downarrow \{F\}G$ is posed and not $(\bar{B} \downarrow \bar{C})$, resp. $(D \downarrow G$ and not $(A \downarrow F))$ has been concluded.

The warnings in case (2) can be partly motivated by the idea that most defined constants in an Automath-book are "λI-constants" (see III.5.5.3, III.6.3) and that most functions in an Automath-book are λI-*functions*, where D is a λI-*function* if: $D \downarrow [x:\alpha]F \Rightarrow x \in \mathrm{FV}(F)$. The following example shows however that this motivation is not quite satisfactory: $D \equiv G \equiv [x:\alpha]\{V\}x$, $A \equiv [y:\beta]p(y,V)$, $F \equiv [y:\beta]p(y,y)$.

4.4.10. Now we discuss the verification of E-formulas. Since the definitions of cantyp in 4.2.3, with their computation of normal forms, are very unpractical, we prefer the alternative approach sketched in 4.1.4. Besides, the latter approach avoids the different definitions of cantyp and is by uniformity easier to implement for several languages simultaneously.

As our "universe", the large language which we use to decide our E-formulas, we take AUT-QE*. Let $\vdash$ denote correctness in AUT-68, AUT-68+, AUT-QE or AUT-QE+ and let $\vdash_*$ stand for correctness in AUT-QE*. One easily proves by induction on A, using LQ, CLPT etc. for $\vdash_*$, the

important properties: (1) $\vdash A \Rightarrow \vdash_* \mathsf{typ}(A)$, and - unless A is a 2-expression in AUT-68(+) -

(2) $\vdash A \Rightarrow \mathsf{typ}(A) \geq \mathsf{cantyp}(A)$.

This justifies the equivalence mentioned in 4.1.4.

$$\vdash A \;\mathsf{E}\; B \Leftrightarrow \vdash A,\ \vdash B,\ \vdash_* \mathsf{typ}(A) \sqsubset B$$

except, trivially, the degree 2 case of AUT-68(+)

$$\vdash^2 A \;\mathsf{E}\; B \Leftrightarrow \vdash^2 A,\ B \equiv \tau$$

The $\downarrow$-procedure of sec. 4.4.7 can be adapted in order to decide $\downarrow$ and simultaneously by making some obvious modifications, e.g.:

- clause (0) becomes: $B \downarrow/\sqsubset/\sqsupset A :\Leftrightarrow A \downarrow/\sqsupset/\sqsubset B$
  (where "$B \downarrow/\sqsubset/\sqsupset A$" reads "$B \downarrow A$ resp. $B \sqsubset A$ resp. $B \sqsupset A$", etc.)
- to clause (i) there is added: degree $(A) = 1 \Rightarrow A \sqsubset \tau$
- clause (vii) becomes: $\alpha \downarrow \beta,\ A \downarrow/\sqsubset/\sqsupset B \Leftrightarrow: [x\ \alpha]A \downarrow/\sqsubset/\sqsupset [x{:}\beta]B$

etc.

We do not bother to give a practical algorithm for deciding E in AUT-QE' and AUT-QE*, because we think that these languages are of mere theoretical purpose.

4.4.11. Rather than computing domains via the domain normal forms (dnf's) of sec. 4.2.4.4. we use the alternative approach of 4.1.6 of searching through the $\rightarrow$-reduction tree of an expression. Recall that $\rightarrow$ is generated by (1) ordinary reduction, (2) taking typ. We promised the following theorem.

Theorem: $\rightarrow$ is well-founded on the correct expressions

Proof: As long as we stay inside the correct expressions we can use a double induction, viz. (1) on degree, (2) on θ(=length of reduction tree). For, reduction preserves degree and decreases θ, and taking typ decreases degree. We must be a bit careful with applying typ to a degree 2 AUT-QE* expression - such as, e.g., can originate by taking typ of a degree 3 AUT-QE expression - because an incorrect and even not normable 1-expression might arise. A typical example is $\{A\}\tau$. However, this does no harm to the well-foundedness, because β^1-SN can be proved, without using norms at all, for all degree correct expressions.

Also, we have another uniqueness result (compare 4.2.4.2).

Theorem: A correct, $A \rightarrow [x{:}\alpha]C$, $A \rightarrow [x{:}\beta]D \Rightarrow \alpha \downarrow \beta$

Proof: For 3-expressions A we even have a kind of CR-result $A \geq A' \Rightarrow \mathsf{typ}(A) \downarrow \mathsf{typ}(A')$. Now let degree $(A) = 2$, and let $A \geq A'$. In AUT-68(+) and AUT-QE(+) this gives $\vdash_* \mathsf{typ}(A) \sqsupseteq \mathsf{typ}(A')$, but in AUT-QE* this is not generally true, because $\mathsf{typ}(A)$ and $\mathsf{typ}(A')$ need not be correct. Luckily such incorrect 1-expressions (see the proof of the previous theorem) never reduce to an abstr-expression. So by UD we still get the desired result.

4.4.12. The internal η-reductions included in $\rightarrow$ are of course useless during domain computation where one only wants to reach an abstr-expression. So in an algorithm for domain computation we rather employ a restriction of $\rightarrow$ which we name $\rightarrow_h$ and is generated by head reduction $\geq_h^-$ and taking typ.

In general unrestricted search through the $\rightarrow_h$-reduction tree can be permitted - provided the degree restrictions are respected. However, the 2-expressions of AUT-QE and AUT-QE+ form an exception. Here the search for an abstr-expression *has to* start with taking typ. Otherwise too many expressions would get a domain, which would give rise to typical AUT-QE* appl-expressions.

Besides, unrestricted search can be very unpractical. E.g. in AUT-68(+) one never needs to inspect 1-expressions: if the 2-expressions in the $\rightarrow_h$-reduction tree fail to produce a domain, going to the 1-expression by taking typ will not help. In general it is no good strategy to start the domain computation with reduction, unless we are obliged to because the expression under consideration is already of minimal value degree.

So, a simple and probably rather practical strategy for AUT-68(+) and AUT-QE(+) may run as follows. Let A be the expression we start with. Take typ until one arrives at an expression of minimal value degree. Then reduce (with $\geq_h^-$) until one possibly finds a domain. If this does not succeed, A can still have a domain if it is a 3-expression of AUT-QE(+), otherwise A has no domain. In the indicated case unrestricted search of the $\rightarrow_h$-reduction tree of $\mathsf{typ}(A)$ is required, to be executed as follows: one-step reduce $(\mathsf{typ}(A) >_h^- B)$, then take typ, then reduce (with $\geq_h^-$). If this does not yield a domain, one-step reduce B once more etc. The well-foundedness of $\rightarrow$ guarantees the termination of this procedure.

CHAPTER VI. THE βη-CHURCH-ROSSER PROBLEM OF GENERALIZED TYPED λ-CALCULUS

VI.1. Introduction

1.1. The problem with βη-CR in Automath-like languages was first pointed out by Nederpelt ([51], p.71). Let $x \notin FV(\beta)$, then

$$[x:\alpha]C <_\beta [x:\alpha]\{x\}[x:\beta]C >_\eta [x:\beta]C$$

and the question is whether $[x:\alpha]C$ and $[x:\beta]C$ have a common reduct, i.e. whether $\beta\eta\text{-CR}_1$ holds. In untyped λ-calculus this case of CR_1 is particularly trivial, because without the type-labels there just remains

$$\lambda x.C <_\beta \lambda x.(\lambda x.C)x >_\eta \lambda x.C$$

and for the common reduct we can simply take $\lambda x.C$ itself. If $[x:\alpha]\{x\}[x:\beta]C$ is not necessarily correct, a common reduct does not need to exist, for α and β can be any expressions.

Nederpelt conjectured already that for correct expressions βη-CR (so $\beta\eta\text{-CR}_1$) does hold. This we shall prove below, making free use of the results of the previous chapter, in particular sec. 3. So, if $\vdash [x:\alpha]\{x\}[x:\beta]C$ then by SA we know $\alpha\ Q\ \beta$ so $[x:\alpha]C\ Q\ [x:\beta]C$; but we know nothing about a common reduct.

It is possible that certain versions of the algorithmic definition allow a proof of $\beta\eta\text{-CR}_1$. But then it is not so easy to infer CR, because we do not yet know CL for the algorithmic system. An alternative to the approach below is presented in the next chapter. There CR and CL are proved simultaneously for an algorithmic system, by induction on so-called *big trees*.

1.2. Below we concentrate on βη-reduction and leave δ-reduction out of consideration. It is easy to extend our result to βηδ-CR, since δ commutes with βη-reduction:

$$B \leq_\delta A \geq_{\beta\eta} C \Rightarrow B \geq_{\beta\eta} D \leq_\delta C$$

and, of course, δ-CR holds.

We start (in sec. 2) with a partial solution of the βη-problem, for η-reduction of degree 2, which works for regular langages only. Then (sec. 3) we prove full βη-CR.

VI.2. A first result concerning βη-CR for regular languages

2.1. We prove the Church-Rosser property for regular languages with a reduction relation $\geq$ generated by β-reduction and η^2-reduction, i.e. η-reduction of degree 2: degree(A) = 2, $x \notin FV(A) \Rightarrow [x:\alpha]\{x\}A >^2_\eta A$.

The motivation for studying this restricted βη-reduction lies in the fact that the actual verification of mathematics in AUT-QE (in particular, of Jutting's Landau-translation, see [37]) just required this specific type of η-reduction. I.e. the Automath texts offered to the verification program appeared to be correct $\beta\delta\eta^2$-AUT-QE.

2.2. Heuristics

The idea is to proceed in two stages. First we consider a seemingly weaker form of η^2-reduction which is tailor-made to avoid the critical βη-case mentioned in the introduction. For this restricted $\beta\eta^2$-reduction we prove CR. Afterwards (sec.2.5) it is shown that full $\beta\eta^2$-equality is equivalent to the restricted form. This can be compared with the situation in sec. V.3.3.8 - where η^1-equality turned out to be provable.

How to define the restricted form of η-reduction? I.e. under which conditions do we permit the reduction of $[x:\alpha]\{x\}A$ to A? Clearly, we require:

(1) $x \notin FV(A)$

Further, that A is not of the form $[y:\beta]C$ - to avoid the critical case -. But this is not enough. Consider, e.g., $[x:\alpha]\{x\}F$, where $F \geq [y:F_1]F_2$, $x \notin FV(F)$. So we require:

(2) $A \not\geq [y:\beta]C$

i.e. A does not reduce to an expression of the form $[y:\beta]C$.
Thirdly we want to preserve the substitution lemma

$$B \geq B' \Rightarrow B[\![D]\!] \geq B'[\![D]\!]$$

at least for D of degree 3, so we further require

(3) degree(A) = 2

This shows why the method works for regular languages only.
Condition (2) can now be weakened to

(2') $A \not\equiv^2_\beta [y{:}\beta]C$

or, in the presence of δ-reduction, to: $A \not\equiv^2_{\beta\delta} [y{:}\beta]C$.

2.3. The definition of the restricted reduction relation

For definiteness we give a formal definition:

(1) > is the disjoint one-step reduction generated by the elementary reductions:

(i) $\{A\}[x{:}B]C > C[\![A]\!]$

(ii) $x \notin FV(A)$, $A \not\equiv^2_\beta [y{:}\beta]C$, degree($A$) = 2 ⇒ $[x{:}\alpha]\{x\}A > A$

(2) ≥ is the transitive closure of >

2.4. The proof of CR for the restricted reduction

2.4.1. Substitution lemma I: (i) $A > A' \Rightarrow B[\![A]\!] > B[\![A']\!]$

(ii) $A \geq A' \Rightarrow B[\![A]\!] \geq B[\![A']\!]$

Proof: As usual, by induction on B and ≥ respectively.

2.4.2. *Weak* β^i-β^j-*postponement*: if $i \neq 3$ and A is degree correct then

$$A \geq^{ij}_\beta B \Rightarrow A \geq^i_\beta C \geq^j_\beta D \leq^i_\beta B$$

Proof: If a β^j-contraction produces an essentially new β^i-redex then i=3 or i=j. If i=j there is nothing to prove, so unless i=3 we have $A >^j_{1,\beta} \geq^i_{1,\beta} B \Rightarrow A >^i_{1,\beta} \geq^j_\beta C \leq^i_\beta B$. So, using β^i-SN, β^i-CR and the fact that β^i and β^j commute we get the desired property, as in II.7.4.

2.4.3. Something about β^2 (for degree correct expressions)

(i) Degree$(B) = 2$, $B \geq [y{:}C]D \Rightarrow B \geq^2_\beta [y{:}C']D'$

(ii) If degree$(B) = 2$, degree(A) = degree$(x) = 3$ then

$$B[\![x/A]\!] \geq^2_\beta [y{:}C]D \Rightarrow B \geq^2_\beta [y{:}C']D'$$

Proof: (i) Let $B \geq [y{:}C]D$, degree$(B) = 2$. By $\beta\eta$-postponement and weak β^2-β^3-postponement we get $B \geq^2_\beta F \geq^3_\beta G \leq^2_\beta H \geq_\eta [y{:}C]D$. Then H, G, F are abstractions expressions, q.e.d.

(ii) Use the square brackets lemma (II.11.5, IV.2.4) and the previous property.

2.4.4. Substitution lemma II: if degree(A) = degree$(x) = 3$ and A, B are degree correct then

(i) $B > B' \Rightarrow B[\![x/A]\!] > B'[\![x/A]\!]$

(ii) $B \geq B' \Rightarrow B[\![x/A]\!] \geq B'[\![x/A]\!]$

Proof: (i) By induction on B. The crucial case is when $B \equiv [y{:}B_1]\{y\}B_2$, $y \notin FV(B_2)$, $B_2 \not\geq^2_\beta [y{:}C]D$, degree$(B)$ = = degree$(B_2) = 2$. Of course, $y \notin FV(B_2[\![A]\!])$, degree$(B_2[\![A]\!]) = 2$ and, by 2.4.3.(ii) $B_2[\![A]\!] \not\geq^2_\beta [y{:}C]D$. So $B[\![A]\!] \equiv [y{:}B_1[\![A]\!]]\{y\}B_2[\![A]\!] > B_2[\![A]\!]$ q.e.d.

(ii) By induction on $\geq$.

2.4.5. Theorem (CR_1 for the restricted reduction): if A degree correct then

$$A > B,\ A > C \Rightarrow B \downarrow C$$

Proof: Let $A > B$, $A > C$. By induction on A we define a common reduct D of B and C. The crucial cases are

(i) $A \equiv \{A_1\}[x{:}A_2]A_3$, $B \equiv A_3[\![A_1]\!]$ (by β-red.), $C \equiv \{A_1'\}[x{:}A_2']A_3'$ (by monotonicity). Take $D \equiv A_3'[\![A_1']\!]$ and use the substitution lemmas.

(ii) $A \equiv \{A_1\}[x{:}A_2]\{x\}A_3$, $B \equiv \{A_1'\}A_3$ (by η-red. and monotonicity), $C \equiv \{A_1\}A_3$ (by β-red.). Simply take $D \equiv B$.

(iii) $A \equiv [x{:}A_1]\{x\}A_2$, $B \equiv A_2$ (by η-red.), $C \equiv [x{:}A_1']\{x\}A_2'$ (by monotonicity). Clearly degree(A_2') = degree$(A_2) = 2$, $x \notin FV(A_2')$.

If $A_2' \geq_\beta^2 [y{:}C_1]C_2$ then $A_2' \geq [y{:}C_1]C_2$ so by 2.4.3.(i) $A_2 \geq_\beta^2 [y{:}C_1']C_2'$. Hence $A_2' \not\downarrow_\beta^2 [y{:}C_1]C_2$ so $D \equiv A_2'$ can serve as the common reduct.

2.4.6. Corollary: If A degree correct and normable then $CR(A)$.

Proof: By induction on the reduction tree of A.

2.5. The extension to full $\beta\eta^2$-reduction

2.5.1. From now on we label the notions referring to the restricted reduction relation with a subscript o. Thus we write $>_o$, $\geq_o$ and $\downarrow_o$, and by $\vdash_o$ we denote correctness in AUT-QE(+) with an equality relation Q_o generated, e.g., by

$$\vdash_o A,\ \vdash_o B,\ A >_o B \text{ or } B >_o A \Rightarrow A\ Q_o\ B$$

By 2.4.6. we have

$$A\ Q_o\ B \Rightarrow A \downarrow_o B.$$

On the other hand the notations without a subscript have to be interpreted in terms of "full" $\beta\eta^2$-reduction. Thus, we write $\vdash$ for correctness in AUT-QE(+) with equality Q, generated by

$$\vdash A,\ \vdash B,\ A > B \text{ or } B > A \Rightarrow A\ Q\ B.$$

Below we sketch the equivalence of the two systems. The implications $>_o \Rightarrow >$ so $\vdash_o \Rightarrow \vdash$ and $Q_o \Rightarrow Q$ are immediate.

2.5.2. First we go through some theory of the o-language (i.e. with $\vdash_o$ and Q_o). The theorems about *renaming of contexts* and *weakening* (see V.2.9) are still valid. We have a restricted *substitution theorem*: If $\eta \equiv (\eta_1, \bar{y}\ E\ \bar{\beta})$, all y_i in $\bar{y}$ have degree 3, and $\bar{B}\ E\ \bar{\beta}[\![\bar{B}]\!]$ then

$$\eta \vdash_o C(E/Q_o\ D) \Rightarrow \eta_1 \vdash_o C[\![\bar{B}]\!](E/Q_o\ D[\![\bar{B}]\!])$$

So we have the *single substitution theorem*: if degree(y) = 3 then

$$\vdash_o B\ E\ \beta,\ y\ E\ \beta \vdash_o C(E/Q_o\ D) \Rightarrow \vdash_o C[\![B]\!](E/Q_o\ D[\![B]\!]).$$

Hence, from SA^i we can infer 2.4 β^i-CLPT, as usual. Now SA^2 works precisely as in the previous chapter (V.3.2.4) so we may assume β^2 - CL.

2.5.3. The proof that $\vdash \Rightarrow \vdash_o$ and $Q \Rightarrow Q_o$ goes by induction on $\vdash$. The only interesting case is when $\vdash^2[x:\alpha]\{x\}A$, $x \notin FV(A)$, $A \geq^2_\beta [x:A_1]A_2$. Then η^2-reduction is possible, but restricted reduction is not. So from $\vdash A$ one gets $\vdash[x:\alpha]\{x\}A\ Q\ A$ and we like to show that $\vdash_o[x:\alpha]\{x\}A\ Q_o\ A$ holds as well. By the ind. hyp. $\vdash_o[x:\alpha]\{x\}A$ and $\vdash_o A$, and by β^2-CL $A\ Q_o\ [x:A_1]A_2$ and $[x:\alpha]\{x\}A\ Q_o\ [x:\alpha]\{x\}[x:A_1]A_2\ Q_o\ [x:\alpha]A_2$. By SA^2 $\alpha\ Q_o\ A_1$ so by the substitution theorem $[x:\alpha]A_2\ Q_o\ [x:A_1]A_2$, whence $[x:\alpha]\{x\}A\ Q_o\ A$.

2.5.4. So the o-language is equivalent with the $\beta\eta^2$-language, for which the properties CL, PT, SA etc. can be proved as in the previous chapter. Now let $A\ Q\ B$. By the equivalence $A\ Q_o\ B$ and by CR $A \downarrow_o B$, so a fortiori we have CR for all full $\beta\eta^2$-reduction.

Extension to the corresponding δ-language is possible as in sec. V.3.3.

VI.3. A proof of CR for full $\beta\eta$-reduction from closure and strong normalization

3.1. The assumptions

3.1.1. In contrast with the proof in the previous section, the sequel does not presuppose regularity of the language. So, after having proved CL for, e.g., Nederpelt's Λ, the present proof applies to this language. We assume that correctness of expressions and equality formulas is defined relative to a correct book $\mathcal{B}$ and a context ξ. The book is fixed throughout this section and omitted in the notation.

Below we introduce an extended reduction relation and a correspondingly extended equality. Since we want to reserve our usual notations $\geq$, Q for these new relations, we write $\geq_o$ and Q_o for the ordinary $\beta\eta$-reduction and the corresponding equality relation, generated e.g., by by

$$\xi \vdash A,\ \xi \vdash B,\ A \geq_o C \leq_o B \Rightarrow \xi \vdash A\ Q_o\ B.$$

We use our ordinary shorthand notation, writing

$$\eta \vdash A \quad \text{for} \quad \xi,\eta \vdash A \quad \text{and}$$

$$A\ Q_o\ B \quad \text{for} \quad \xi \vdash A\ Q_o\ B \text{ etc.}$$

3.1.2. For definiteness we give a list of the properties which we assume through this section and use in the proof.

(1) Strengthening, and in particular the following consequence:
if $\eta \equiv (\eta_0,\eta_1)$ then

$$\eta_0 \vdash A,\ \eta_0 \vdash B,\ \eta \vdash A\ Q_o\ B \Rightarrow \eta_0 \vdash A\ Q_o\ B$$

(2) Soundness of equality w.r.t. abstraction,

$$\alpha\ Q_o\ \beta,\ x\ E\ \alpha \vdash A\ Q_o\ B \Leftrightarrow [x{:}\alpha]A\ Q_o\ [x{:}\beta]B$$

(3) w.r.t. application,

$$A\ Q_o\ B,\ C\ Q_o\ D \Rightarrow \{A\}C\ Q_o\ \{B\}D$$

(a consequence of LQ, see below)

(4) and w.r.t. substitution

$$A_1\ Q_o\ A_1',\ \ldots,\ A_k\ Q_o\ A_k' \Rightarrow c(\bar{A})\ Q_o\ c(\bar{A}')$$

(also a consequence of LQ)

(5) closure: $\vdash A,\ A \geq_o B \Rightarrow \vdash B$

(6) SA, so (this concerns directly the critical $\beta\eta$-case)

$$\vdash [x{:}\alpha]\{x\}[y{:}\beta]C \Rightarrow x\ E\ \alpha \vdash \alpha\ Q_o\ \beta$$

(7) strong normalization (with respect to $\geq_o$): $\vdash A \Rightarrow SN(A)$.

Remark: the properties (3) and (4) depend on LQ. As we know (see V.3.3.10) LQ fails in AUT-QE(+) with δ-reduction, but CR for these languages can be proved in two ways:

(1) From CR for AUT-QE(*)

(2) By first proving CR for a δ-less version, and then extend the result by using UE.

3.2.1 Heuristics

We saw that in the critical case of βη-reduction the two direct reducts of $[x:\alpha]\{x\}[x:\beta]C$ are *syntactically equal* (≡) *but for the domains* α *and* β *which are just definitionally equal* (Q_0). Below we define the relation ≈ which precisely covers this kind of *syntactic similarity* intermediate between ≡ and Q_0.

It would be straightforward to try and prove a modified CR-property

$$B \leq_0 A \geq_0 C \Rightarrow B \geq_0 D \approx D' \leq_0 C$$

by proving ≈-postponement, i.e.

$$A \approx B \geq_0 C \Rightarrow A \geq_0 B' \approx C$$

However there is a problem with the latter property if $A \equiv [x:\alpha]\{x\}A_1$, $B \equiv [x:\alpha]\{x\}C$, $x \notin FV(C)$, $A_1 \approx C$. For it is possible that $x \in FV(A_1)$. So we take a different approach. We define an extended reduction relation > which is disjoint βη-one-step reduction, enriched by the clause

$$A \approx B \Rightarrow A > B \quad (\textit{elementary} \approx\textit{-reduction}).$$

This means that internal contractions in the domains for the bookkeeping of reduction steps are ignored. For the new reduction relation we can simply prove CR_1. Further there holds a certain version of ≥-SN, which gives us CR.

3.2.2. Structure of the proof

We point out the difference with the approach in sec. VI.2. There we first restricted our reduction relation, proved CR for the restricted reduction and then extended the result to the original reduction. On the other hand, here we start with proving CR for the extended reduction relation ≥, and afterwards we still must prove CR for $\geq_0$. In fact we first prove modified uniqueness of ≥-normal form, i.e. uniqueness with

respect to $\approx$: $A\ Q\ B$, A and B $\geq$-normal $\Rightarrow A \approx B$. And then, using the equivalence of Q_o and Q, uniqueness of $\geq_o$-normal form. So we have $\geq_o$-CR. For a comparison of $\geq_o$- and $\geq$-normalisation see sec. 3.7.1 below.

3.3. Definition of the extended reduction relation

3.3.1. By simultaneous inductive definition we introduce the *syntactic similarity* $\approx$, the *extended reduction* relation $\geq$, with one-step reduction $>$, and the *extended definitional equality* Q, between correct expressions, as follows.

I. Elementary reductions

(1) $\{A\}[x{:}B]C > C[\![A]\!]$ (β-reduction)

(2) $[x{:}B]\{x\}C > C$ if $x \notin FV(C)$ (η-reduction)

(3) $A \approx B \Rightarrow A > B$ ($\approx$-reduction)

II. Monotonicity rules

(1) $A > A'$, $B > B' \Rightarrow \{A\}B > \{A'\}B'$

(2) $x\ E\ \alpha \vdash A > A' \Rightarrow [x{:}\alpha]A > [x{:}\alpha]A'$

(3) $A_1 > A_1'$, ..., $A_k > A_k' \Rightarrow C(\bar{A}) > C(\bar{A}')$

III. (1) $\geq$ is the transitive closure of $>$

(2) Q is the equivalence generated by $>$

IV. (1) $A \approx A$

(2) $\alpha\ Q\ \alpha'$, $x\ E\ \alpha \vdash B \approx B' \Rightarrow [x{:}\alpha]B \approx [x{:}\alpha']B'$

(3) $A \approx A'$, $B \approx B' \Rightarrow \{A\}B \approx \{A'\}B'$

(4) $A_1 \approx A_1'$, ..., $A_k \approx A_k' \Rightarrow C(\bar{A}) \approx C(\bar{A}')$

3.2.2. Some remarks concerning the definition

3.3.2.1. It is not necessary to define the above notions simultaneously.

For in view of 3.4.3. below, we might as well have taken instead of IV.(2)

IV.(2') $\alpha\ Q_o\ \alpha',\ x\ E\ \alpha \vdash B \approx B' \Rightarrow [x{:}\alpha]B \approx [x{:}\alpha']B'$

3.3.2.2. Except for the rules I.3 and II.2, the rules of I and II are the ordinary rules for $\bar{>}_{1,\beta\eta}$, disjoint one-step $\beta\eta$-reduction. Rule I.3 can be considered a strong form of the reflexivity rule $A > A$. Rule II.2 is one half of the usual monotonicity rule for abstr. expressions. The other half can be derived using IV.1, IV.2 and I.3: if $\alpha > \alpha'$ then $\alpha\ Q\ \alpha'$, further $A \approx A$ so

$$[x{:}\alpha]A \approx [x{:}\alpha']A \quad \text{so} \quad [x{:}\alpha]A > [x{:}\alpha']A\ .$$

3.3.2.3. If we had defined > to be the corresponding "nested" one-step reduction we might have been able to prove the diamond property for >. Then we could have avoided the appeal to SN when deriving CR from CR_1.

3.4. Some easy properties

3.4.1. By simultaneous induction on definition 3.3.1., using the soundness of Q_o w.r.t. expression formation, we get

$$\text{if } A > A' \text{ or } A \geq A' \text{ or } A\ Q\ A' \text{ or } A \approx A' \text{ then } A\ Q_o\ A'$$

3.4.2. From 3.3.2.2. it is clear that $\geq$ satisfies all the monotonicity rules and that

$$A >_o B \Rightarrow A \geq B,$$

so $$A \geq_o B \Rightarrow A \geq B,$$

and $$A\ Q_o\ B \Rightarrow A\ Q\ B$$

3.4.3. So combining this we have $Q_o \Leftrightarrow Q$.
As a corollary we have the monotonicity rules 3.1.2.(2)-(4) now also for Q. The monotonicity of $\approx$ is immediate. Further $\approx$ is an equivalence relation.

3.5. On $\approx$-reduction and normalization

3.5.1. In certain λ-calculus systems (see, e.g.[25]) renaming of bound variables is not ignored - like we do here - but formalized in the form of α-*reduction*:

$$y \notin FV(B) \Rightarrow [x:\beta]B >_{\alpha} [y:\beta]B[x/y]$$

Then (see our definition of substitution, sec.II.2.4) it is possible that α-reductions are needed before some β-reduction can be carried out. In such systems, a suitable definition of *proper reduction sequence* is: a sequence in which only a finite number of α-reductions occur. I.e. a reduction sequence $\Sigma_1 > \Sigma_2 > \ldots$ is proper if from a certain Σ_n on, only α-reductions are applied. Similarly Σ is *normal* if only α-reductions of Σ are possible.

3.5.2. Here we treat the $\approx$-reductions analogously, as extended α-reduction, and call them *improper* reductions. *Proper reduction sequences* are reduction sequences in which only a finite number of such improper reductions occur. An expression is now SN if all its proper reduction sequences terminate and *normal* if only improper reductions are possible. So

$$A \text{ is normal}, A \geq A' \Rightarrow A \approx A' .$$

3.5.3. In 3.5.1. we mentioned the possibility that α-reductions created new β-redices. For $\approx$-reductions this is not the case. Let $>_{\beta}$ (resp. $>_{\eta}$) denote the disjoint one-step reduction generated by the rules I.(1) (resp. I.(2)) and II of 3.3.1. So, e.g., $A >_{\beta} A'$ if some β-redices not lying inside a "domain" are contracted. Then we have, indeed, $\beta \approx$ - postponement

$$A \approx B >_{\beta} C \Rightarrow A >_{\beta} B' \approx C$$

However $\eta \approx$ -postponement fails because $\approx$ -reductions can create new η-redices (see 3.2.1.). Fortunately we have $\approx$ η-postponement instead

$$A >_{\eta} B \approx C \Rightarrow A \approx B' >_{\eta} C$$

3.5.4. Now we can prove SN (in the sense of 3.5.2). Let a proper

reduction sequence $\Sigma_1 > \Sigma_2 > \ldots$ be given. If no β-step turns up then the sequence terminates because from some Σ_n on only η-steps are applied, which decrease the length of the expression. Otherwise, for some n, by $\approx$ η-PP

$$\Sigma_1 \approx \Gamma \geq_\eta \Sigma_n >_\beta \Sigma_{n+1}$$

By $\beta\eta$-PP and $\beta\approx$-PP

$$\Sigma_1 >_\beta \Gamma' \approx \Gamma'' \geq_\eta \Sigma_{n+1}$$

By o-SN, i.e. SN with respect to $\geq_o$, $\theta_\beta(\Sigma)$ is defined for correct Σ and $\theta_\beta(\Sigma_1) > \theta_\beta(\Gamma')$. So by induction on θ_β we can prove SN.

3.6. CR for $\geq$

3.6.1. Substitution lemma I: If $\vdash B[\![A]\!]$, $\vdash B[\![A']\!]$ then

(i) $A > A' \Rightarrow B[\![A]\!] > B[\![A']\!]$

(ii) $A \geq A' \Rightarrow B[\![A]\!] \geq B[\![A']\!]$

(iii) $A \; Q \; A' \Rightarrow B[\![A]\!] \; Q \; B[\![A']\!]$

(iv) $A \approx A' \Rightarrow B[\![A]\!] \approx B[\![A']\!]$

Proof: All parts can be proved separately by ind. on B using the monotonicity rules for $>$, $\geq$, Q and $\approx$.

3.6.2. Substitution lemma II: If $\vdash B[\![A]\!]$ and $\vdash B'[\![A]\!]$ then

(i) $B > B' \Rightarrow B[\![A]\!] > B'[\![A]\!]$

(ii) $B \geq B' \Rightarrow B[\![A]\!] \geq B'[\![A]\!]$

(iii) $B \; Q \; B' \Rightarrow B[\![A]\!] \; Q \; B'[\![A]\!]$

(iv) $B \approx B' \Rightarrow B[\![A]\!] \approx B'[\![A]\!]$

Proof: By simultaneous induction on the definition of $>$, $\geq$, Q and $\approx$.

3.6.3. Main lemma (CR_1): If A correct, $B < A > C$ then $B \downarrow C$.

Proof: By ind. on A. If $A \approx B$ then for the common reduct D we can take $D \equiv C$. Similarly if $A \approx C$. In case $A \equiv \{A_1\}A_2$, $B \equiv \{B_1\}B_2$, $C \equiv \{C_1\}C_2$, $B_1 < A_1 > C_1$, $B_2 < A_2 > C_2$ then by the ind. hyp. and by monotonocity of $\geq$ we find a common reduct $\{D_1\}D_2$ with $B_1 \geq D_1 \leq C_1$, $B_2 \geq D_2 \leq C_2$. Similarly if $A \equiv C(A_1,\ldots,A_k)$.
Further distinguish:

(i) $A \equiv \{A_1\}[x{:}A_2]A_3$, $B \equiv \{B_1\}[x{:}B_2]B_3$, $C \equiv A_3[\![A_1]\!]$, $A_1 > B_1$, $A_2\ Q\ B_2$, $A_3 > B_3$. By the substitution lemmas above $B > B_3[\![B_1]\!] \leq A_3[\![A_1]\!]$ so take $D \equiv B_3[\![B_1]\!]$.

(ii) $A \equiv \{A_1\}[x{:}A_2]\{x\}A_3$, $B \equiv \{B_1\}A_3$ (by η-red.), $C \equiv \{A_1\}A_3$ (by β-red.), $x \notin FV(A_3)$, $A_1 > B_1$. Then $C \equiv B$ and take $D \equiv B$.

(iii) $A \equiv [x{:}A_1]A_2$, $B \equiv [x{:}B_1]B_2$, $C \equiv [x{:}C_1]C_2$, $A_1\ Q\ B_1$, $A_1\ Q\ C_1$ $B_2 < A_2 > C_2$. By ind. hyp. $B_2 \geq D_2 \leq C_2$ so take e.g. $D \equiv [x{:}B_1]D_2$.

(iv) $A \equiv [x{:}A_1]\{x\}A_2$, $B \equiv [x{:}B_1]\{x\}B_2$, $C \equiv A_2$ (by η-red.), $x \notin FV(A_2)$, $A_1\ Q\ B_1$, $A_2 > B_2$. It is easy to see that $A_2 \geq_{\beta\eta} D_2 \approx B_2$. Clearly $x \notin FV(D_2)$ so $B \approx [x{:}B_1]\{x\}D_2 > D_2 \leq A_2 \equiv C$. So take $D \equiv D_2$.

(v) $A \equiv [x{:}A_1]\{x\}[x{:}A_2]A_3$, $B \equiv [x{:}A_1]A_3$, $C \equiv [x{:}A_2]A_3$, $x \notin FV(A_2)$. This is the critical case. By assumption (6) from 3.1.2 $A_1\ Q\ A_2$ so we can take $D \equiv B \approx C$.

3.6.4. Theorem (CR): If A correct then $CR(A)$

Proof: By SN we can define $\theta(A)$ the maximal number of proper reductions steps in reduction sequences of A. Use induction on $\theta(A)$. Let $B \leq A \geq C$. The cases $A \approx B$ and $A \approx C$ are trivial. Otherwise, for certain proper reducts B_1 and C_1, $A > B_1 \geq B$, $A > C_1 \geq C$. First apply 3.6.3. to get $B_1 \geq D_1 \leq C_1$. Then apply the ind. hyp. to B_1, C_1 and D_1.

3.6.5. Corollaries: I. $A\ Q\ B \Rightarrow A \downarrow B$

II. similarity of normal forms:

$A\ Q\ B$, A and B normal $\Rightarrow A \approx B$

3.7. CR for $\geq_o$

3.7.1. Call an expression o-normal if it is normal with respect to $\geq_o$, i.e. if it does not contain β- or η-redices. So, if A o-normal then there are no reduction steps $A >_\beta B$ or $A >_\eta B$ possible. But it might be possible - as long as we do not have CR - that after some $\approx$-reductions new η-redices are created. So a priori we do not know whether A is normal.

But, if A is o-normal and A does not have abstraction form and $A \geq B$ then this reduction is an internal, and not a main reduction. E.g. $A \equiv \{A_1\}A_2 \Leftrightarrow B \equiv \{B_1\}B_2$, and:

$$A \equiv \{A_1\}A_2 \Rightarrow B \equiv \{B_1\}B_2,\ A_1 \geq B_1,\ A_2 \geq B_2$$

3.7.2. Theorem (uniqueness of o-normal form): Let A and B be o-normal, then

$$A\ Q_o\ B \Rightarrow A \equiv B$$

Proof: By induction on the sum of the lengths of A and B. Let $A\ Q_o\ B$, so $A\ Q\ B$, so $A \geq C \leq B$. Distinguish the following cases:

(1) Both A and B are abstr-expressions, $[x{:}A_1]A_2$ resp. $[x{:}B_1]B_2$. By prop. 3.1.2.(2), $A_1\ Q_o\ B_1$, $x \in A_1 \vdash A_2\ Q_o\ B_2$. By the ind. hyp. $A_1 \equiv B_1$, $A_2 \equiv B_2$ so $A \equiv B$.

(2) Neither A nor B are abstr-expressions. Then A and B and C have the same form. E.g. if $A \equiv \{A_1\}A_2$, then $C \equiv \{C_1\}C_2$, so $B \equiv \{B_1\}B_2$ with $A_1 \geq C_1 \leq B_1$ and $A_2 \geq C_2 \leq B_2$. So $A_1\ Q\ B_1$, $A_2\ Q\ B_2$ and $A_1\ Q_o\ B$, $A_2\ Q_o\ B$ and by the ind. hyp. $A_1 \equiv B_1$, $A_2 \equiv B_2$.

(3) A has abstr. form and B has not. Then $A \equiv [x{:}A_1]A_2$, $A_2 \geq \{x\}D_2$, $x \notin FV(D_2)$, $A_1\ Q\ D_1$, and $A \geq [x{:}D_1]\{x\}D_2 > D_2 \geq C \leq B$. By CL, $x \in D_1 \vdash \{x\}D_2$ and by 3.1.2.(3), $x \in D_1 \vdash \{x\}D_2\ Q\ \{x\}B$. So $x \in A_1 \vdash A_2\ Q\ \{x\}B$ and both A_2 and $\{x\}B$ are o-normal. By the ind. hyp. $A_2 \equiv \{x\}B$. Clearly $x \notin FV(B)$, so A is not o-normal, contradiction. So this case does not occur.

3.7.3. Corollary (CR):

(i) A correct, $A \geq_o B$, $A \geq_o C \Rightarrow B \geq_o D \leq_o C$

(ii) $A\ Q_o\ B \Rightarrow A \geq_o C \leq_o B$

3.7.4. Now we can conclude

A o-normal $\Rightarrow$ A normal

For, if A o-normal, $A \approx B >_\eta C$ (i.e. A is not normal) then $A \equiv \ldots[x{:}A_1]\{x\}A_2\ldots$, $x \in \mathrm{FV}(A_2)$, $B \equiv \ldots[x{:}B_1]\{x\}B_2\ldots$, $x \notin \mathrm{FV}(B_2)$, $A_1\ Q\ B_1$, $x\ E\ A_1 \vdash A_2\ Q\ B_2$. By CR, $B_2 \geq_o A_2$, so $\mathrm{FV}(A_2) \subset \mathrm{FV}(B_2)$, impossible.

CHAPTER VII. THE ALGORITHMIC DEFINITION AND THE THEORY OF NEDERPELT'S Λ: THE BIG TREE THEOREM, CLOSURE AND CHURCH-ROSSER

VII.1. Introduction and summary

1.1. The history of Λ

A further unification of the concepts underlying AUT-68 and AUT-QE led Nederpelt and the Bruijn [49, 50, 9], after the construction of an intermediate version λ-AUT, to the introduction of the language Λ or, as de Bruijn names it, AUT-SL, for: *single line* Automath.

First Nederpelt noticed that via a suitable translation *instantiation*, i.e. substitution in constant-expressions $c(x_1,\ldots,x_n)$, could be replaced by *application* and that, by this translation, δ-reduction reduced to β-reduction. We used this fact for one of our proofs of δ-SN in III.5.4. However, in order to cover substitution with 2-expressions, as is allowed in Automath languages, the restriction to *argument degree* 3 and *domain degree* 2 had to be dropped. This would in combination with type-inclusion have given a higher order system, so to avoid normability- and normalization problems, one had to skip type-inclusion. Then, a further streamlining of the definition was attained by dropping the restriction as to *inhabitable degree* as well, thus allowing expressions of any degree.

By the aforementioned translation and the relaxation of the degree restrictions it became possible to dispense completely with constants and schemes: constants could be translated into variables, schemes could be turned into assumptions and a book could be transformed into a context. Besides, quantification over all free variables was allowed now, so all assumptions $x \in \alpha$ from a context could be converted into abstractors $[x:\alpha]$.

Thus, a statement $B;\xi \vdash A$ expressing the correctness of A w.r.t. book B and context ξ could be translated into the correctness of a single expression $[\bar{p}:\bar{\beta}][\bar{x}:\bar{\alpha}]A'$, where the abstractor strings $[\bar{p}:\bar{\beta}]$ and $[\bar{x}:\bar{\alpha}]$ and the expression A' are intended to symbolize the translations of B, ξ and A respectively. I.e. a whole book reduces to a *single line*. For details of the translation see 6.2.1, 6.3.3 and 6.4.6.

Resuming, Nederpelt's Λ - as defined in his dissertation - is characterized by the following three features: no degree restriction at all, no type-inclusion, and single-line presentation. His definition is a typical algorithmic definition - for the terminology see V.1.1. - which, due to these simplifications, is remarkably short and elegant. Nederpelt introduced his norm as a measure of functional complexity and proved normability, normalization and strong normalization for his system. He just conjectured, in the introduction to this thesis, that the system satisfied closure and βη-Church-Rosser.

1.2. The present treatment

The discussion in the previous chapters: starting from the E-definition (V.2), first proving closure (V.3) and βη-Church-Rosser (VI), and finally proving the equivalence with the algorithmic definition (V.4), though concentrating on the socalled regular languages AUT-QE and AUT-68, applies to Nederpelt's language as well, which shows that this conjectures were justified.

Here we choose an altogether different approach. Below we start with the algorithmic definition of correctness (VII.2). We follow Nederpelt but for his single-line presentation: we fit the system into the book-and-context framework of the previous chapters. Whereas the definition of the constant-less part of the language (sec. 2.1) simply can take place in the *pretyped* expressions(see IV.3), it turns out that adding constant-expressions (sec. 2.2) requires the introduction of *degree-norm correct* expressions (2.2.4).

Then both Nederpelt's conjectures are proved directly from the algorithmic definition, using the socalled *big-tree theorem* (BT). This theorem states that, on the correct expressions - and, in fact, on the much larger domain of normable expressions - the partial order $\overset{*}{\rightarrow}$ generated by Sub (i.e. taking proper sub-expressions), by ≥ and by taking typ is well-founded. So BT is an SN-result for an extended reduction relation and, hence, implies ordinary SN. The big tree theorem was first formulated and proved by de Vrijer [70] for his regular language λλ.

Section 3 below contains the closure proof of Λ without constants, serving as a motivation for BT. Section 4 contains two different proofs of BT, and in sec. 5 we prove closure and CR for the constant-less part of Λη. In sec. 6 we give some equivalence proofs: of the systems with

and without (definitional) constants, and of the single-line version with the book-and-context presentation. As a result we get the various nice properties for all these systems.

VII.2 The definition of Λ and $\Lambda\eta$

2.1 The part without constant expressions

2.1.1 Both Λ and $\Lambda\eta$ are systems of *admissible* expressions in the sense of IV. . The correctness of books and contexts is standard (see so we just present the part of the definition concerning the correctness of expressions. A simplification compared with e.g. AUT-QE is that no degree restrictions are imposed. If in the definition below $>$ (resp. $\geq$, resp. $\not>$) is interpreted in terms of $\beta\eta$-reduction then we get $\Lambda\eta$ otherwise just Λ.

The function typ is defined as in IV.3.2, degrees are as in IV.4.4.2. Throughout sec. 2.1 we follow Nederpelt and do not admit constant-expressions. Later on (secs. 2.2, 2.3) we show how the language can be extended with the formation of constant expressions.

2.1.2 By taking typ of a *non-constant*-expression A the degree is decreased by one (see IV.3 and IV.4), so by successively taking typ one arrives at a 1-expression. This 1-expression is called $\mathrm{typ}^*(A)$. So,

$$\mathrm{typ}^*(A) :\equiv A \text{ if } \mathrm{degree}(A) = 1$$

$$\mathrm{typ}^*(A) :\equiv \mathrm{typ}^*(\mathrm{typ}(A)) \text{ otherwise.}$$

Now let $\mathcal{B}$ be correct and let ξ be correct w.r.t. $\mathcal{B}$. We use the conventional shorthand: $\eta\vdash A$ instead of $\mathcal{B};\xi,\eta\vdash A$, typ instead of ξ-typ etc. Of course, as long as we do not form constant-expressions, the presence of the book $\mathcal{B}$ is completely irrelevant. Now correctness of non-constant-expressions is defined as follows:

(i) $\quad\vdash\tau$

(ii) $\quad\vdash x$ if x among the variables in ξ

(iii) $\quad\vdash[x:\alpha]B$ if $\vdash\alpha$ and $x\ \mathrm{E}\ \alpha\vdash B$

(iv) $\vdash\{A\}B$ if $\vdash A$, $\vdash B$, $\mathsf{typ}(A) \geq \alpha$, $\mathsf{typ}^*(B) \geq [x{:}\alpha]C$ for some α, C.

2.1.3 So correct expressions are pretyped expressions satisfying the socalled *application condition*: in appl. expressions $\{A\}B$ the expression B has a domain (to compute from $\mathsf{typ}^*(B)$) corresponding with the typ of A. In the next section where we also introduce constant-expressions, an additional condition concerning instantiation will be imposed.

There are various alternative, equivalent, formulations of the application condition possible. E.g. one can replace "$\mathsf{typ}(A) \geq \alpha$" by "$\mathsf{typ}(A) \vdash \alpha$". In Λ (i.e. without η-reduction) we have CR, so it is even sufficient to require $\mathsf{typ}(A) = \alpha$ and $\mathsf{typ}^*(B) = [x{:}\alpha]C$, in other words: $\mathsf{typ}^*(B) = [x{:}\mathsf{typ}(A)]C$ - where $=$ is full definitional equality (see II.4.6-7, V.2.11) - or, anticipating certain results of sec.6.2.6, we might restrict the computation of the domain of B by requiring $\mathsf{typ}^*(B) \geq_\beta^1 [x{:}\alpha]C$ (compare V.3.3).

2.1.4 Since norms are preserved under taking typ and under reduction (see IV.3.4) the correct expressions are *strictly normable*. This can be shown by induction on the definition of $\vdash$. E.g. that $\{A\}B$ is strictly normable if it is correct: By ind. hyp. A and B are normable, so $\mu(A) \equiv \mu(\mathsf{typ}(A)) \equiv \mu(\alpha)$ and $\mu(B) \equiv \mu(\mathsf{typ}^*(B)) \equiv \mu([x{:}\alpha]C) \equiv [\mu(\alpha)]\mu(C)$, so $\{A\}B$ is normable, with $\mu(\{A\}B) \equiv \mu(C)$.

Hence the correct expressions are SN and the system is decidable.

2.2 Introducing constant-expressions; degree-norm correctness

2.2.1 We allowed the presence of a book containing schemes for the constants. Now we can simply introduce constant-expressions by adding the *instantiation* rule:

(v) If $\bar{y}\ E\ \bar{\beta} \star c(\bar{y})\ E\ \gamma$ is a scheme of B, $k = |\bar{y}|$, $\vdash B_1, \ldots, \vdash B_k$ and $\mathsf{typ}(B_1) \vdash \beta_1, \ldots, \mathsf{typ}(B_k) \vdash \beta_k[\![B]\!]$ then $\vdash c(\bar{B})$.

That is, in a constant-expression $c(\bar{B})$, the arguments B_i have to satisfy the *instantiation condition* $\mathsf{typ}(B_i) \vdash \beta_i[\![\bar{B}]\!]$.

However, we have to make sure that typ^* is still well-defined, particularly that taking typ still decreases the degree by one. E.g. $\mathsf{typ}(c(\bar{B}))$ $(\equiv \mathsf{typ}(c)[\![\bar{B}]\!] \equiv \gamma[\![\bar{B}]\!])$ and $\mathsf{typ}(c)(\equiv \gamma)$ must have the same degree.

2.2.2 Call a *substitution* $[\![\bar{y}/\bar{B}]\!]$ *degree correct* if degree(y_i) = degree(B_i) for i=1,...,$|\bar{y}|$. Degree correct substitutions preserve the degree:

If γ is a $\bar{y}$-expression and $[\![\bar{y}/\bar{B}]\!]$ is degree correct then $\gamma[\![\bar{B}]\!]$ and γ have the same degree. So, if we would add the requirement of degree correct substitution to the instantiation condition, then we might be satisfied. But this is not what we want: we rather would like to show that the instantiation condition *implies* the degree correctness of the substitution involved. This amounts to showing that degrees are preserved under reduction as well. To this end we introduce the concept of *degree-norm correctness*.

2.2.3 Degree-norms are defined by:

(i) positive integers are degree-norms

(ii) if $\nu 1$, $\nu 2$ are degree-norms then $[\nu 1]\nu 2$ is a degree-norm.

So, just like ordinary norms (IV.2.1) are built up from τ and square brackets, degree-norms are constructed from 1,2,... and square brackets.

For degree-norms ν we define the degree-norm $\nu+1$ as follows:

(i) if ν is an integer then $\nu+1$ is as usual

(ii) if $\nu \equiv [\nu 1]\nu 2$ then $\nu+1 :\equiv [\nu 1](\nu 2+1)$.

So $([[2]3]2) + 1 \equiv [[2]3]3$.

2.2.4 Now we define *degree-norm correctness* of books, contexts (w.r.t. a book) and expressions (w.r.t. book and context). It is implicitly intended that an expression is degree-norm correct (*dnc*), if its degree-norm (dn), w.r.t. book and context, is defined.

The definition of the latter runs as follows:

(i) $\mathsf{dn}(\tau) :\equiv 1$

(ii) $\mathsf{dn}(x) :\equiv \mathsf{dn}(\mathsf{typ}(x)) + 1$

(iii) $\mathsf{dn}([x:\alpha]B) :\equiv [\mathsf{dn}(\alpha) + 1]\mathsf{dn}(B)$

(iv) $\mathsf{dn}(\{A\}B) :\equiv$ if $\mathsf{dn}(B) \equiv [\mathsf{dn}(A)]\nu$ then ν

(v) $\mathsf{dn}(c(\bar{B})) :\equiv \mathsf{dn}(\mathsf{typ}(c)) + 1$, if $\mathsf{dn}(B_i) \equiv \mathsf{dn}(y_i)$ for i=1,...,$|\bar{y}|$, where $\bar{y}\ \mathsf{E}\ \bar{\beta} * c(\bar{y})\ \mathsf{E}\ \gamma$ is the scheme of c.

Here the notational conventions are just like those w.r.t. ordinary norms: we write dn instead of ξ-dn and e.g., clause (iii) would in full read like this:

(iii) $\quad \xi\text{-dn}([x{:}\alpha]B) :\equiv [(\xi\text{-dn}(\alpha))+1](\xi, x \in \alpha)\text{-dn}(B)$.

Further a context is dnc if all its type parts are so, and a book is dnc, if all the contexts and typ's of it are dnc.

2.2.5 A degree-norm ν can be translated into an ordinary norm ν^* by replacing all occurrences of numbers by τ. Notice that $(\nu+1)^* \equiv \nu^*$, so $\text{dn}(A)^* \equiv \mu(A)$. This shows that dnc-ness implies strict normability.

Further, degree(A) can also be constructed from dn(A), for dn(A) ends precisely in the degree of A.

We call a substitution $[\![\bar{y}/\bar{B}]\!]$ dnc if $\text{dn}(B_i) \equiv \text{dn}(y_i)$, for $i=1,\ldots,|\bar{y}|$. Clearly dnc substitutions are degree correct.

Degree-norm correctness is preserved under dnc substitutions: if $\bar{y} \in \bar{\beta}\vdash\gamma$, $k=|\bar{y}|, \vdash B_1, \ldots, \vdash B_k$, γ dnc and $[\![\bar{y}/\bar{B}]\!]$ dnc then

$$\text{dn}(\gamma) \equiv \text{dn}(\gamma[\![\bar{B}]\!])$$

Proof: By induction on the definition of dn(γ). □

This gives us the following corollaries:

(1) C dnc, degree(C)$\neq$1 $\Rightarrow$ typ(C)dnc, dn(typ(C))+1 $\equiv$ dn(C)

(2) C dnc, $C \geq D \Rightarrow D$ dnc, dn(D) $\equiv$ dn(C)

(3) C dnc, degree(C)$\neq$1 $\Rightarrow$ degree(typ(C))+1 = degree(C)

(4) C dnc, $C \geq D \Rightarrow$ degree(D) = degree(C).

So typ* is total on the dnc expressions and, since dnc-ness is clearly decidable, typ* is *well-defined* on all the expressions, in the sense of V.4.4.1.

2.2.6 Now we are able to show that correctness implies degree-norm correctness.

Proof: By induction on $\vdash$. E.g. let $\vdash A$, $\vdash B$, typ(A) $\geq \alpha$, typ*(B) $\geq [x{:}\alpha]C$. By ind. hyp. A and B are dnc (so typ (B) is indeed defined), so typ(A), α, typ(B), typ(typ(B)),...,typ*(B) and

$[x{:}\alpha]C$ are dnc as well. Now $\mathsf{dn}(\mathsf{typ}^*(B)) \equiv \mathsf{dn}([x{:}\alpha]C) \equiv [\mathsf{dn}(\alpha)+1]\mathsf{dn}(C) \equiv [\mathsf{dn}(\mathsf{typ}(A))+1]\mathsf{dn}(C) \equiv [\mathsf{dn}(A)]\mathsf{dn}(C)$, while $\mathsf{dn}(\mathsf{typ}^*(B))$ and $\mathsf{dn}(B)$ just differ as to their "end number" so $\mathsf{dn}(B) \equiv [\mathsf{dn}(A)]\nu$ for some ν. Hence $\{A\}B$ is dnc.

Or, let $\bar{y} \in \bar{\beta} * c(\bar{y}) \in \gamma$ be a scheme, let $\vdash B_1, \ldots, \vdash B_k$ (with $k=|\bar{y}|$) and let the B_i satisfy the instantiation condition: $\mathsf{typ}(B_i) \downarrow \beta_i[\![\bar{B}]\!]$. By ind. hyp. the B_i and the β_i are dnc. Now $\mathsf{dn}(B_1) \equiv \mathsf{dn}(\mathsf{typ}(B_1))+1 \equiv \mathsf{dn}(\beta_1)+1 \equiv \mathsf{dn}(y_1)$, so $[\![y_1/B_1]\!]$ is a dnc substitution. So $\mathsf{dn}(B_2) \equiv \mathsf{dn}(\mathsf{typ}(B_2))+1 \equiv \mathsf{dn}(\beta_2[\![B_1]\!])+1 \equiv \mathsf{dn}(\beta_2)+1 \equiv \mathsf{dn}(y_2)$. So $[\![y_1,y_2/B_1,B_2]\!]$ is dnc, etc. Hence $c(\bar{B})$ is dnc. □

So typ* is also total on the correct expressions, and correctness is well-defined. Further, the above proof shows that the system with constants is strictly normable as well, so (using SN) it is decidable.

2.3 Introducing definitional constants

2.3.1 After the formulation of instantiation and application condition, it will also be clear how the *compatibility condition* of def and typ for the formation of definitional constant schemes has to read:

$$\mathsf{typ}(\mathsf{def}(d)) \downarrow \mathsf{typ}(d), \text{ for definitional constants } d.$$

2.3.2 The scheme of a definitional constant d is defined to be dnc, if $\mathsf{dn}(\mathsf{def}(d)) \equiv \mathsf{dn}(\mathsf{typ}(d))+1$, and for the corresponding $d(\bar{B})$ we define

$$\mathsf{dn}(d(B)) :\equiv \mathsf{dn}(\mathsf{typ}(d))+1$$

provided $[\![\bar{y}/\bar{B}]\!]$ is dnc, where $\bar{y} \in \bar{\beta}$ is the context of the scheme.

So, still $\mathsf{dn}(d(B)) \equiv \mathsf{dn}(\mathsf{typ}(d) \equiv \mathsf{dn}(\mathsf{typ}(d)[\![\bar{B}]\!])+1 \equiv \mathsf{dn}(\mathsf{typ}(d(\bar{B})))+1$, and degree-norms remain preserved under reduction: $\mathsf{dn}(d(\bar{B})) \equiv \mathsf{dn}(\mathsf{typ}(d))+1 \equiv \mathsf{dn}(\mathsf{def}(d)) \equiv \mathsf{dn}(\mathsf{def}(d)[\![\bar{B}]\!])$. And, by induction on correctness, we can prove that correctness implies degree-norm correctness. E.g. let the scheme of d be correct, then $\vdash \mathsf{def}(d)$, so $\mathsf{def}(d)$ dnc, and $\mathsf{dn}(\mathsf{def}(d)) \equiv \mathsf{dn}(\mathsf{typ}(\mathsf{def}(d)))+1$, and $\vdash \mathsf{typ}(d)$ so $\mathsf{typ}(d)$ dnc, $\mathsf{dn}(\mathsf{typ}(d)) \equiv \mathsf{dn}(\mathsf{typ}(\mathsf{def}(d)))$ and $\mathsf{dn}(\mathsf{def}(d)) \equiv \mathsf{dn}(\mathsf{typ}(d))+1$, q.e.d.

VII.3 The closure proof for Λ

3.1 What to prove

The decidability of the Automath languages is one of the major aims of the language theory. By using an algorithmic definition we got the decidability of Λ and Λη, both with and without constants, directly from normalization (see 2.1.4 and 2.2.6). So one might wonder what else there is to prove.

First there are both Nederpelt's conjectures, the *Church-Rosser* property (CR) for Λη, and the *closure* property (CL). We define

$$\mathsf{CR}(A) : B \leq A \geq C \Rightarrow B \downarrow C$$

$$\mathsf{CL}(A) : \vdash A,\ A \geq B \Rightarrow \vdash B$$

A main lemma for β-CL (and δ-CL) is the *substitutivity of correctness*: substitution with correct expressions of the right types preserves correctness. Formally:

$$x \in \alpha \vdash B,\ \vdash A,\ \mathsf{typ}(A) \downarrow \alpha \Rightarrow \vdash B[\![x/A]\!]$$

Other properties which play an important role in the proof of CL, are *sound applicability* (SA), *preservation of* typ(PT), *of* typ*(P*T) and *of domain* (PD). We write

$$\mathsf{SA}(A):\ A \equiv \{B\}[x{:}C]D \Rightarrow \mathsf{typ}(B) \downarrow C$$

$$\mathsf{PT}(A):\ A \geq B \Rightarrow \mathsf{typ}(A) \downarrow \mathsf{typ}(B) \quad (\mathrm{degree}(A) \neq 1,\ \mathrm{degree}(B) \neq 1)$$

$$\mathsf{P^*T}(A):\ A \geq B \Rightarrow \mathsf{typ^*}(A) \downarrow \mathsf{typ^*}(B)$$

$$\mathsf{PD}(A):\ A \equiv [x{:}B]C,\ A \geq [x{:}D]E \Rightarrow B \downarrow D$$

The properties PT_1, CL_1, $\mathsf{P^*T}_1$ and PD_1 are the respective one-step variants of PT, CL, P*T and PD.

The above properties are not mere technicalities from the closure proof, but are also meaningful from the point of view of *interpretation*. E.g. SA is characteristic for the fact that the Aut-languages do not allow "proper inclusion" of type, and PT (resp. P*T) expresses the nice behaviour of typ (resp. typ*) w.r.t. definitional equivalence.

Further, these properties serve to establish the correspondence

between the present, algorithmic systems and the E-systems, and between the versions with and without constants (see 6.2, 6.3).

3.2 Some simple facts

3.2.1 Throughout this section VII.3 we just discuss Λ without constants. So we may assume CR, and PD(A) (for all A) and SA(A) (for correct A) are immediate.

By induction on $\vdash A$ one also proves easily that $\vdash A$ implies $\vdash \mathrm{typ}(A)$ (so $\vdash \mathrm{typ}(\mathrm{typ}(A)), \ldots, \vdash \mathrm{typ}^*(A)$). This is not easy any more for a system with constants. This property is called *correctness of types*.

3.2.2 As with the E-systems (see V.3.1), we prove CL from CL_1 by ind. on $\geq$. For the β-outside case of CL_1 we need substitutivity and SA. Previously substitutivity (i.e. the substitution theorem, V.2.9) was easy and SA was rather involved, but here SA is easy and substitutivity is quite complicated.

First some properties of substitution, which are valid already for pretyped expressions. Let A be a ξ-expression, let B be a $(\xi, x\ \mathsf{E}\ \alpha, \eta)$-expression. Let C^* denote $C[\![x/A]\!]$. Then

(1) $\mathrm{typ}(A) \downarrow \mathrm{typ}(x) \Rightarrow \mathrm{typ}(B^*) \downarrow \mathrm{typ}(B)^*$, i.e.,

written out in full,

$$\xi\text{-}\mathrm{typ}(A) \downarrow \alpha \Rightarrow (\xi, \eta^*)\text{-}\mathrm{typ}(B^*) \downarrow ((\xi, x\ \mathsf{E}\ \alpha, \eta)\text{-}\mathrm{typ}(B))^*$$

(2) $\mathrm{typ}^*(A) \downarrow \mathrm{typ}^*(x) \Rightarrow \mathrm{typ}^*(B^*) \downarrow \mathrm{typ}^*(B)^*$

Both facts are proved by ind. on the length of B. Notice that (1) and (2) are valid for each right monotonic, reflexive relation instead of $\downarrow$, so e.g. for $\geq$.

3.2.3 The problem with substitutivity is that the condition $\mathrm{typ}(A) \downarrow \alpha$ is clearly not sufficient. We would also like to know something about typ^*. In fact we have the following theorem (modified subst., for short SC):

Let $x\ \mathsf{E}\ \alpha$, $\eta \vdash B$, let $\vdash A$, $\mathrm{typ}(A) \downarrow \mathrm{typ}(x)$ and $\mathrm{typ}^*(A) \downarrow \mathrm{typ}^*(x)$. Let C^* denote $C[\![x/A]\!]$ again. Then $\eta^* \vdash B^*$.

Proof: By induction on $\vdash B$. E.g. the application case. Let $\vdash B_1$, $\vdash B_2$, $\mathsf{typ}(B_1) \geq \beta$, $\mathsf{typ}^*(B_2) \geq [y{:}\beta]C$. By ind. hyp. $\vdash B_1^*$ and $\vdash B_2^*$. By (1), (2) and CR $\mathsf{typ}(B_1^*) \downarrow \beta^*$ and $\mathsf{typ}^*(B_2^*) \downarrow [y{:}\beta^*]C^*$. So by CR again $\mathsf{typ}(B_1^*) \geq \gamma$, $\mathsf{typ}^*(B_2^*) \geq [y{:}\gamma]D$ for some γ, D. So $\vdash \{B_1^*\}B_2^*$.

3.2.4 Corollary:

$$x \in \alpha \vdash B,\ \vdash A,\ \mathsf{typ}(A) \downarrow \mathsf{typ}(x),\ \mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(x) \Rightarrow \vdash B[\![A]\!].$$

Another consequence of (1) is $\mathsf{PT}_1(A)$ for correct A, i.e.

$$\vdash A,\ A > B \Rightarrow \mathsf{typ}(A) \downarrow \mathsf{typ}(B)$$

Proof: Assume for definiteness that $>$ is disjoint one step reduction $\bar{>}_1$.

The proof is by induction on the length of A. For example:

(i) $A \equiv \{A_1\}[x{:}\alpha]A_2$, $B \equiv A_2[\![A_1]\!]$. By SA $\mathsf{typ}(A_1) \downarrow \alpha$ so by (1) above $\mathsf{typ}(A) \equiv \{A_1\}[x{:}\alpha]\mathsf{typ}(A_2) > \mathsf{typ}(A_2)[\![A_1]\!] \downarrow \mathsf{typ}(A_2[\![A_1]\!]) \equiv \mathsf{typ}(B)$.

(ii) $A \equiv \{A_1\}A_2$, $B \equiv \{B_1\}B_2$, $A_1 > B_1$, $A_2 > B_2$. By ind. hyp. $\mathsf{typ}(A) \equiv \{A_1\}\mathsf{typ}(A_2) \downarrow \{A_1\}\mathsf{typ}(B_2) > \{B_1\}\mathsf{typ}(B_2) \equiv \mathsf{typ}(B)$, so by CR we are done.

3.3 Heuristic considerations

3.3.1 At first sight SA, PT_1 and correctness of types seem to give a good starting position for proving CL. In a way this is true: we only have to find the right induction and the right induction hypothesis.

Let us first try to prove $\mathsf{CL}_1(A)$ by induction on the length of A, or rather by induction on the relation "being a subexpression of", for short: by induction *on subexpressions*. We interpret CL_1 in terms of disjoint one step reduction. For the appl. case of inside reduction the ind. hyp. is not strong enough, we additionally need $\mathsf{P^*T}_1$. So instead we try to prove CL_1 and $\mathsf{P^*T}_1$ together, again by induction on subexpressions. Now everything is allright with the inside reductions, but with outside β_1 we still come in trouble: $A \equiv \{A_1\}[x{:}\alpha]A_2$, SA gives $\mathsf{typ}(A_1) \downarrow \alpha$ but in view of the previous section we also want

$$\mathsf{typ}^*(A_1) \downarrow \mathsf{typ}^*(\alpha).$$

3.3.2 So let us see under what conditions we might *prove* this typ^*-requirement. First notice: if we knew CL already, then we could use PT_1 to prove PT (for correct expressions), e.g. by induction on $\geq$. The induction step runs as follows: let $\vdash A$, $A \geq B \geq C$. By CL we get $\vdash B$ and by ind. hyp. $\mathsf{typ}(A) \downarrow \mathsf{typ}(B) \downarrow \mathsf{typ}(C)$ whence by CR: $\mathsf{typ}(A) \downarrow \mathsf{typ}(C)$, q.e.d. An alternative proof of $\mathsf{PT}(A)$ *from* CL works by *induction on the reduction tree of* A (by virtue of $\mathsf{SN}(A)$), for short: by induction *on reducts*. Viz. let $\vdash A$, $A \geq C$. If $A \equiv C$ then $\mathsf{typ}(A) \equiv \mathsf{typ}(C)$. Otherwise for some B, $A >_1 B \geq C$. By PT_1 $\mathsf{typ}(A) \downarrow \mathsf{typ}(B)$, by CL $\vdash B$ and by ind. hyp. $\mathsf{typ}(B) \downarrow \mathsf{typ}(C)$, so by CR $\mathsf{typ}(A) \downarrow \mathsf{typ}(C)$.

3.3.3 Further from PT we can prove P*T, or rather:

$$\vdash A,\ \vdash B,\ A \downarrow B \Rightarrow \mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(B)$$

by induction on degree(A) + degree(B), as follows. If degree(A) = 1 then degree(B) = 1 too so $\mathsf{typ}^*(A) \equiv A \downarrow B \equiv \mathsf{typ}^*(B)$. Otherwise, degree$(B) \neq 1$ either, so we can apply PT to A and B. By CR we get $\mathsf{typ}(A) \downarrow \mathsf{typ}(B)$, by correctness of types $\vdash \mathsf{typ}(A)$, $\vdash \mathsf{typ}(B)$ so by the ind. hyp. $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(B)$, q.e.d. An alternative proof of P*T from CL and PT is by induction on $\rightarrow$, the order generated by (1) "being a proper reduct of", (2) "being the typ of" (as in V.). So the induction on $\rightarrow$ includes the induction on reducts mentioned before. That $\rightarrow$ is indeed well-founded will become clear in the sequel.
The proof looks like this. Let $\vdash A$, let $A \geq B$. By CL $\vdash B$ and by PT $\mathsf{typ}(A) \geq F \leq \mathsf{typ}(B)$. By correctness of types $\vdash \mathsf{typ}(A)$, $\vdash \mathsf{typ}(B)$ and by the ind. hyp. $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(F) \downarrow \mathsf{typ}^*(B)$, and by CR $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(B)$.

3.3.4 In section 3.2.2 we announced to prove CL from CL_1 by induction on $\geq$. However, this can be interpreted in two ways:

(1) to prove $\vdash A$, $A \geq B \Rightarrow \vdash B$, by induction on $A \geq B$, i.e. on the number of reduction steps between A and B,

(2) to prove $\mathsf{CL}(A)$ by induction on the reduction tree of A, i.e. by induction on reducts. Both inductions work, but the second one has an advantage: we just *need* $\mathsf{CL}_1(A)$, but can freely *use* $\mathsf{CL}(B)$ in the course of the proof, for each proper reduct A of B!

3.3.5 Now it becomes probably plausible to try and prove CL(A) directly by an induction on $\overset{*}{\to}$, the order generated by $\to$ (3.3.3) and by sub. In this way we combine the induction on subexpressions (3.3.1, for the "inside" cases of CL_1), on reducts (3.3.2, to prove PT), and on $\to$ (3.3.3, to prove P*T).

In order to make the induction work we need the well-foundedness of $\overset{*}{\to}$ on the correct expressions, i.e. the socalled *big tree theorem* BT.

Section 3.4 contains the proof of CL as sketched above, assuming BT, section 4 is devoted to the proof of BT.

3.4 The actual closure proof

3.4.1 Definition of $\to$

$\to$ is the reflexive and transitive relation generated by

(1) $A \to \mathsf{typ}(A)$

(2) $A \geq B \Rightarrow A \to B$

3.4.2 Definition of $\overset{*}{\to}$

$\overset{*}{\to}$ is the reflexive and transitive relation generated by

(1) $B \ \mathsf{sub}\ A \Rightarrow A \overset{*}{\to} B$

(2) $A \to B \Rightarrow A \overset{*}{\to} B$

3.4.3 The *big tree* of an expression A is the reduction tree of A w.r.t. the extended reduction relation $\overset{*}{\to}$. We assume the *big tree theorem* BT, which states that $\overset{*}{\to}$ is well-founded on the correct expressions (and, hence, that their big trees are finite).

3.4.4 Lemma: Let $\vdash A$, CL(A). Then PT(A) (degree(A) $\neq 1$)

Proof: As in 3.3.2, e.g. by ind. on reducts, using PT_1 and CR.

3.4.5.1 Define:

$$CL^+(A) : \Leftrightarrow A \to B \Rightarrow \vdash B$$

3.4.5.2 So $CL^+(A) \Rightarrow CL(A)$

3.4.6 Lemma: Let $\vdash A$, $CL^+(A)$. Then $P^*T(A)$.

Proof: By BT we can use induction on $\rightarrow$. Let $A \geq B$. If degree$(A) = 1$ then degree$(B) = 1$ too and there is nothing to prove. Otherwise, degree$(B) \neq 1$ either, so by the previous lemma $PT(A)$, i.e. $typ(A) \geq F \leq typ(B)$. By CL and correctness of types $\vdash typ(A)$, $\vdash typ(B)$ and by the ind. hyp. $typ^*(A) \downarrow typ^*(F) \downarrow typ^*(B)$. Now use CR.

3.4.7 Theorem: $\vdash A \Rightarrow CL(A)$

Proof: By BT we can use induction on $\overset{*}{\rightarrow}$. Let $\vdash A$, $A \geq B$. If $A \equiv B$ then there is nothing to prove. Otherwise $A > C \geq B$ with C a proper reduct of A. We want $\vdash C$. The interesting cases are:

(1) $A \equiv \{A_1\}A_2$, $C \equiv \{C_1\}C_2$, $\vdash A_1$, $typ(A_1) \geq \alpha$, $\vdash A_2$, $typ^*(A_2) \geq [x{:}\alpha]D$, $A_1 > C_1$, $A_2 > C_2$. By ind. hyp. $\vdash C_1$, $\vdash C_2$. By PT_1 $typ(A_1) \downarrow typ(C_1)$, so by CR $typ(C_1) \downarrow \alpha$. Now by the ind. hyp. we can assume $CL^+(A_2)$, so $P^*T(A_2)$ and $typ^*(A_2) \downarrow typ^*(C_2)$, and by CR $typ^*(C_2) \downarrow [x{:}\alpha]D$, q.e.d.

(2) $A \equiv \{A_1\}[x{:}\alpha]A_2$, $\vdash A_1$, $\vdash [x{:}\alpha]A_2$, $typ(A_1) \downarrow \alpha$. By ind. hyp. we can assume $CL^+(A_1)$, $CL^+(\alpha)$, so $typ^*(A_1) \downarrow typ^*(\alpha)$, and by substitutivity (3.2.4) $\vdash A_2[\![A_1]\!] \equiv C$, q.e.d.

VII.4 The Big Tree Theorem

4.1 Introduction

For the definition of the extended reduction relations $\rightarrow$ and $\overset{*}{\rightarrow}$ we refer to sec. 3.4. Both definitions make use of typ, so $\rightarrow$ and $\overset{*}{\rightarrow}$ are only defined on pretyped expressions, i.e. expressions with a context. Notice: taking subexpressions often requires extension of the context.

The big tree of an expression A is its reduction tree w.r.t. $\overset{*}{\rightarrow}$, i.e. the branches of the tree are the proper $\overset{*}{\rightarrow}$-reduction sequences of A.

We define:

$\mathsf{BT}(A)$: $\Leftrightarrow$ A has no infinite proper $\overset{*}{\to}$-reduction sequences

The big tree is infinitary so:

$\mathsf{BT}(A)$ $\Leftrightarrow$ the big tree of A is finite

In this section VII.4 we prove the *big tree theorem* BT:

(BT) $\quad A$ normable $\Rightarrow \mathsf{BT}(A)$.

So BT states that on the normable expressions $\overset{*}{\to}$ is well-founded, i.e. that $\overset{*}{\to}$-SN holds.

De Vrijer [70] introduced $\overset{*}{\to}$ and big trees, and proved BT for a system of normable expressions containing his language $\lambda\lambda$.

Below we give two different proofs of BT. The first (sec. 4.5) is modelled after the second proof of β-SN (IV.2.5), the second one (sec. 4.6) uses an idea from de Vrijer's proof (the "bookkeeping pairs") but further follows the first β-SN proof (IV.2.4.4). Actually both proofs deal with a modification $\geq_{\beta\tau}$ of $\overset{*}{\to}$ which is somewhat easier to handle and gives rise to even bigger trees (sec. 4.4.2).

For simplicity we start with a system without constants, and take just β-reduction for the ordinary reduction $\geq$ involved in $\to$ and $\overset{*}{\to}$. Later (5.2, 6.2, 6.3) BT will be extended to cover the remaining cases.

4.2 Heuristics 1

After de Vrijer we also call $\to$ and $\overset{*}{\to}$ *rt-reduction* and *rst-reduction* respectively, with r for ordinary *reduction*, s for *subexpression*, t for *type*. Similarly we speak about r-reduction (i.e. ordinary $\geq$), s-reduction (A s-reduces to its subexpression), t-reduction (A t-reduces to $\mathsf{typ}(A)$ etc.) and their combinations. The meaning of rs-SN, st-SN etc. and θ_{rs} - the length of rs-reduction tree of an rs-SN expression - etc. will be clear.

We want BT, i.e. rst-SN for the normable expressions. Let us summarize what SN-results we know already:

(1) r-SN. This is ordinary β-SN as proved in IV.2.4 for the normable expressions.

(2) s-SN and t-SN. s-reduction decreases length of expressions, t-reduction decreases degree of (pre-typed) expressions.

(3) rt-SN. This was proved for correct expressions in V.4.4. The same induction (1) on degree, (2) on θ_r, applies to all degree-norm correct expressions: taking typ decreases the degree, r-reduction preserves degree.

(4) rs-SN. Provable for the normable expressions by induction on (1) θ_r, (2) length of expression. In fact the induction used in the proof of the square brackets lemma SQBR (IV.2.4.3), and in several β-SN proofs as a subordinate induction (IV.2.4.4, IV.2.5.3) is just induction on the rs-reduction tree.

(5) st-SN. Can be proved by induction on the definition of pre-typed expressions (IV.3.2).

Clearly these inductions fail for full rst-SN: s-reduction can increase the degree, r-reduction generally increases length of expression, and taking typ can increase both length of expression and length of r-reduction tree. Besides, on the normable expressions r-reduction does not preserve the degree.

4.3.1 Norm properties

From IV.2.1 we recall some properties of the norm μ and of the normable expressions. We write $A <_\mu B$ for: $\mu(A)$ is shorter than $\mu(B)$.

(1) $\{A\}B$ normable $\Rightarrow \{A\}B <_\mu B$ and $A <_\mu B$

(2) A normable $\Rightarrow \mu(\mathsf{typ}(A)) \equiv \mu(A)$

(3) $\mu(x) \equiv \mu(A)$, B normable $\Rightarrow \mu(B[\![x/A]\!]) \equiv \mu(B)$

(4) $A \geq B$, A normable $\Rightarrow \mu(B) \equiv \mu(A)$

(5) $B \subset A$, A normable $\Rightarrow B$ normable

Properties (2), (4), (5) make that the normable expressions are closed under $\overset{*}{\to}$ and that $\to$ preserves the norm.

4.3.2 BT-conditions

Similarly to the SN-conditions in IV.2.4.1 we can formulate necessary and sufficient BT-conditions:

(1) $\mathsf{BT}(x) \Leftrightarrow \mathsf{BT}(\mathsf{typ}(x))$

(2) $\mathsf{BT}([y{:}B_1]B_2) \Leftrightarrow \mathsf{BT}(B_1),\ \mathsf{BT}(B_2)$

(3) $\mathsf{BT}(\{B_1\}B_2) \Leftrightarrow \mathsf{BT}(B_1),\ \mathsf{BT}(B_2)$ and $(B_2 \to [y{:}\beta]C \Rightarrow C[\![B_1]\!]\mathsf{BT})$

Proof: We just give the $\Leftarrow$-part of (3). Let $\mathsf{BT}(B_1)$, $\mathsf{BT}(B_2)$ and $B_2 \to [y{:}\beta]C \Rightarrow \mathsf{BT}(C[\![B_1]\!])$. B_2 is rst-SN so rt-SN so we can use $\theta_{rt}(B_2)$. B_1 is rst-SN so r-SN so we can use $\theta_r(B_1)$. Using induction on $\theta_r(B_1) + \theta_{rt}(B_2)$ we prove that all one-step rst-reducts of $\{B_1\}B_2$ are BT. Distinguish:

(i) D sub $\{B_1\}B_2$, so $D \subset B_1$ or $D \subset B_2$, so $\mathsf{BT}(D)$.

(ii) $B_2 >_{1,\beta} D$ or $D \equiv \mathsf{typ}(B_2)$. We have $\mathsf{BT}(B_1)$, $\mathsf{BT}(D)$ and $D \to [y{:}\beta]C \Rightarrow \mathsf{BT}(C[\![B_1]\!])$. Apply the ind. hyp. to $\{B_1\}D$, this gives $\mathsf{BT}(\{B_1\}D)$.

(iii) $B_1 >_{1,\beta} D$. Apply the ind. hyp. to $\{D\}B_2$.

(iv) $B_2 \equiv [y{:}\beta]C$. Then by assumption $\mathsf{BT}(C[\![B_1]\!])$.

4.3.3 Heuristics 2

If $\mathsf{BT}(B_2)$, $B_2 \to [y{:}\beta]C$ then clearly $\mathsf{BT}(C)$. So BT-condition (3) above suggests as a main step in proving BT the *substitution theorem for* BT: $\mathsf{BT}(A)$, $\mu(x) \equiv \mu(A)$, $\mathsf{BT}(B) \Rightarrow \mathsf{BT}(B[\![x/A]\!])$.

Indeed, if we knew this theorem, we could simply proceed by induction on pretyped expressions and get BT. The similarity with the situation around β-SN suggests us to use SQBR (IV.2.4.3), for $\to$ instead of $\geq$: If $B^* \to [y{:}\beta]C$ then either (1) $B \to [y{:}\beta_0]C_0$ with $\beta_0^* \geq \beta$, $C_0^* \to C$, or (2) $B \to \{\bar{F}\}x$, $(\{\bar{F}\}x)^* \to [y{:}\beta]C$, where * stands for $[\![x/A]\!]$.

However the following counter example shows that this lemma is wrong: Take $B \equiv \{B_1\}[z{:}\gamma][y{:}\beta]\{z\}x$, $A \equiv [u{:}\phi]\cdot\cdot u\cdot\cdot u$. Then $B^* \to [y{:}\beta^*[\![B_1^*]\!]]\cdot\cdot B_1^*\cdot\cdot\gamma^*$, but $B \to [y{:}\beta[\![B_1]\!]]\{B_1\}x$, and $(\{B_1\}x)^* \to \cdot\cdot B_1^*\cdot\cdot B_1^*$.

4.4 $\beta\tau$-reduction

4.4.1 One point which makes SQBR break down for $\to$ is that nót:

$$B \to C \Rightarrow B[\![x/A]\!] \to C[\![x/A]\!]$$

Example: $B \equiv x$, $C \equiv \mathsf{typ}(x)$ and the only connection between x and A concerns their norms (not their typ's).

The other substitution property: $A \to A' \Rightarrow B[\![A]\!] \to B[\![A']\!]$ does not hold either, due to the lack of monotonicity clauses in the definition of $\to$. Example: $A \to \mathsf{typ}(A)$ but not $\cdots A \cdots \to \cdots \mathsf{typ}(A) \cdots$.

4.4.2 Now we introduce $\beta\tau$-*reduction* by adding these monotonicity rules to the definition of $\to$. What we get is a reduction in the usual sense, that a one step reduction consists of replacing a subexpression (redex) by another expression (contractum). The redices are here of two kinds:

(1) β-redices which contract as usual

(2) τ-redices: variables x which contract according to $x >_\tau \mathsf{typ}(x)$.

We use the same terminology as before (II.7.1.2): $\geq_\tau$, $>_{1,\tau}$, $>_{\beta\tau}$ etc., τ-SN, $\beta\tau$-SN, $\theta_{\beta\tau}$ etc.

Now $\geq_{\beta\tau}$ satisfies the second substitution property (above) indeed but the first one is still not valid (same counter example).

Just like $\to$ and $\overset{*}{\to}$, $\geq_{\beta\tau}$ is only defined for pretyped expressions. Formally, we ought to speak about "$\geq_{\beta\tau}$ w.r.t. context ξ", and the monotonicity for abstr. expressions then would read:

If $B_1 >_{\beta\tau} C_1$ w.r.t. ξ and $B_2 >_{\beta\tau} C_2$ w.r.t. $(\xi, y \in B_1)$

then $[y{:}B_1]B_2 >_{\beta\tau} [y{:}C_1]C_2$ w.r.t. ξ

4.4.3 We are going to prove $\beta\tau$-SN and then conclude BT from the Theorem: $\beta\tau\text{-SN}(A) \Rightarrow \mathsf{BT}(A)$

Proof: Let $\beta\tau\text{-SN}(A)$. Using induction on (1) $\theta_{\beta\tau}(A)$, (2) length of A we show that all one-step rst-reducts of A are BT. So A itseld is BT.

4.4.4 $\beta\tau$-SN conditions

These are quite similar to the BT-conditions. The only non-trivial modification concerns the appl. case.

(3) $\beta\tau\text{-SN}(\{B_1\}B_2) \Leftrightarrow \beta\tau\text{-SN}(B_1)$, $\beta\tau\text{-SN}(B_2)$ and

$$B_2 \geq_{\beta\tau} y{:}\beta\ C \Rightarrow \beta\tau\text{-SN}(C[\![B_1]\!])$$

Proof: As in 4.2.3 but now we use induction on $\theta_{\beta\tau}(B_1) + \theta_{\beta\tau}(B_2)$.

4.4.5 Something on $\geq_\tau$

Just like st-SN (see 4.2(5)) we can prove τ-SN. Further we verify τ-CR: Let Σ contain subexpressions $\Delta \equiv [x{:}\alpha]\cdot\cdot x\cdot\cdot$, $\Gamma \equiv [y{:}\beta]\cdot\cdot y\cdot\cdot$. Then $\Delta >_\tau \Delta' \equiv [x{:}\alpha]\cdot\cdot\alpha\cdot\cdot$, $\Gamma >_\tau \Gamma' \equiv [y{:}\beta]\cdot\cdot\beta\cdot\cdot$ and we want a common τ-reduct of $\cdots\Delta'\cdot\cdot\Gamma\cdots$ and $\cdots\Delta\cdot\cdot\Gamma'\cdots$. As in II.8.2 we consider all the possible cases. Generally the reductions simply commute: $\cdots\Delta'\cdot\cdot\Gamma\cdots >_\tau \cdots\Delta'\cdot\cdot\Gamma'\cdots <_\tau \cdots\Delta\cdot\cdot\Gamma'\cdots$. In case the specific x occurs in β or the specific y occurs in α then two τ-steps are needed, e.g. $[y{:}\cdot\cdot x\cdot\cdot]\cdot\cdot y\cdot\cdot >_\tau [y{:}\cdot\cdot\alpha\cdot\cdot]\cdot\cdot y\cdot\cdot >_\tau [y{:}\cdot\cdot\alpha\cdot\cdot]\cdot\cdot(\cdot\cdot\alpha\cdot\cdot)\cdot\cdot <_\tau <_\tau [y{:}\cdot\cdot x\cdot\cdot]\cdot\cdot(\cdot\cdot x\cdot\cdot)\cdot\cdot$. Anyhow the weak diamond property holds for $>_\tau$, so by τ-SN we get τ-CR, and uniqueness of τ-normal form.

4.4.6 This gives an easy way of reaching a $\beta\tau$-normal form: first τ-normalize then β-normalize. Notice: the norm properties guarantee that $\geq_{\beta\tau}$ preserves the norm of normable expressions.

$\geq_\beta$ and $\geq_\tau$ do not commute, but we still can get $\beta\tau$-CR for the normable expressions, as follows. For norms ν we define a $\beta\tau$-normal expression ν^*: (1) $\tau^* \equiv \tau$, (2) $([\nu_1]\nu_2)^* \equiv [x{:}\nu_1^*]\nu_2^*$. Now we can prove

$$A \text{ normable} \Rightarrow A \geq_{\beta\tau} (\mu(A))^*$$

by ind. on the definition of μ. This gives $\beta\tau$-CR and uniqueness of $\beta\tau$-normal form. The procedure above assures the existence, so for normable A we can speak of $\beta\tau$-nf(A).

In fact ν^* is Nederpelt's original representation of the norm ν.

4.5 First proof of $\beta\tau$-SN; a correction to IV.2.5.3

4.5.1 In view of 4.4.4 it seems reasonable to concentrate on the *substitution theorem for* $\beta\tau$-SN: A $\beta\tau$-SN, B $\beta\tau$-SN, $\mu(x) = \mu(A) \Rightarrow B[\![A]\!]$ $\beta\tau$-SN. Just like with $\to$, SQBR fails for $\geq_{\beta\tau}$, so we rather let us inspire by the second proof of β-SN (IV.2.5.3).

In fact we also take the occasion to indicate (and repair) a flaw in that proof, concerning the distinction between *replacement* and *substitution*.

4.5.2 Replacement vs.substitution

When defining substitution (II.2.4) we have assumed the concept of literary *replacement* to be understood. Substitution amounts to replacement *with precautions*, viz. that no *clash of variables* takes place, and substitution can also be considered a special case of replacement.

Now let us see what went wrong in IV.2.5.3 (and also in (IV.2.6.2). Essentially we wanted to replace a specific subexpression Δ in Σ by another expression Δ', thus producing Σ'. We had the idea that this replacement of Δ with Δ' could be performed via substitution for a new "fresh" variable y, such that $\Sigma_0 \equiv \cdot\cdot y \cdot\cdot$, $\Sigma \equiv \Sigma_0[\![y/\Delta]\!]$, $\Sigma' \equiv \Sigma_0[\![y/\Delta']\!]$. However this is wrong: possible bound variables of Σ, which become free in Δ, can never get the appropriate bindings in $\Sigma_0[\![y/\Delta]\!]$.

What we need here is literary replacement (LR) of y with Δ and Δ' resp. We introduce a new notation: $B[\![x/A]\!]_{LR}$ is the result of literary replacing all free occurrences of x in B by A.

4.5.3 Below we follow the general idea of IV.2.5.3, but instead of using a substitution theorem for SN, we use the - stronger! - replacement theorem - as we ought to have done there (and in IV.2.6.2) too.

The easiest way is to use replacement with a *set of expressions*. Notation: $B\{\!\{x/\alpha\}\!\}_{LR}$, where α is a set of expressions, is the set of expressions which result from B by (literary) replacing all free x in B by an expression $A \in \alpha$, but possibly different A's for different occurrences of x (compare multiple substitution, in II.10).

4.5.4 The monotonicity of $\geq_{\beta\tau}$ makes the *replacement property* work:

$$A \geq_{\beta\tau} A' \Rightarrow B[\![A]\!]_{LR} \geq_{\beta\tau} B[\![A']\!]_{LR}$$

provided A has been put in the appropriate extended context.
We make this slightly more explicit. Let Δ be an occurrence of a subexpression in Σ. The *context of* Δ *in* Σ can be defined by induction on the length of Σ. Intuitively speaking, it consists of all the assumptions $x \in \alpha$, which one encounters (in the form of abstractors $[x:\alpha]$) when scanning Σ from "left to right" until one arrives at Δ. The crucial clause in the definition is of course: if ξ is the context of Δ in Σ_2 then $(x \in \Sigma_1, \xi)$ is the context of Δ in $[x:\Sigma_1]\Sigma_2$.

Now the context of A in the replacement property must provide all free variables of A with the same typing as they get when A is inserted in B. E.g. we can take (ξ,η_0) where ξ is the context of B and η_0 is the intersection (in the sense of context inclusion sub, cf.V.2.6) of all the η's which are the context of a free occurrence of x in B.

We define $\rho(A)$ to be the set of $\beta\tau$-reducts of A. Then, again if A has been put in the right context,

$$C \in B\{x/\rho(A)\}_{LR} \Rightarrow B\{x/\rho(A)\}_{LR} \geq_{\beta\tau} C$$

4.5.5 The other replacement property $B \geq_{\beta\tau} C \Rightarrow B^* \geq_{\beta\tau} C^*$, where * stands for $[\![x/A]\!]_{LR}$ is still not generally valid, but we have a restricted version. Lemma: If $A \geq_{\beta\tau} \mathrm{typ}(x)$ and $B \geq_{\beta\tau} C$ then $B^* \geq_{\beta\tau} C^*$.
Proof: Ind. on $\geq_{\beta\tau}$. E.g. if $B >_{1,\tau} C$, $B \equiv \cdots x \cdots x \cdots$, $C \equiv \cdots \mathrm{typ}(x) \cdots x \cdots$, then $B^* \equiv \cdots A \cdots A \cdots \geq_{\beta\tau} \cdots \mathrm{typ}(x) \cdots A \cdots \equiv C^*$.

Corollary: B^* $\beta\tau$-SN, $A \geq_{\beta\tau} \mathrm{typ}(x) \Rightarrow B$ $\beta\tau$-SN.
Proof: Use ind. on (1) $\theta_{\beta\tau}(B^*)$, (2) length of B^*. E.g. inspect the $\beta\tau$-SN conditions.

4.5.6 Now we are ready for the $\beta\tau$-SN proof.

Replacement theorem for $\beta\tau$-SN: Let * denote $\{x/\rho(A)\}_{LR}$.
Let B normable, $\mu(x) \equiv \mu(A)$, A, B $\beta\tau$-SN. Then

$$C \in B^* \Rightarrow C \ \beta\tau\text{-SN}$$

provided A has the right context.

Proof: By induction on (I) $\mu(A)$, (II) $\theta_{\beta\tau}(B)$, (III) the "*capacity*" of the transition from B to C, i.e. the sum of the $\theta_{\beta\tau}$'s of the reducts of A inserted in B. Now consider a single reduction step $C >_{1,\beta\tau} D$. We distinguish: (1) this reduction step concerns an old redex, i.e. a redex already present in B, (2) this step concerns a new redex. The latter are of two kinds: (2a) multiplied redices, i.e. redices inside an inserted reduct of A, (2b) newly composed redices. All τ-redices fall under case (1) or (2a) and the β-redices are classified as before, so the only possibility of case (2b) is as follows: $B \equiv \cdots x \cdots \{B_1\}x \cdots$, $C \equiv \cdots A_1 \cdots \{C_1\}[y{:}\gamma]E \cdots$, $D \equiv \cdots A_1 \cdots E[\![C_1]\!] \cdots$, where $C_1 \in B_1^*$, $A \geq_{\beta\tau} A_1$, $A \geq_{\beta\tau} [y{:}\gamma]E$.

In case (1) and (2a) the replacement and the reduction commute, i.e. $B > D_0$, $D \in D_0^*$. To be precise, let $\{C_1\}[y{:}\gamma]C_2$ be an "old" redex, i.e. $\{B_1\}[y{:}\beta]B_2 \subset B$, $C_1 \in B_1^*$, $C_2 \in B_2^*$. Then $D \equiv \cdots C_2[\![C_1]\!]\cdots \in (\cdots B_2[\![B_1]\!]\cdots)\{x/\rho(A[\![B_1]\!])\}_{LR}$, and not simply $D \in D_0^*$. Then we get $\beta\tau$-SN(D) by ind. hyp. II (case (1)) or III (case (2a)).

Now we tackle case (2b): create a new variable z and form B_0 by replacing the intended $\{B_1\}x$ by z. So $B \equiv B_0[\![z/\{B_1\}x]\!]_{LR}$. For simplicity we put $\mathsf{typ}(z) \equiv \beta\tau\text{-nf}(\{B_1\}x)$, so $\mu(z) \equiv \mu(\{B_1\}x)$ and $\beta\tau$-SN(B_0) - by 4.5.5. Then we form $B_0' \in B_0^*$ by replacing the remaining free x's of B_0 with the appropriate reducts of A, i.e. the same as used in the formation of C, and finally replace the z of B_0' by $E[\![C_1]\!]$. This gives us $D \equiv B_0'[\![z/E[\![C_1]\!]]\!]_{LR}$ back. Informally: $B_0 \equiv \cdots x\cdots z\cdots$, $B_0' \equiv \cdots A_1\cdots z\cdots$, $D \equiv \cdots A_1\cdots E[\![C_1]\!]\cdots$. Either by ind. hyp. II or III we get $\beta\tau$-SN(C_1). Further $\beta\tau$-SN(A) so $\beta\tau$-SN($[y{:}\gamma]E$) so $\beta\tau$-SN(E). By normability $B_1 <_\mu x$ so $C_1 <_\mu x$. Substitution is a special case of replacement, and replacement $[\![\ ]\!]_{LR}$ is a special case of $\{\ \}_{LR}$ so by the first ind. hyp. $\beta\tau$-SN($E[\![C_1]\!]$). B_0' is $\beta\tau$-SN by ind. hyp. II or III, $E[\![C_1]\!] <_\mu x$ so by ind. hyp. I again $\beta\tau$-SN(D) q.e.d.

4.5.7 Corollary 1: B normable, $\mu(x) \equiv \mu(A)$, A, B $\beta\tau$-SN $\Rightarrow$ $B[\![A]\!]\beta\tau$-SN
(substitution theorem for $\beta\tau$-SN)

Corollary 2: B normable $\Rightarrow$ B $\beta\tau$-SN (see 4.4.4)

Corollary 3: B normable $\Rightarrow$ BT(B) (as in 4.4.3)

4.6 Second proof of $\beta\tau$-SN

4.6.1 Bookkeeping pairs, τ-expansion and π-reduction

4.6.1.1 Assume that $A \geq_\tau B$, i.e. B results from A by successively replacing variables x by their type $\mathsf{typ}(x)$. Alternatively we can work backwards from τ-nf(A), by successively replacing newly created subexpressions by the original variable.

In general it is of course not possible to retrace which subexpressions are newly created, and from which variable they stem, unless we store this information somewhere inside the expression!

Following de Vrijer [70] we use a new *pairing operation* $\ulcorner \cdots, \cdots \urcorner$ for this kind of bookkeeping.

Definitions: (1) If A, B are expressions then $\ulcorner A,B \urcorner$ is an expression.

(2) If A, B are ξ-expressions then $\ulcorner A,B \urcorner$ is a ξ-expression.

(3) If A, B are normable, $\mu(A) \equiv \mu(B)$ then $\mu(\ulcorner A,B \urcorner) \equiv \mu(A)$.

For the rest the definitions of pretyped and normable expressions are unaltered. The notions of subexpression and substitution are extended in a straightforward way. As a new monotonicity rule, for each kind of reduction, we can have, e.g. $A > A'$, $B > B' \Rightarrow \ulcorner A,B \urcorner > \ulcorner A',B' \urcorner$.

4.6.1.2 Now the alternative way of producing B from A (above) can be described as follows: (1) first provide all variables x successively with a copy of their type, i.e. replace x by $\ulcorner x,\mathsf{typ}(x) \urcorner$ and so on, (2) then for some of these pairs simple restore the lefthand part, and for the rest pick the righthand part.

In the process (1) the τ-*expansion of* A, $\tau\text{-}\mathsf{exp}(A)$, is constructed, i.e. each x of A is replaced by $\ulcorner x,\tau\text{-}\mathsf{exp}(\mathsf{typ}(x)) \urcorner$. The process (2) we describe in terms of a *projection reduction* (π-reduction $\geq_\pi$).

Definitions: (1) The τ-exp of pretyped expressions is defined inductively:

(i) $\tau\text{-}\mathsf{exp}(x) \equiv \ulcorner x,\tau\text{-}\mathsf{exp}(\mathsf{typ}(x)) \urcorner$

(ii) $\tau\text{-}\mathsf{exp}(\{A\}B) \equiv \{\tau\text{-}\mathsf{exp}(A)\}\tau\text{-}\mathsf{exp}(B)$

(iii) $\tau\text{-}\mathsf{exp}([x:\alpha]B) \equiv [x:\tau\text{-}\mathsf{exp}(\alpha)]\tau\text{-}\mathsf{exp}(B)$

(iv) $\tau\text{-}\mathsf{exp}(\ulcorner A,B \urcorner) \equiv \ulcorner \tau\text{-}\mathsf{exp}(A),\tau\text{-}\mathsf{exp}(B) \urcorner$

(2) (i) one-step π-reduction $>_{1,\pi}$ is generated from π-contraction: $\ulcorner A,B \urcorner >_{1,\pi} A$, $\ulcorner A,B \urcorner >_{1,\pi} B$ by the monotonicity rules

(ii) π-reduction $\geq_\pi$ is the transitive and reflexive closure of $>_{1,\pi}$.

4.6.1.3 Remark: Formally we should have defined the τ-expansion of expressions w.r.t. their context, notation $\xi\text{-}\tau\text{-}\mathsf{exp}(B)$. The abstr. case of the definition then becomes:

$$\xi\text{-}\tau\text{-}\mathsf{exp}([x:\alpha]B) \equiv [x:(\xi\text{-}\tau\text{-}\mathsf{exp}(\alpha))]((\xi,x \in \alpha)\text{-}\tau\text{-}\mathsf{exp}(B))$$

4.6.1.4 The point of this alternative approach of $\geq_\tau$, making use of

$$A \geq_\tau B \Rightarrow \tau\text{-exp}(A) \geq_\pi B \quad \text{(see 6.2.2)}$$

is that $\geq_\pi$ is definitely easier to handle than $\geq_\tau$, roughly because $\geq_\pi$ does *not* depend on the context, and that $\geq_{\beta\tau}$-reductions of an expression can be simulated by $\geq_{\beta\pi}$-reductions of its τ-expansion.

Our proof below consists of two parts: first we show that $\beta\pi$-SN implies $\beta\tau$-SN, then we prove the SQBR lemma for $\geq_{\beta\pi}$ and $\beta\pi$-SN.

4.6.2 $\beta\pi$-SN implies $\beta\tau$-SN

4.6.2.1 Lemma: $A >_{1,\tau} B \Rightarrow \tau\text{-exp}(A) \geq_\pi \tau\text{-exp}(B)$ (in fact $\bar{>}_{1,\pi}$)

Proof: Ind. on $>_{1,\tau}$:

(i) τ-contraction, $A \equiv x$, $B \equiv \text{typ}(x)$. Then $\tau\text{-exp}(A) \equiv \ulcorner x, \tau\text{-exp}(\text{typ}(x))\urcorner >_{1,\pi} \tau\text{-exp}(\text{typ}(x)) \equiv \tau\text{-exp}(B)$

(ii) Monotonicity, e.g. $A \equiv [x{:}A_1]x$, $B \equiv [x{:}B_1]x$, $A_1 >_{1,\tau} B_1$: By ind.hyp. $\tau\text{-exp}(A_1) \geq_\pi \tau\text{-exp}(B_1)$, so $\tau\text{-exp}(A) \equiv [x{:}\tau\text{-exp}(A_1)]\ulcorner x, \tau\text{-exp}(A_1)\urcorner \geq_\pi [x{:}\tau\text{-exp}(B_1)]\ulcorner x, \tau\text{-exp}(B_1)\urcorner \equiv \tau\text{-exp}(B)$

4.6.2.2 Corollary 1: $A \geq_\tau B \Rightarrow \tau\text{-exp}(A) \geq_\pi \tau\text{-exp}(B)$

Corollary 2: $A \geq_\tau B \Rightarrow \tau\text{-exp}(A) \geq_\pi B$ (because $\tau\text{-exp}(B) \geq_\pi B$)

4.6.2.3 Lemma: Let A be a ξ-expression, let B be a $(\xi, x \in \alpha, \eta)$-expression. Let $^{\text{I}}$ and $^{\text{II}}$ stand for $[\![x/A]\!]$ and $[\![x/\tau\text{-exp}(A)]\!]$ resp. Then

$$\tau\text{-exp}(B)^{\text{II}} \geq_\pi \tau\text{-exp}(B^{\text{I}})$$

with $\tau\text{-exp}(B^{\text{I}})$ taken w.r.t. ξ, η^{I}.

Proof: ind. on the definition of $\tau\text{-exp}(B)$:

(i) $\tau\text{-exp}(x)^{\text{II}} \equiv \ulcorner x, \tau\text{-exp}(\alpha)\urcorner^{\text{II}} \equiv \ulcorner \tau\text{-exp}(A), \tau\text{-exp}(\alpha)\urcorner >_\pi \tau\text{-exp}(A) \equiv \tau\text{-exp}(x^{\text{I}})$.

(ii) $\tau\text{-exp}(y)^{\text{II}} \equiv \ulcorner y, \tau\text{-exp}(\text{typ}(y))^{\text{II}}\urcorner \geq_\pi \ulcorner y, \tau\text{-exp}(\text{typ}(y)^{\text{I}})\urcorner \equiv \tau\text{-exp}(y^{\text{I}})$.

(iii) $(\tau\text{-exp}(\{B_1\}B_2))^{II} \equiv (\{\tau\text{-exp}(B_1)\}\tau\text{-exp}(B_2))^{II} \geq_\pi$
$\{\tau\text{-exp}(B_1^I)\}\tau\text{-exp}(B_2^I) \equiv \tau\text{-exp}((\{B_1\}B_2)^I)$ etc.

4.6.2.4 Corollary: Let A be a ξ-expression, B is a $(\xi, x \in \alpha)$-expression.
Then $\tau\text{-exp}(B)[\![x/\tau\text{-exp}(A)]\!] \geq_\pi \tau\text{-exp}(B[\![x/A]\!])$

4.6.2.5 Corollary: $A \bar{>}_{1,\beta} B \Rightarrow \tau\text{-exp}(A) \bar{>}_{1,\beta} \geq_\pi \tau\text{-exp}(B)$

Proof: Ind. on $\bar{>}_{1,\beta}$:

(i) β-contraction, $A \equiv \{A_1\}[x:A_2]A_3$, $B \equiv A_3[\![A_1]\!]$,
$\tau\text{-exp}(A) \bar{>}_{1,\beta} \tau\text{-exp}(A_3)[\![x/\tau\text{-exp}(A_1)]\!] \geq_\pi \tau\text{-exp}(A_3[\![A_1]\!]) \equiv$
$\tau\text{-exp}(B)$, by 4.6.2.4.

(ii) monotonicity, e.g. $A \equiv \ulcorner A_1, A_2 \urcorner$, $B \equiv \ulcorner B_1, B_2 \urcorner$, $A_1 \bar{>}_{1,\beta} B_1$,
$A_2 \bar{>}_{1,\beta} B_2$. By ind. hyp. $\tau\text{-exp}(A) \equiv \ulcorner \tau\text{-exp}(A_1), \tau\text{-exp}(A_2) \urcorner$
$\bar{>}_{1,\beta} \geq_\pi \ulcorner \tau\text{-exp}(B_1), \tau\text{-exp}(B_2) \urcorner \equiv \tau\text{-exp}(B)$.

4.6.2.6 Theorem: $\tau\text{-exp}(A)\beta\tau\text{-SN} \Rightarrow A\ \beta\tau\text{-SN}$

Proof: Let $\tau\text{-exp}(A)$ be $\beta\pi$-SN, use ind. on $\theta_{\beta\pi}(\tau\text{-exp}(A))$. If $A >_{1,\beta} B$
then $\tau\text{-exp}(A) \bar{>}_{1,\beta} \geq_\pi \tau\text{-exp}(B)$ (by 4.6.2.5), so by ind. hyp.
$\beta\tau\text{-SN}(B)$.
Similarly, if $A >_{1,\tau} B$ then $\beta\tau\text{-SN}(B)$. So A is $\beta\pi$-SN.

4.6.3 The proof of $\beta\pi$-SN

4.6.3.1 The normable expressions are closed (and norms are preserved) under $\geq_{\beta\pi}$. Further $\geq_\pi$ satisfies both substitution properties (see 4.4.1). Notice that $\geq_\pi$ does not satisfy CR but that β and π commute (use nested one step reduction $\tilde{>}_{1,\pi}$, see II.3.4) and that weak $\pi\beta$-postponement holds:
$A \geq_{\beta\pi} B \Rightarrow A \geq_\pi \geq_\beta C \leq_\pi B$

4.6.3.2 $\beta\pi$-SN conditions

These are again quite similar to the β-SN conditions. The interesting clauses are:

(1) A $\beta\pi$-SN, B $\beta\pi$-SN $\Rightarrow [x:A]B$ and $\ulcorner A, B \urcorner$ $\beta\pi$-SN

(2) A $\beta\pi$-SN, B $\beta\pi$-SN and $(B \geq_{\beta\pi} [x{:}\alpha]D \Rightarrow D[\![A]\!]$ $\beta\pi$-SN) $\Rightarrow$ $\{A\}B$ $\beta\pi$-SN

So, again, we want the substitution theorem for $\beta\tau$-SN.

4.6.3.3 Square brackets lemma for $\geq_{\beta\pi}$: Let B be $\beta\pi$-SN. Let * stand for $[\![x/A]\!]$. Let $B^* \geq_{\beta\pi} [y{:}\beta]C$. Then either (1) $B \geq_{\beta\pi} [y{:}\beta_0]C_0$ with $\beta_0^* \geq_{\beta\pi} \beta$, $C_0^* \geq_{\beta\pi} C$, or (2) $B \geq_{\beta\pi} \{B_k\}\cdots\{B_1\}x$, $(\{\bar{B}\}x)^* \geq [y{:}\beta]C$.

Proof: As in IV.2.4.3, by induction on (I) $\theta_{\beta\pi}(B)$, (II) the length of B. The new case is $\ulcorner B_1,B_2\urcorner$, $B^* \equiv \ulcorner B_1^*,B_2^*\urcorner$. Then either $B_1^* \geq_{\beta\pi} [y{:}\beta]C$ or $B_2^* \geq_{\beta\pi} [y{:}\beta]C$, and we can apply ind. hyp. I to B_1 or B_2.

Remark: An alternative proof is provided by Barendregt's lemma, which is still valid for $\geq_{\beta\pi}$ (see II.11.3.5).

4.6.3.4 Substitution for $\beta\pi$-SN: Let B be normable, $\mu(x) \equiv \mu(A)$, A and B are $\beta\pi$-SN. Let * stand for $[\![x/A]\!]$. Then B^* $\beta\pi$-SN.

Proof: As in IV.2.4.4, by ind. on (I) $\mu(A)$, (II) $\theta_{\beta\pi}(B)$, (III) length of B. The new case concerns $B \equiv \ulcorner B_1,B_2\urcorner$, $B^* \equiv \ulcorner B_1^*,B_2^*\urcorner$. Both B_1^* and B_2^* are $\beta\pi$-SN by ind. hyp. II so B^* is $\beta\pi$-SN.

4.6.3.5 Corollary: B normable $\Rightarrow$ B $\beta\pi$-SN

4.6.3.6 Notice that the τ-expansion of normable A is again normable, so A normable $\Rightarrow$ τ-exp(A) normable.

Corollary: A normable $\Rightarrow$ A $\beta\tau$-SN (by 6.2.6)

Corollary: BT

VII.5 Closure and Church-Rosser for $\Lambda\eta$

5.1 Introduction

5.1.1 Here we consider the constant-less part of $\Lambda\eta$, defined as in sec. 2.12, but with $\geq$ standing for $\beta\eta$-reduction. It is easy to derive a strengthening rule (sec. V.1.6) for such an algorithmic system, so η-CL does not cause major difficulties. The problems with closure for $\Lambda\eta$, as compared to Λ, are rather due to the fact that CL and CR appear to be

heavily interwoven. Namely, a proof of CL (see, e.g., VII.3) seems to make quite essential use of CR, while in turn we seem to need CL in the course of the CR-proof - because βη-CR holds for correct expressions only.

The solution is of course to prove CR and CL (and a number of other properties) simultaneously, by induction on big trees. In sec. 5.2, below we prove indeed that BT extends to the present situation.

5.1.2 We introduce some notation that enables us to make the structure of the proof more explicit. Here $\overset{*}{\to}$ is as in VII.3.4.

Definition: If P is a property of expressions then P^* and P_0^* are given by

(1) $P^*(A) :\Leftrightarrow A \overset{*}{\to} B \Rightarrow P(B)$

(2) $P_0^*(A) :\Leftrightarrow (A$ properly $\overset{*}{\to}$-reduces to $B) \Rightarrow P(B)$

Using this notation, we can express our induction step by

$$\vdash A,\ CR_0^*(A),\ CL_0^*(A) \Rightarrow CR(A),\ CL(A)$$

for which, of course, it is sufficient to prove

$$\vdash A,\ CR_0^*(A),\ CL_0^*(A) \Rightarrow CR_1(A),\ CL_1(A)$$

The properties SA, PD, PT and P*T from 3.1 play again a role in the proof, and further property SC, substitutivity of correctness, here defined by $SC(B) :\Leftrightarrow$
$(x\ E\ \alpha \vdash B,\ \vdash A,\ typ(A) \downarrow typ(x),\ typ^*(A) \downarrow typ^*(x) \Rightarrow \vdash B[\![A]\!])$.

5.1.3 Now the proof below is organized as follows. First we present some preliminary facts, among which βη-BT (sec. 5.2), strengthening and η-PT (sec. 5.3).

Section 5.4 contains the actual closure proof. First we assume $\vdash A$, $CR_0^*(A)$, $CL_0^*(A)$, and prove $SA(A)$ and $PD(A)$ (in sec. 5.4.1), $PT_1(A)$, $SC(A)$ and $CR_1(A)$ (in sec. 5.4.2-5.4.4) respectively by a separate induction on big trees, and by simple induction on length. Then we complete the proof by proving $PT(A)$, $P^*T(A)$ and $CL_1(A)$ simultaneously, by induction on the big tree of A again.

5.2 Extension of BT to the βη-case

5.2.1 A postponement result

Let $\geq_{\tau\eta}$ and $\geq_{\beta\tau\eta}$ be the straightforward extensions of $\geq_{\tau}$ and $\geq_{\beta\tau}$, as defined in 4.4.2. Mere verification shows that

$$A \text{ pretyped}, A >_{1,\eta} >_{1,\tau} B \Rightarrow A >_{1,\tau} \bar{>}_{1,\eta} B$$

whence - as in II.7.3.2- τη-postponement:

$$A \text{ pretyped}, A \geq_{\eta\tau} B \Rightarrow A \geq_{\tau} \geq_{\eta} B.$$

Combining this with βη-pp we get

$$A \text{ pretyped}, A \geq_{\beta\tau\eta} B \Rightarrow A \geq_{\beta\tau} \geq_{\eta} B.$$

5.2.2 βητ-SN and βη-BT

In 4.6.3 we proved βτ-SN, which - as in II.7.3.5 - together with (βτ)-η-pp and η-SN gives us βητ-SN, for normable expressions. Then βη-BT follows, as in 4.4.3.

5.3 Some simple facts

5.3.1 Strengthening

If B is a $(\xi, x \mathrel{E} \alpha, \bar{y} \mathrel{E} \bar{\beta})$-expression, but $x \notin \mathrm{FV}(\bar{\beta})$ and $x \notin \mathrm{FV}(B)$, then B is a $(\xi, \bar{y} \mathrel{E} \bar{\beta})$-expression as well, and the typ (if degree$(B) \neq 1$) and typ* of B w.r.t. both contexts are syntactically equal ($\equiv$).

So, by induction on the definition of correctness, we get strengthening: if $x \mathrel{E} \alpha$, $\bar{y} \mathrel{E} \bar{\beta} \vdash (B)$, $x \notin \mathrm{FV}(\bar{\beta})$ (and $x \notin \mathrm{FV}(B)$) then $\bar{y} \mathrel{E} \bar{\beta} \vdash (B)$ - read this twice, with and without the parts concerning B - .

As a corollary we have: $x \mathrel{E} \alpha \vdash A$, $x \notin \mathrm{FV}(A) \Rightarrow \vdash A$

whence η-outside-CL_1: $\vdash x{:}\alpha\ \{x\}A$, $x \notin \mathrm{FV}(A) \Rightarrow \vdash A$.

5.3.2 η-PT and η-P*T

For pretyped A there holds

$$A >_{\eta} B \quad \mathrm{typ}(A) >_{\eta} \mathrm{typ}(B) \text{ (if degree}(A) \neq 1), \ \mathrm{typ}^*(A) >_{\eta} \mathrm{typ}^*(B)$$

Proof: Induction on the length of A.

So, induction on $\geq_\eta$ gives

$A \geq_\eta B \Rightarrow \mathsf{typ}(A) \geq_\eta \mathsf{typ}(B)$ (if $\mathsf{degree}(A) \neq 1$), $\mathsf{typ}^*(A) \geq_\eta \mathsf{typ}^*(B)$

and, a fortiori, we have η-PT and η-P*T

$A \geq_\eta B \Rightarrow \mathsf{typ}(A) \downarrow \mathsf{typ}(B)$ (if $\mathsf{degree}(A) \neq 1$), $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(B)$

5.3.3 From 3.2.1 we recall the property of correctness of types

$$\vdash A \Rightarrow \vdash \mathsf{typ}(A)$$

and the substitution properties from 3.2.2

(1) $\mathsf{typ}(A) \downarrow \mathsf{typ}(x) \Rightarrow \mathsf{typ}(B[\![A]\!]) \downarrow \mathsf{typ}(B)[\![A]\!]$

(2) $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(x) \Rightarrow \mathsf{typ}^*(B[\![A]\!]) \downarrow \mathsf{typ}^*(B)[\![A]\!]$

5.3.4 Property: Let $\mathsf{degree}(A) = 1$, $\mu(A) \equiv [\nu_1]\cdots[\nu_k]\varepsilon$. Then $A \geq [x_1{:}\alpha_1]\cdots[x_k{:}\alpha_k]C$.

Proof: Induction on the length of A. E.g. let $A \equiv \{A_1\}A_2$, then

$\mu(A_2) \equiv [\mu(A_1)][\nu_1]\cdots[\nu_k]\varepsilon$, so by ind. hyp.

$A_2 \geq [x{:}\beta][x_1{:}\alpha_1]\cdots[x_k{:}\alpha_k]C$ and $A \geq [x_1{:}\alpha_1']\cdots[x_k{:}\alpha_k']C'$, q.e.d.

Corollary: $\mathsf{Degree}(A) = 1$, $\mu(A) \equiv [\nu_1]\nu_2 \Rightarrow A \geq [x{:}\alpha]C$.

Corollary: $\vdash^1 A$, $A \equiv [x{:}\alpha]C$, $A \geq F \Rightarrow F \geq [x{:}\beta]D$

Proof: If A correct, then A normable, so F normable, with

$\mu(F) \equiv \mu(A) \equiv [\mu(\alpha)]\mu(C)$.

Corollary: $\vdash^1 A$, $A \equiv [x{:}\alpha]C$, $A \downarrow F \Rightarrow F \geq [x{:}\beta]D$.

5.4 The actual closure proof

5.4.1 Lemma: Let $\vdash A$, $\mathsf{CR}_0(A)$, $\mathsf{CL}_0(A)$. Then $\mathsf{PD}(A)$ and $\mathsf{SA}(A)$

Proof: By induction on the big tree of A.

(PD). Let $A \equiv [x{:}A_1]A_2$, $A \geq [x{:}B_1]B_2$. If $A_1 \geq B_1$, $A_2 \geq B_2$ then certainly $A_1 \downarrow B_1$. Otherwise $A_2 \geq \{x\}[x{:}B_1]B_2$. The latter expression is correct, satisfies CR* and CL*, so we can use SA and get $A_1 \downarrow B_1$, q.e.d.

(SA). Let $A \equiv \{A_1\}[x{:}A_2]A_3$. Then $\vdash A_1$, $\mathsf{typ}(A_1) \geq \phi$, $\vdash [x{:}A_2]A_3$,

$\mathsf{typ}^*([x{:}A_2]A_3) \equiv [x{:}A_2]\mathsf{typ}^*(A_3) \geq [x{:}\phi]C$. By correctness of types $\vdash[x{:}A_2]\mathsf{typ}^*(A_3)$, which also satisfies CR^* and CL^* so we can apply PD and get $A_2 \downarrow \phi$, whence $\mathsf{typ}(A_1) \downarrow A_2$, q.e.d.

5.4.2 Lemma: Let $\vdash A$, $\mathsf{CR}_0^*(A)$, $\mathsf{CL}_0^*(A)$. Then $\mathsf{PT}_1(A)$

Proof: Induction on length(A). η-PT_1 we know already (sec. 5.3.2). For β-outside-PT_1 let $A \equiv \{A_1\}[x{:}A_2]A_3$. By 5.4.1 $\mathsf{typ}(A_1) \downarrow A_2$ and by the substitution property 5.3.3.(1) $\mathsf{typ}(A) \equiv \{A_1\}[x{:}A_2]\mathsf{typ}(A_3) > \mathsf{typ}(A_3)[\![A_1]\!] \downarrow \mathsf{typ}(A_3[\![A_1]\!])$, q.e.d. The other cases are immediate.

5.4.3 Lemma: Let $x \in \alpha$, $\bar{y} \in \bar{\beta}\vdash B$, $\mathsf{CR}_0^*(B)$, $\mathsf{CL}_0^*(B)$, $\vdash A$, $\mathsf{typ}(A) \downarrow \alpha$, $\mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(\alpha)$. We write * for $[\![x/A]\!]$. Then $(\mathsf{SC}(B))$ $\bar{y} \in \bar{\beta}^*\vdash B^*$.

Proof: Induction on length(B). The crucial case is: $B \equiv \{B_1\}B_2$, $\mathsf{typ}(B_1) \geq \phi$, $\mathsf{typ}^*(B_2) \geq [u{:}\phi]\psi$. By ind. hyp. $\vdash B_1$, $\vdash B_2$. We do not know CR or CL for the substitution results, so we use a trick. Distinguish:

(1) B_1 does not end in x, then $\mathsf{typ}(B_1) \equiv \mathsf{typ}(B_1)^* \geq \phi^*$.

(2) Otherwise, let $B_1 \equiv \cdots x \cdots x$ and form C_1 from B_1 by just replacing the final x, $C \equiv \cdots x \cdots \mathsf{typ}(A)$. Then $C_1 \downarrow \mathsf{typ}(B_1)$ and by CR, $C_1 \downarrow \phi$. So $\mathsf{typ}(B_1^*) \equiv C_1^* \downarrow \phi^*$.

Anyhow, in both cases $\mathsf{typ}(B_1^*) \geq \phi'^*$, with $\phi' \downarrow \phi$.

Further distinguish:

(1) B_2 does not end in x, then $\mathsf{typ}^*(B_2) \equiv \mathsf{typ}^*(B_2)^* \geq [u{:}\phi^*]\psi^*$.

(2) Otherwise form C_2 from B_2 by replacing its final x, $B_2 \equiv \cdots x \cdots x$, $C_2 \equiv \cdots x \cdots \mathsf{typ}^*(A) \downarrow \mathsf{typ}^*(B_2)$. Then, by $\mathsf{CR}(\mathsf{typ}^*(B_2))$, $C_2 \downarrow [u{:}\phi]\psi$ and, by 5.3.4 $C_2 \geq [u{:}\phi"]\psi"$ with, by PD, $\phi \downarrow \psi"$. Now $\mathsf{typ}^*(B_2^*) \equiv C_2^* \geq [u{:}\phi"^*]\psi"^*$.

So in both cases $\mathsf{typ}^*(B_2^*) \geq [u{:}\phi"^*]\psi"^*$, with $\phi \downarrow \phi"$.

Now use $\mathsf{CR}(\phi)$, this gives $\phi' \downarrow \phi"$, whence $\phi'^* \downarrow \phi"^*$ and $\mathsf{typ}(B_1^*) \downarrow \phi"^*$. So $\vdash\{B_1^*\}B_2^*$, q.e.d.

5.4.4. Lemma: Let $\vdash A$, $\mathsf{CR}_0^*(A)$, $\mathsf{CL}_0^*(A)$. Then $\mathsf{CR}_1(A)$

Proof: Again by induction on length. The crucial case is the critical $\beta\eta$-case: $A \equiv [x{:}A_1]\{x\}[x{:}A_2]A_3$, $x \notin \mathsf{FV}(A_2)$. By 5.4.1 $\mathsf{SA}(\{x\}[x{:}A_2]A_3)$ so $A_1 \downarrow A_2$, $[x{:}A_1]A_3 \downarrow [x{:}A_2]A_3$, q.e.d.

5.4.5 Lemma: Let $\vdash A$, $CR_0^*(A)$, $CL_0^*(A)$. Then $CL_1(A)$, $PT(A)$ and $P^*T(A)$.

Proof: Induction on the big tree of A.

(1) (CL_1). Let $A > B$, we must prove $\vdash B$. The η-outside case we know already. Consider, e.g.: $A \equiv \{A_1\}[x{:}A_2]A_3$, $B \equiv A_3[\![A_1]\!]$. By 5.4.1 $typ(A_1) \downarrow A_2$. By P^*T - ind. hypothesis - we get $typ^*(A_1) \downarrow typ^*(x)$ as well, so by 5.4.3 we are done. This is β-outside CL_1.
Or consider: $A \equiv \{A_1\}A_2$, $A_1 > B_1$, $A_2 > B_2$, $B \equiv \{B_1\}B_2$, $typ(A_1) \geq \phi$, $typ^*(A_2) \geq [u{:}\phi]\psi$. By (e.g.) the ind. hyp. we get $\vdash B_1$, $\vdash B_2$, $typ(A_1) \downarrow typ(B_1)$ and $typ^*(A_2) \downarrow typ^*(B_2)$. Now use CR, this gives $typ(B_1) \downarrow \phi$ and $typ^*(B_2) \downarrow [u{:}\phi]\psi$.
So, by 5.3.4, $typ^*(B_2) \geq [u{:}\phi']\psi'$ and by 5.4.1 $\phi \downarrow \phi'$. Finally $CR(\phi)$ yields $typ(B_1) \downarrow \phi'$, so $\vdash \{B_1\}B_2$, q.e.d. The remaining case of CL_1 is trivial.

(2) (PT). PT_1 we know already. Now let $A >_1 B \geq C$. By CL_1 $\vdash B$ and by ind. hyp. $PT(B)$, so by $CR(typ(B))$, $typ(A) \downarrow typ(C)$, q.e.d.

(3) (P^*T). Let $degree(A) = 1$. Then by PT, if $A \geq B$, $typ(A) \geq F \leq typ(B)$. By $CL_1(A)$ (this implies $CL(A)$) $\vdash B$, so by correctness of types, $\vdash typ(A)$ and $\vdash typ(B)$. Now apply the ind. hyp.: $typ^*(A) \downarrow typ^*(F) \downarrow typ^*(B)$ and use CR: $typ^*(A) \downarrow typ^*(B)$, q.e.d.

5.4.6 Theorem: If $\vdash A$ then $CR(A)$, $CL(A)$

Proof: By induction on the big tree of A. The ind. hyp. reads $CR_0^*(A)$, $CL_0^*(A)$, and the preceding lemmas produce $CR_1(A)$ and $CL_1(A)$. As we noticed before, this yields $CR(A)$ and $CL(A)$.

5.4.7 Corollary: If $\vdash A$ then $SA(A)$, $PD(A)$, $PT(A)$, $P^*T(A)$ and $SC(A)$.

5.4.8 Note: The separate inductions on big trees in 5.4.1, 5.4.5 and 5.4.6 can of course be compressed into a single induction on big trees.

VII.6 Various equivalence results

6.1 Introduction

In VII.2 we introduced $\Lambda(\eta)$ with and without (definitional) constants. The results in VII.3-5 are derived for the constant-less system. In this section we extend these results in an indirect way to the remaining systems, by showing that, in a certain sense, they can be embedded in the constant-less version.

Sec. 6.2 is devoted to primitive constants only. First we give a translation which eliminates the constant-expression. Then we explain the relations between (a) the system with constants, (b) its image under the translation, and (c) the constant-less system. Afterwards we easily extend our nice properties (CL, CR, BT) to the system with constants.

Sec. 6.3 covers the additional extension with definitional constants. In 6.4 we prove another equivalence: between Nederpelt's single line presentation with *abstractorstrings* Q and our presentation, with contexts ξ. In this case too, the correspondence is close enough to show that Nederpelt's original system satisfies the required properties.

6.2 Eliminating primitive constants

6.2.1 The translation '

For the system with constants (for short: C-system) we use the notations $\Lambda(\eta)_C$ and $\vdash_C$. Now we define a translation of the C-system into the system without constants. The translation (notation ') is characterized by:

(1) it transforms constants p into variables p',

(2) it converts constant-expressions $p(A_1,\cdots,A_k)$ into appl. expressions $\{A_k'\}\cdots\{A_1'\}p'$,

(3) it eliminates *schemes* $\bar{y}\ \mathsf{E}\ \bar{\beta} * p(\bar{y})\ \mathsf{E}\ \gamma$ one by one from the book by including an additional *assumption* $p'\ \mathsf{E}\ [\bar{y}:\bar{\beta}']\gamma'$ in the context,

(4) it commutes with the other formation rules (for expressions, strings and contexts).

Thus a statement $\mathcal{B}$; $\vdash_C A$ is translated into $\mathcal{B}'$, $\xi' \vdash A'$ where $\mathcal{B}'$ is understood to be a context consisting of the additional assumptions for the new variables p'.

6.2.2 Why the indirect approach?

Below we use the properties of the constant-less system in our proof of the desired correspondence. Afterwards we can extend these properties to the C-system.

The point is that the constant-less system is definitely easier to handle. In particular: the fact that the typ of a constant-expression is constructed by *substitution* is a complicating factor, because correctness of types is not immediate any more.

E.g. by using this indirect approach we would have been able to introduce constants without using degree-norms.

6.2.3 The nature of the correspondence

For terminology about extensions we refer to V.3.3.2. However, because we study an algorithmic system now, we replace $A \in B$ by $\mathsf{typ}(A) \downarrow B$ and $A\ Q\ B$ by $A \downarrow B$.

Clearly the C-system is an *extension* of the system without constants. Because typ and $\geq$ remain the same, it is a *conservative* extension too. Of course it is *not* an *unessential* one: primitive constant-expressions do not main reduce at all, so they can never be definitionally equivalent to an expression without constants.

Contrarily, the translation ' maps expression (and contexts), correct w.r.t. $\mathcal{B}$ in the C-system, *properly* into the expressions (and contexts), correct w.r.t. $\mathcal{B}'$: expressions $\{\bar{A}\}p'$ that do not have enough arguments in front, i.e. where $|\bar{A}|$ is smaller than the arity of p have no counterpart in the C-system.

For the image of the C-system (w.r.t. a fixed book $\mathcal{B}$) under ', we introduce the notation $\vdash\_$. I.e.

$$\eta \vdash\_, \text{ resp. } \eta \vdash\_ B: \Leftrightarrow \eta \equiv \xi',\ B \equiv A' \text{ and } \mathcal{B};\ \xi \vdash_C \text{ resp. } \mathcal{B};\ \xi \vdash_C A.$$

Then below it will appear that the expressions (and contexts) correct w.r.t. $\mathcal{B}'$ in the constant-less system, form a conservative extension of

the system $\vdash\_$. In the presence of η-reduction, it will be definitional (so unessential) too. See sec. 6.2.9.

6.2.4 Facts about '

Notice that ' is a purely "syntactical" matter, which has nothing to do with correctness: pretyped-ness is sufficient.

As a map from statements $\mathcal{B}; \xi \vdash A$ to statements $\mathcal{B}', \xi' \vdash A'$ the translation in not one-one, but as a map from $\mathcal{B}$-expressions and - contexts into $\mathcal{B}'$-expressions and - contexts it is one-one indeed. For the (partial) inverse we use the notation $_0$:

$$(A')_0 :\equiv A$$

Clearly, $A[\![\bar{B}]\!]' \equiv A'[\![\bar{B}']\!]$ so $A \geq B \Rightarrow A' \geq B'$, so $A \downarrow B \Rightarrow A' \downarrow B'$. Further $\mathsf{typ}(A') \geq \mathsf{typ}(A)'$ - there are only *head*-β^i contractions involved, where degree(A) = i+1 (for the definition of head- and of ^{i}-reduction see V.3.3.3 and V.4.3.3.5). And $\mathsf{typ}(A') \equiv \phi'$ for some ϕ.

If there is no η-reduction then we have

(1) $A' > B \Rightarrow A > B_0,\ B_0' \equiv B$

so (2) $A' \geq B' \Rightarrow A \geq B$

and (3) $A' \downarrow B' \Rightarrow A \downarrow B$

6.2.5 ' and η-reduction

With η-reduction, (1) above does not hold any more:
$([x{:}\alpha]p(A_k,\cdots,A_1,x))' \equiv [x{:}\alpha'][x]\{\bar{A}'\}p'$ may reduce to $\{\bar{A}'\}p'$.

Lemma: $A' \geq_\eta B' \Rightarrow A \geq_\eta B$

Proof: Ind. on the length of A. E.g. let $A \equiv [x{:}\alpha]C$, so $A' \equiv [x{:}\alpha']C'$.

If $B' \equiv [x{:}\beta']D'$ with $\alpha' \geq_\eta \beta'$, $C' \geq_\eta D'$ use the ind. hyp. Otherwise $C' \geq_\eta \{x\}B'$. The latter expression is $(\{x\}B)'$ so by ind. hyp. $C \geq_\eta \{x\}B$ and $A \geq_\eta B$, q.e.d. □

Now let $A' \geq B'$ then by βη-pp: $A' \geq_\beta C \geq_\eta B'$. This $C \blacksquare C_0'$, so $C_0 \geq_\eta B$ by the lemma, and $A \geq B$. This is property (2) above. Property (3) can be proved in the same fashion.

6.2.6 Something about typ^*

Lemma: $\vdash B' \Rightarrow (\vdash \mathrm{typ}^*(B)',\ \mathrm{typ}^*(B)' \downarrow \mathrm{typ}^*(B'))$

Proof: The translation ' preserves the degree, of course. We use induction on degree(B'). The degree 1 case is immediate. Otherwise $\mathrm{typ}^*(B') \equiv \mathrm{typ}^*(\mathrm{typ}(B'))$ and $\mathrm{typ}^*(B)' \equiv \mathrm{typ}^*(\mathrm{typ}(B))'$. By correctness of types $\vdash \mathrm{typ}(B')$, reducing to $\mathrm{typ}(B)'$ and by $\mathrm{P}^*\mathrm{T}$ $\mathrm{typ}^*(B') \downarrow \mathrm{typ}^*(\mathrm{typ}(B)')$. By CL $\vdash \mathrm{typ}(B)'$ so by ind. hyp. $\vdash \mathrm{typ}^*(B)'$, q.e.d., and $\mathrm{typ}^*(B)' \downarrow \mathrm{typ}^*(\mathrm{typ}(B)')$. By correctness of types $\vdash \mathrm{typ}^*(\mathrm{typ}(B)')$ so by CR $\mathrm{typ}^*(B)' \downarrow \mathrm{typ}^*(B')$, q.e.d. □

Now that we know CL, CR, PD and SA for $\Lambda(\eta)$ we can extend property 5.3.4 to: $\vdash^1 A$, $\vdash^1 [x{:}\alpha]C$, $A \downarrow [x{:}\alpha]C \Rightarrow A \geq_\beta [x{:}\beta]D$, $\alpha \downarrow \beta$. So, as alternative application condition, equivalent to the one used originally:

$$\vdash A,\ \vdash B,\ \mathrm{typ}(A) \geq \alpha,\ \mathrm{typ}^*(B) \geq [x{:}\alpha]C \Rightarrow \vdash \{A\}B$$

we can as well use, e.g.

$$\mathrm{typ}(A) \downarrow \alpha,\ \mathrm{typ}^*(B) \geq_\beta [x{:}\alpha]C$$

or

$$\mathrm{typ}(A) \downarrow \alpha,\ \mathrm{typ}^*(B) \downarrow [x{:}\alpha]C,\ \vdash [x{:}\alpha]C$$

6.2.7 The proof of the correspondence

Theorem: $\mathcal{B};\ \xi \vdash_C A \Leftrightarrow \mathcal{B}', \xi' \vdash A'$

Proof: $\Rightarrow$. By induction on correctness. The formation of the context $\mathcal{B}'$ is allowed, due to the liberal degree conventions of $\Lambda(\eta)$. Consider, e.g. the appl.rule: let $\vdash_C A$, $\vdash_C B$, $\mathrm{typ}(A) \geq \alpha$, $\mathrm{typ}^*(B) \geq [x{:}\alpha]C$. By ind. hyp. $\vdash A'$, $\vdash B'$, further $\mathrm{typ}(A') \geq \mathrm{typ}(A)' \geq \alpha'$ and by the lemma in 6.2.6 $\vdash \mathrm{typ}^*(B)'$, $\mathrm{typ}^*(B') \downarrow \mathrm{typ}\ (B)' \geq [x{:}\alpha']C'$. By CR, $\mathrm{typ}^*(B') \downarrow [x{:}\alpha']C'$. By CL, $\vdash [x{:}\alpha']C'$ so, by the alternative appl. rule $\vdash \{A'\}B'$. Or consider the instantiation rule: $\vdash_C B_1, \cdots, \vdash_C B_k$, $\bar{y}\ \mathrm{E}\ \bar{\beta}^* p(\bar{y})\ \mathrm{E}\ \gamma$ is a scheme in $\mathcal{B}$, $|\bar{y}| = k$ and $\mathrm{typ}(B_i) \downarrow \beta_i[\![\bar{B}]\!]$ for $i=1,\cdots,k$. The translated scheme reads $p'\ \mathrm{E}\ [(\bar{y}{:}\bar{\beta}')]\gamma'$. By ind. hyp. $\vdash B_1', \cdots, \vdash B_k'$. Now $\mathrm{typ}(B_1') \geq \mathrm{typ}(B_1)' \downarrow \beta_1'$, $\mathrm{typ}^*(p') \equiv [y_1{:}\beta_1']\cdots\tau$, so $\vdash \{B_1'\}p'$. Further $\mathrm{typ}(B_2') \geq \mathrm{typ}(B_2)' \downarrow \beta_2[\![B_1]\!]' \equiv \beta_2'[\![B_1']\!]$ and $\mathrm{typ}^*(\{B_1'\}p') \equiv \{B_1'\}\mathrm{typ}^*(p') > [y_2{:}\beta_2'[\![B_1']\!]]\cdots\tau$, so $\vdash \{B_2'\}\{B_1'\}p'$.

Etc. up to $\vdash\{B'_k\}\cdots\{B'_1\}p' \equiv p(\bar{B})'$, q.e.d.

$\Leftarrow$ Also by induction on correctness. E.g. consider an appl. expression. Either it is $(\{A\}B)'$ or it is $p(\bar{B})'$. First case: if $\vdash\{A'\}B'$ then $\vdash A'$ (so $\vdash_C A$), $\vdash B'$ (so $\vdash_C B$), $\mathsf{typ}(A)' \leq \mathsf{typ}(A') \geq \alpha$ (so $\mathsf{typ}(A)' \downarrow \alpha$), $\mathsf{typ}^*(B)' \downarrow \mathsf{typ}^*(B') \geq [x{:}\alpha]C$ (so $\mathsf{typ}^*(B) \downarrow [x{:}\alpha]C$). Hence $\mathsf{typ}^*(B)' \geq_\beta [x{:}\beta]D \equiv [x{:}\beta'_0]D'_0 \equiv ([x{:}\beta_0]D_0)'$ with $\alpha \downarrow \beta$. By CR $\mathsf{typ}(A)' \downarrow \beta'_0$, so $\mathsf{typ}(A) \downarrow \beta_0$, and $\mathsf{typ}^*(B) \geq [x{:}\beta_0]D_0$, so $\vdash_C\{A\}B$. Second case: $\vdash\{B'_k\}\cdots\{B'_1\}p'$ so $\vdash_C B_k, \cdots, \vdash_C B_1$. Let $\bar{y}\ \bar{E}\ \bar{\beta} \star p(\bar{y})\ E\ \gamma$ be the scheme of p. $\mathsf{Typ}(B'_1) \geq \phi_1$, $\mathsf{typ}^*(p') \equiv [y_1{:}\beta'_1]\cdots\tau \geq [y_1{:}\phi_1]\cdots\tau$ so $\mathsf{typ}(B_1)' \downarrow \beta'_1$, $\mathsf{typ}(B_1) \downarrow \beta_1$. Further $\mathsf{typ}(B'_2) \geq \phi_2$, and $[y_2{:}\beta'_2[\![B'_1]\!]]\cdots\tau < \{B'_1\}\mathsf{typ}^*(p') \equiv \mathsf{typ}^*(\{B'_1\}p') \geq [y_2{:}\phi_2]\cdots\tau$, so $\mathsf{typ}(B_2) \downarrow \beta_2[\![B_1]\!]$. Etc. up to $\mathsf{typ}(B_k) \downarrow \beta_k[\![\bar{B}]\!]$ and $\vdash_C p(\bar{B})$ q.e.d. □

6.2.8 The required properties

Theorem: The strictly normable constant-expressions (see IV.3.4) satisfy BT

Proof: Strictly normable C-expressions transform into strictly normable expressions without constants under the translation '. And all $\overset{*}{\to}$ sequences of C-expressions A transform into subsequences of $\overset{*}{\to}$ sequences of A': (1) $\mathsf{typ}(A') \geq \mathsf{typ}(A)'$, (2) $A >_1 B \Rightarrow A' >_1 B'$, (3) $A \subset B \Rightarrow A' \subset B'$. So by BT for the constant-less version we are done. □

Theorem: $\Lambda(\eta)_C$ satisfies CR

Proof: Let $\vdash_C A$, $A \geq B$, $A \geq C$. By the $\Rightarrow$-part of the correspondence $\vdash A'$ and by CR for $\Lambda(\eta)$ $B' \downarrow C'$, so $B \downarrow C$, q.e.d. □

Theorem: $\Lambda(\eta)_C$ satisfies CL

Proof: Let $\vdash_C A$, $A > B$. Then $\vdash A'$, A', B' so by CL $\vdash B'$. So $\vdash_C B$.

Theorem: $\Lambda(\eta)_C$ satisfies SA, PD, PT, P*T, SC etc.

Proof: Either from CL and CR, or using the correspondence □

6.2.9 An unessential extension result

Now we explain the connection between the $\vdash\_$-system and the ordinary $\vdash$-system of $\Lambda(\eta)$ without constants. Recall

$\vdash_- A' \Leftrightarrow \vdash_C A$, i.e. $\vdash_- A \Leftrightarrow \vdash_C A_0$

The first half of the correspondence result shows $\vdash_- \Rightarrow \vdash$, i.e. a simple extension result. Now we define a translation $^-$ from the larger into the smaller system, as follows: if $\bar{x} \in \bar{\alpha} * p(\bar{x}) \in \gamma$ is a scheme in B, $|\bar{x}| = k$, $i < k$ then

$(\{A_i\}\cdots\{A_1\}p')^- :\equiv [x_{i+1}:\alpha_{i+1}[\![\bar{A}^-]\!]]\cdots[x_k:\alpha_k'[\![\bar{A}^-]\!]]\{x_k\}\cdots\{x_{i+1}\}\{A_i^-\}\cdots\{A_1^-\}p'$, i.e. we η-expand until p' gets enough arguments in front. For the rest $^-$ acts as identity.

Clearly $A^- \geq_\eta A$, $A^- \equiv (A_0^-)'$. Viz. $(\{A_i\}\cdots\{A_1\}p')^-_0 \equiv$

$[x_{i+1}:\alpha_{i+1}[\![\bar{A}_0^-]\!]]\cdots[x_k:\alpha_k[\![\bar{A}_0^-]\!]]p(\bar{A}_0^-, x_{i+1}, \cdots, x_k)$.

The translation is a bit intricate, because $(\{A\}B)^-$ is not necessarily $\{A^-\}B^-$. In general $\{A^-\}B^- \geq_\beta (\{A\}B)^-$ and $B^-[\![A^-]\!] \geq_\beta (B[\![A]\!])^-$. Further $\mathrm{typ}(A^-) \geq_\eta \mathrm{typ}(A)^-$, and also $\mathrm{typ}(A^-) \downarrow_\beta \mathrm{typ}(A)^-$. Without proof we state that $A \geq B \Rightarrow A^- \geq B^-$, and that $\mathrm{typ}^*(A^-) \downarrow \mathrm{typ}^*(A)^-$. From these facts, it can be proved that: $\vdash A \Rightarrow \vdash A^-$, so by the second part of the correspondence $\vdash A \Rightarrow \vdash_- A^-$.

In case of $\beta\eta$-reduction, this is a typical unessential extension result.

6.3 The case of definitional constants

6.3.1 We have three main possibilities to incorporate definitional constants in our theory. The first one studies the new system (we call it $\Lambda(\eta)_d$, with correctness predicate $\vdash_d$, and also speak about the d-system etc.) independently, as a separate subject, the second one considers it as an extension of $\Lambda(\eta)_C$, and the third one embeds it into $\Lambda(\eta)$, by extending the translation ' from the previous sections in order to cover definitional constants.

Here we actually use the second method, and just mention some points on the third one.

But we start by proving the *big tree theorem* for $\Lambda(\eta)_d$, for reasons of completeness and as an indispensable prerequisite for the separate study of the system (method one above).

6.3.2 The big tree theorem for $\Lambda(\eta)_d$

In 6.2.8 we proved BT for $\Lambda(\eta)_c$ by means of the embedding ' into $\Lambda(\eta)$. It is indeed possible to extend ' to the case of definitional constants, but (see 6.3.3) the translation does not reflect the type-structure sufficiently, which makes this method fail here.

So instead we revise the BT-proof of 5.2 (for $\Lambda(\eta)$) and adapt in to the $\Lambda(\eta)_d$-case, which is relatively easy. First we mention the BT-condition (see 4.3.2):

(5) $BT(p(\bar{A})) \Leftrightarrow BT(A_1), \cdots, BT(A_k), BT(typ(p)[\![\bar{A}]\!])$

(6) $BT(d(\bar{A})) \Leftrightarrow BT(A_1), \cdots, BT(A_k), BT(typ(d)[\![\bar{A}]\!]), BT(def(d)[\![\bar{A}]\!])$.

The $\beta\delta\tau$-SN conditions are quite analogous, and, as in 4.4.3, we have:

Theorem: $\beta\delta\tau\text{-SN}(A) \Rightarrow BT(A)$

This suggests that, in this case as well, the substitution property of $\beta\delta\tau$-SN is crucial. We choose to adapt the first BT-proof (sec. 4.5) so need the replacement theorem (see 4.5.6) instead: Let * denote $\{x/\rho(A)\}_{LR}$, let B be normable, $\mu(x) \equiv \mu(A)$, A, B $\beta\delta\tau$-SN. Then:

$$C \in B^* \Rightarrow C \ \beta\delta\tau\text{-SN}$$

Proof: As in 4.5.6. We consider a single reduction step $C >_{1,\beta\delta\tau} D$. For all β-steps and all τ-steps concerning variables (not constants), $\beta\delta\tau$-SN(D) can be proved as in 4.5.6. The remaining steps, i.e. δ-steps and τ-steps of constants, can only fall into the categories (1) and (2a) so we get $\beta\delta\tau$-SN(D) by ind. hyp. II or ind. hyp. III. So we have a list of corollaries:

(1) B normable, $\mu(x) \equiv \mu(A)$, A, B $\beta\delta\tau$-SN $\Rightarrow B[\![A]\!]$ $\beta\delta\tau$-SN

(2) B normable, $\mu(x_i) \equiv \mu(A_i)$,
A_i (i=1,$\cdots$,k) and B $\beta\delta\tau$-SN $\Rightarrow B[\![\bar{A}]\!]$ $\beta\delta\tau$-SN

Proof: The simultaneous substitution can be simulated by iterated single substitution.

(3) B normable $\Rightarrow B$ $\beta\delta\tau$-SN

Proof: Induction on pretyped expressions. For the new cases use the previous corollary.

(4) B normable $\Rightarrow B$ $\beta\eta\delta\tau$-SN

Proof: $\tau\eta$-pp extends to the present case (see 5.2.1), $\delta\eta$-pp we knew already (see II.7.4). This gives $(\beta\delta\tau)$-η-pp and, by η-SN, $\beta\eta\delta\tau$-SN.

(5) B normable $\Rightarrow$ $\beta\eta\delta$-BT(B)

6.3.3 The translation into $\Lambda(\eta)$

Here we show how the translation ' can be extended to the d-case. Viz. an expression $d(\bar{A})$ transforms into $\{A'_k\}\cdots\{A'_1\}[\bar{x}:\bar{\alpha}']D'$, where $\bar{x}\ \mathsf{E}\ \bar{\alpha} \star d(\bar{x}) :\equiv D \star d(\bar{x})\ \mathsf{E}\ \gamma$ is the scheme of d.
This translation behaves nicely w.r.t. to reduction: $A > B \Rightarrow A' \geq B'$. But of course it is possible that an expression A' β-reduces to an expression which is not some B'. This is in contrast with the situation with primitive constants where this could only occur by η-reduction. The best we can get is: $A' >_{1,\beta} B \Rightarrow B \geq_\beta C$, $A >_{1,\beta\delta} C$. So, e.g. by ind. on $\theta_\beta(A')$, we get $A' \geq_\beta B \Rightarrow B \geq_\beta C'$, $A \geq C$. For the rest the translation seems to be not too useful, because properties like $A' \downarrow B' \Rightarrow A \downarrow B$ (at least where η-reduction is allowed) and $\mathsf{typ}(A') \downarrow \mathsf{typ}(A)'$ are only valid in the correct fragment. Note that $\mathsf{typ}(A') \geq \mathsf{typ}(A)'$ is simply wrong here.

6.3.4 Some properties of $\Lambda(\eta)_c$

Translation of $\Lambda(\eta)_d$ into $\Lambda(\eta)_c$ just requires the elimination of abbreviations, which can be done by δ-normalization. In the next sections we show that this actually constitutes a translation, i.e. that it preserves correctness. Here we first give some properties of $\Lambda(\eta)_c$ which we need in the - rather complicated - proof below.

The single substitution result (of $\Lambda(\eta)$, and of $\Lambda(\eta)_c$ too)

$$\vdash A,\ \mathsf{typ}(A) \downarrow \alpha,\ (x\ \mathsf{E}\ \alpha, \eta \vdash B) \Rightarrow \eta[\![A]\!] \vdash B[\![A]\!]$$

can, by induction on $|\bar{x}|$, be extended to a simultaneous substitution result

$$\vdash \bar{A},\ \mathsf{typ}(A_i) \downarrow \alpha_i[\![\bar{A}]\!] \quad \text{for } i = 1,\cdots,|\bar{A}|,\ (\bar{x}\ \mathsf{E}\ \bar{\alpha} \vdash B) \Rightarrow \vdash B[\![\bar{A}]\!]\ .$$

The properties of sec. 3.2.2 concerning the typ of substitution results can be generalized to (1) the simultaneous substitution case,

(2) successive applications of typ, resulting in:
$typ^j(A_i) \twoheadrightarrow typ^j(x_i)[\![\bar{A}]\!]$, for $i = 1,\cdots,|\bar{A}| \Rightarrow typ^j(B)[\![\bar{A}]\!] \twoheadrightarrow typ^j(B[\![\bar{A}]\!])$, for all relevant j, where typ^j stands for j successive applications of typ. This holds for $\Lambda(\eta)$ but also for $\Lambda(\eta)_c$ and $\Lambda(\eta)_d$. Notice, that in case B does not end in one of the x_i we even have

$$typ^j(B[\![\bar{A}]\!]) \equiv typ^j(B)[\![\bar{A}]\!]$$

6.3.5 The translation into $\Lambda(\eta)_c$

Our notation for the translation is $^-$. For expressions $^-$ amounts just to taking δ-normal form. It is clear how $^-$ acts on strings and contexts. It is intended that the book $\mathcal{B}^-$ is formed from $\mathcal{B}$ by δ-normalizing and by skipping the abbreviational schemes. The translation is of course not 1-1.

We recall that $B[\![\bar{A}]\!]^- \equiv B^-[\![\bar{A}^-]\!]$, that $d(\bar{A})^- \equiv def(d)^-[\![\bar{A}^-]\!]$, and that δ-reduction commutes with $\beta\eta$-reduction. The latter implies

$$A \geq B \Rightarrow A^- \geq B^- \quad \text{and} \quad A \twoheadrightarrow B \Rightarrow A^- \twoheadrightarrow B^-.$$

6.3.6 The translation preserves correctness

Theorem: $\mathcal{B}$; $\xi \vdash_d A \Rightarrow \mathcal{B}^-$; $\xi^- \vdash_c typ^i(A)^-$, $typ^i(A)^- \twoheadrightarrow typ^i(A^-)$ for $i=0,\cdots,$ degree(A)-1 (this concludes $\vdash_c A^-$ itself).

Proof: By induction on $\vdash_d$. Crucial cases are: (1) the application case: $A \equiv \{A_1\}A_2$, $\vdash_d A_1$, $\vdash_d A_2$, $typ(A_1) \geq \alpha$, $typ(A_2) \geq [x{:}\alpha]C$. By the ind. hyp. $\vdash_c A_1^-$, $\vdash_c typ(A_1)^-$, $typ(A_1)^- \twoheadrightarrow typ(A_1^-)$, $\vdash_c typ^i(A_2)^-$, $typ^i(A_2)^- \twoheadrightarrow typ^i(A_2^-)$. Clearly $typ(A_1)^- \geq \alpha^-$ so by CR $typ(A_1^-) \twoheadrightarrow \alpha^-$. Similarly, $typ^*(A_2)^- \geq [x{:}\alpha^-]\bar{C}$, and $typ^*(typ^i(A_2)^-) \twoheadrightarrow typ^*(typ^i(A_2^-)) \equiv typ^*(A_2^-) \twoheadrightarrow typ^*(A_2)^-$ (by P*T), so by CR, $typ^*(typ^i(A_2)^-) \twoheadrightarrow [x{:}\alpha^-]C^-$. Hence $\vdash_c typ^i(\{A_1\}A_2)^- (\equiv \{A_1^-\}typ^i(A_2)^-)$. See 6.2.6 for the alternative appl. condition. The property $typ^i(\{A_1\}A_2)^- \twoheadrightarrow typ^i((\{A_1\}A_2)^-)$ is trivial. (2) the definitional constant case: $A \equiv d(\bar{B})$, $\vdash_d B_j$, $typ(B_j) \twoheadrightarrow \beta_j[\![\bar{B}]\!]$ for $j=1,\cdots,|\bar{y}|$, where $\bar{y}\ E\ \bar{\beta} * d(\bar{y}) := D * d(\bar{y})\ E\ \gamma$ is the scheme of d. By ind.

hyp. $\vdash_C B_j^-$ and $typ(B_j^-) \downarrow typ(B_j)^- \downarrow \bar{\beta}_j[\![\bar{B}^-]\!]$. Also by ind. hyp. $\bar{y} \in \bar{\beta}^- \vdash_C D^-$, $\bar{y} \in \bar{\beta}^- \vdash_C \gamma^-$, $\bar{y} \in \bar{\beta}^- \vdash_C typ(D)^-$ and $typ(D)^- \downarrow typ(D^-)$. So, by the simultaneous subst. property, $\vdash_C D^-[\![\bar{B}^-]\!] (\equiv A^-)$, $\vdash_C \gamma^-[\![\bar{B}^-]\!] (\equiv typ(A)^-)$. We know that $\gamma \downarrow typ(D)$, so $\gamma^- \downarrow typ(D)^-$ so by CR $typ(D^-) \downarrow \gamma^-$, whence $typ(D^-)[\![\bar{B}^-]\!] \downarrow \gamma^-[\![\bar{B}^-]\!]$ and, again by CR, $typ(A^-) \downarrow typ(A)^-$. Now there is left to prove: (1) $\vdash_C typ^i(A)^- (\equiv typ^{i-1}(\gamma[\![\bar{B}]\!])^-)$, and (2) $typ^i(A)^- \downarrow typ^i(A^-)$, i.e. $typ^{i-1}(\gamma[\![\bar{B}]\!])^- \downarrow typ^i(D^-[\![\bar{B}^-]\!])$, for i=2,•••,degree(A)-1. The ind. hyp. gives us $\vdash_C typ^{i-1}(\gamma)^-$, $\vdash_C typ^i(D)^-$, $typ^{i-1}(\gamma)^- \downarrow typ^{i-1}(\gamma^-)$, $typ^i(D)^- \downarrow typ^i(D^-)$ for these i, and $\vdash_C typ^k(B_j)^- (\downarrow typ^k(B_j^-))$, for k=0,•••,degree($B_j$)-1, for j=1,•••,$|\bar{y}|$. Now (2) is simple: $typ^{i-1}(\gamma[\![\bar{B}]\!]) \downarrow typ^{i-1}(\gamma)[\![\bar{B}]\!]$ so $typ^{i-1}(\gamma[\![\bar{B}]\!])^- \downarrow typ^{i-1}(\gamma)^-[\![\bar{B}^-]\!] \downarrow typ^{i-1}(\gamma^-)[\![\bar{B}^-]\!] \downarrow typ^i(\bar{D}^-)[\![\bar{B}^-]\!] \downarrow typ^i(D^-[\![\bar{B}^-]\!])$. Here we use PT and the substitution property of types. By CR we get (2). Property (1) we formulate in the form of a lemma.

Lemma: Let $\bar{y} \in \bar{\beta} \vdash_d \gamma$, $\vdash_d B_j$, for j=1,•••,$|\bar{y}|$ with γ and $\bar{B}$ as above. Then $\vdash_C typ^i(\gamma[\![\bar{B}]\!])^-$, for i=0,•••,degree($\gamma$)-1.

Proof: If γ does not end in some of the y_j then $typ^i(\gamma[\![\bar{B}]\!])^- \equiv typ^i(\gamma)^-[\![\bar{B}^-]\!]$ which is correct by the simultaneous subst. property. This also covers the case i=0 (which we knew already). For the rest we use induction on the length of γ. The case $\gamma \equiv y_j$ is true by assumption. Further consider the application case: $\gamma \equiv \{\gamma_1\}\gamma_2$, $\vdash_d \gamma_1$, $\vdash_d \gamma_2$, $typ(\gamma_1) \geq \phi$, $typ^*(\gamma_2) \geq [z{:}\phi]E$. By ind. hyp. $\vdash_C \gamma_1[\![\bar{B}]\!]^-$, $\vdash_C typ(\gamma_1[\![\bar{B}]\!])^-$, $\vdash_C typ^i(\gamma_2[\![\bar{B}]\!])^-$ for all i. We have $typ(\gamma_1^-[\![\bar{B}^-]\!]) \downarrow typ(\gamma_1^-)[\![\bar{B}^-]\!] \downarrow typ(\gamma_1)^-[\![\bar{B}^-]\!] \geq \phi^-[\![\bar{B}^-]\!]$, so by CR $typ(\gamma_1[\![\bar{B}]\!]^-) \downarrow \phi[\![\bar{B}]\!]^-$. Similarly $typ^i(\gamma_2[\![\bar{B}]\!])^- \downarrow typ^i(\gamma_2)^-[\![\bar{B}^-]\!] \downarrow typ^i(\gamma_2^-)[\![\bar{B}^-]\!]$. So by CR and P*T $typ^*(typ^i(\gamma_2[\![\bar{B}]\!])^-) \downarrow typ^*(typ^i(\gamma_2^-)[\![\bar{B}^-]\!]) \downarrow typ^*(\gamma_2^-)[\![\bar{B}^-]\!] \downarrow typ^*(\gamma_2)^-[\![\bar{B}^-]\!] \geq [z{:}\phi[\![\bar{B}]\!]^-]E[\![\bar{B}]\!]^-$. Again by CR, $typ^*(typ^i(\gamma_2[\![\bar{B}]\!])^-) \downarrow$

$[z:\phi[\![\bar{B}]\!]^-]E[\![\bar{B}]\!]^-$, whence $\vdash_c \mathrm{typ}^i(\{\gamma_1[\![\bar{B}]\!]\}\gamma_2[\![B]\!])^-$, q.e.d.

The abstr. case is straightforward. This finishes the proof of the lemma. This finishes the definitional constant case of the theorem. Now the remaining cases of the theorem are straightforward. This finishes the proof of the theorem. □

Corollary: $B;\ \xi\vdash_d A \Rightarrow B^-;\xi^-\vdash_c A^-,\ B^-;\ \xi^-\vdash_c \mathrm{typ}(A)^-,\ B^-;\ \xi^-\vdash_c \mathrm{typ}^*(A)^-$

and $\mathrm{typ}(A^-) \downarrow \mathrm{typ}(A)^-,\ \mathrm{typ}^*(A^-) \downarrow \mathrm{typ}^*(A)^-$.

6.3.7 Is $\Lambda(\eta)_d$ a definitional extension of $\Lambda(\eta)_c$?

The above corollary amounts to the unessential extension properties UE2 and UE3 (see V.3.3.2). Of course we also have $\vdash_c A \Rightarrow A \equiv A^-$ and it is tempting to conclude the other half of UE1:

$$B;\ \xi\vdash_d A \Rightarrow B;\ \xi\vdash_d A^-$$

from the corollary. This is however not immediate as yet: we can conclude

$$B;\ \xi\vdash_d A \Rightarrow B^-;\ \xi^-\vdash_d A^-$$

and we know

$$(B;\xi)\text{-}\mathrm{typ}(A^-) \geq (B^-;\xi^-)\text{-}\mathrm{typ}(A^-)$$

but we hardly know anything about

$$(B;\xi)\text{-}\mathrm{typ}^*(A^-).$$

Instead, we first prove the substitution theorem for $\Lambda(\eta)_d$; this gives correctness of types, as well as δ-CL. The latter implies UE1, which completes our definitional extension result.

6.3.8 Some nice properties of $\Lambda(\eta)_d$

The corollary in 6.3.6 gives us already some nice results.

Theorem: $\Lambda(\eta)_d$ satisfies (1) CR, (2) SA and (3) PD

Proof: (1) Let $\vdash_d A,\ B \leq A \geq C$. Then $\vdash_c A^-,\ B^- \leq A^- \geq C^-$. By CR $B^- \downarrow C^-$, so $B \downarrow C$.

(2) Let $\vdash_d \{A\}[x:B]C$. Then $\vdash_c \{A^-\}[x:B^-]C^-$ so $\mathrm{typ}(A^-) \downarrow B^-$. Further $\mathrm{typ}(A)^- \downarrow \mathrm{typ}(A^-)$ and by CR, $\mathrm{typ}(A) \downarrow B$.

(3) Let $\vdash_d [x:\alpha]A$, $[x:\alpha]A \geq [x:\beta]B$. Then $\vdash_c [x:\alpha^-]A^-$, $[x:\alpha^-]A^- \geq [x:\beta^-]B^-$. By PD $\alpha^- \downarrow \beta^-$ so $\alpha \downarrow \beta$. □

Remark: We also prove some form of PT and P*T.

Let $\vdash_d A$, $\vdash_d B$, $A \geq B$. Then $typ(A) \downarrow typ(B)$ and $typ^*(A) \downarrow typ^*(B)$.

Proof: $\vdash_c A^-$, $\vdash_c B^-$, $A^- \geq B^-$, so $typ(A)^- \downarrow typ(A^-) \downarrow typ(B^-) \downarrow typ(B)^-$ and by CR $typ(A) \downarrow typ(B)$. Similar for typ^*. □

6.3.9 The substitution theorem for $\Lambda(\eta)_d$

Lemma: Let $\vdash_d B_1$, $\vdash_d B_2$, $\vdash_c \{B_1^-\}B_2^-$. Then $\vdash_d \{B_1\}B_2$

Proof: $typ(B_1) \geq typ(B_1)^- \downarrow typ(B_1^-) \geq \phi$, $typ^*(B_2) \geq typ^*(B_2)^- \downarrow typ^*(B_2^-) \downarrow [u:\phi]\psi$. By CR, $typ(B_1) \downarrow \phi$, $typ^*(B_2) \downarrow [u:\phi]\psi$. So $\vdash_d \{B_1\}B_2$. □

Lemma: Let $\vdash_d B$, i=1,···,k. Let $\bar{y}\ E\ \bar{\beta} * c(\bar{y})\ E\ \gamma$ be the scheme of c, with $|\bar{y}| = k$. Let $\vdash_c c(\bar{B}^-)$. Then $\vdash_d c(\bar{B})$.

Proof: $typ(B_i) \geq typ(B_i)^- \downarrow typ(B_i^-) \downarrow \beta_i^-[\![\bar{B}^-]\!] \leq \beta_i[\![\bar{B}]\!]$. By CR, $typ(B_i) \downarrow \beta_i[\![\bar{B}]\!]$. So $\vdash_d c(\bar{B})$. □

Theorem: Let $\bar{x}\ E\ \bar{\alpha} \vdash_d B$. Let * stand for $[\![\bar{x}/\bar{A}]\!]$. Let $\vdash_d A_i$ and $typ(A_i) \downarrow \alpha_i^*$ for i=1,···,$|\bar{x}|$. Then $\vdash_d B^*$.

Proof: We use induction on $\vdash_d B$. So, by ind. hyp. $\vdash_d \alpha_i^*$ for i=1,···,$|\bar{x}|$. Now $typ(A_1) \downarrow \alpha_1$. So $typ(A_1^-) \downarrow typ(A_1)^- \downarrow \alpha_1^-$ and by CR $typ(A_1^-) \downarrow \alpha_1^-$. Similarly $typ(A_2^-) \downarrow \alpha_2^{*-} \equiv \alpha_2^-[\![\bar{A}^-]\!]$. Etc., and for all i $typ(A_i) \downarrow \alpha_i^-[\![\bar{A}^-]\!]$. Now consider, e.g., the application case: $\bar{x}\ E\ \bar{\alpha} \vdash_d \{B_1\}B_2$. By 6.3.6, $\bar{x}\ E\ \bar{\alpha}^- \vdash_c \{B_1^-\}B_2^-$ and by the subst. theorem in $\Lambda(\eta)_c$, $\vdash_c \{B_1^-[\![\bar{A}^-]\!]\}B_2^-[\![\bar{A}^-]\!] (\equiv \{B_1^{*-}\}B_2^{*-})$. By ind. hyp. $\vdash_d B_1^*$, $\vdash_d B_2^*$, so by the first lemma, $\vdash_d \{B_1^*\}B_2^*$. Similarly use the second lemma for the constant-expression case. The other cases are immediate. □

6.3.10 The remaining nice properties for $\Lambda(\eta)_d$

Corollaries of the preceding theorem are (1) correctness of types, (2) δ-outside-CL_1, (3) β-outside-CL_1 (use SA).

Lemma: $\Lambda(\eta)_d$ satisfies CL_1

Proof: The η-outside case is mere strengthening. We use the lemmas in 6.3.9 for the inside cases. Let $\vdash_d \{B_1\}B_2$, $B_1 > C_1$, $B_2 > C_2$. By ind. hyp. $\vdash_d C_1$, $\vdash_d C_2$. By 6.3.6 $\vdash_c \{B_1^-\}B_2^-$, and $B_1^- > C_1^-$, $B_2^- > C_2^-$, so $\vdash_c \{C_1^-\}C_2^-$ so $\vdash_d \{C_1\}C_2$. Similarly for const. expressions. □

Theorem: $\Lambda(\eta)_d$ satisfies CL

Proof: As usual, by ind. on $\geq$. □

Further we get the remaining UE-result:

$$\vdash_d A \Rightarrow (\vdash_d A^-,\ typ(A) \downarrow typ(A^-),\ typ^*(A) \downarrow typ^*(A^-))$$

6.4 Nederpelt's original formulation

6.4.1 Nederpelt's original definition of Λ [51] used single-line presentation. I.e. instead of defining correctness of expression relative to a context, he defined correctness of expressions having an abstractor string $[\bar{x}:\bar{\alpha}]$ (notation Q) in front.

For definiteness we give his rules. We write $\vdash_N$ for correctness in his system. But for certain provisions making sure that no confusion of variables occurs, the rules read:

(1) $\vdash_N \tau$

(2) $\vdash_N Q\alpha \Rightarrow \vdash_N Q[x:\alpha]x$

(3) $\vdash_N Q\alpha,\ \vdash_N Qy \Rightarrow \vdash_N Q[x:\alpha]y$

(4) $\vdash_N QA,\ \vdash_N QB,\ typ(QA) \geq Q\alpha,\ typ^*(QB) \geq Q[x:\alpha]C \Rightarrow \vdash_N Q\{A\}B$

6.4.2 Apart from the use of abstractor strings instead of contexts, there are two other points that make the two approaches not completely parallel. The first point concerns abstraction; our abstraction rule has no counterpart in Nederpelt's system. Nederpelt rather follows a

combinatory (in the sense of combinatory logic) way of building expressions. In the language of combinatory logic, rule (2) above is the rule for I_α, the identity in α, and rule (3) is the rule for $K_\alpha y$, the constant function on α with outcome y. Alternatively, rule (3) might be called a rule of weakening (see V.2.9.3).

6.4.3 The second point that requires attention is that an abstractor string can get involved in a reduction (notably an η-step), whereas contexts are of course immune to reduction. First some notation. We write $|Q|$ for the number of abstractors in Q. We write $Q \geq Q'$ if $Q \equiv [\bar{x}:\bar{\alpha}]$, $Q' \equiv [\bar{x}:\bar{\alpha}']$ and $\bar{\alpha} \geq \bar{\alpha}'$ in the obvious sense.
Now we have the following lemma: $QA \geq Q'A'$, $|Q| = |Q'| \Rightarrow A \geq A'$.

Proof: If there are no η-steps involving the border line between Q and A, then clearly $Q \geq Q'$, $A \geq A'$. Otherwise $Q \equiv Q_1[x:\alpha]$, $\alpha \geq \alpha'$, $Q_1 \geq Q_1'$, $A \geq \{x\}B$ with $x \notin FV(B)$ and $Q_1'B \geq Q_1''[x:\beta]A'$. I.e. $QA \equiv Q_1[x:\alpha]A \geq Q_1'[x:\alpha']\{x\}B >_\eta Q_1'B \geq Q_1''[x:\beta]A'$. Now we can, e.g., use ind. on $\theta(QA)$ and conclude that $B \geq [x:\beta]A'$. But then $A \geq A'$, q.e.d.

6.4.4 The equivalence proof

Now we are ready for the equivalence proof.

Theorem: Let

$$Q \equiv [\bar{x}:\bar{\alpha}],\ \xi \equiv \bar{x} \in \bar{\alpha}.$$

Then

$$\vdash_N QA \Leftrightarrow \xi \vdash A$$

Proof: The $\Leftarrow$-part is immediate. We use induction on $\vdash$. E.g. consider our variable rule: from $\bar{x} \in \bar{\alpha} \vdash$ we conclude $\bar{x} \in \bar{\alpha} \vdash x_i$. If x_i is the most "recent" variable then we must use rule (2). Viz. $\bar{x} \in \bar{\alpha} \vdash$ is itself a result from $x_1 \in \alpha_1, \cdots, x_{i-1} \in \alpha_{i-1} \vdash \alpha_i$. By ind. hyp. we get $\vdash_N [x_1:\alpha_1]\cdots[x_{i-1}:\alpha_{i-1}]\alpha_i$. Otherwise we must insert the abstractors inbetween $[x_i:\alpha_i]$ and the end of Q by successive applications of rule (3). Now consider the $\Rightarrow$-part. The crucial case is the application clause. So let $\vdash_N QA$, $\vdash_N QB$, $\mathrm{typ}(QA) \geq Q\alpha$, $\mathrm{typ}^*(QB) \geq Q[x:\alpha]C$. By ind. hyp $\xi \vdash A$, $\xi \vdash B$. Now $\mathrm{typ}(QA) \equiv Q\ \mathrm{typ}(A) \geq Q\alpha$ so by the lemma $\mathrm{typ}(A) \geq \alpha$. Similarly $\mathrm{typ}^*(B) \geq [x:\alpha]C$. So we conclude $\xi \vdash \{A\}B$, q.e.d.

6.4.5 The nice properties for Nederpelt's system

One of the consequences of the theorem is:

$$\vdash_N A \Leftrightarrow \vdash A$$

so the N-system can be considered a part of our system. This gives us CR and CL immediately. From this one can get the other properties SA, PD, PT etc. as usual.

6.4.6 Alternative way of embedding Λ_d into Λ_N

Resuming the results of the preceding sections: we have constructed an embedding of $\Lambda(\eta)_d$ (via $\Lambda(\eta)_c$ and Λ) into Λ_N.

Here we introduce an alternative way (due to Nederpelt [49]) of embedding $\Lambda(\eta)_d$ directly into Λ_N. Our notation for the translation is, again, '. Let a statement $\mathcal{B}; \xi \vdash_d A$ be given. Primitive schemes $\bar{x}\ \mathsf{E}\ \bar{\alpha} * p(\bar{x})\ \mathsf{E}\ \gamma$ are, as is to be expected, turned into abstractors $[p'\ \mathsf{E}\ [\bar{x}:\bar{\alpha}']\gamma']$. The context ξ is of course transformed into an abstractorstring $\xi' \equiv Q$. Essential is the translation of definitional constant schemes. A scheme $\bar{x}\ \mathsf{E}\ \bar{\alpha} * d(\bar{x}) := D * d(\bar{x})\ \mathsf{E}\ \gamma$ is translated into an expression "segment" $\{[\bar{x}:\bar{\alpha}']D'\}[d' : [\bar{x}:\bar{\alpha}']\gamma']$. All constant expressions $c(\bar{A})$ are now translated into $\{A_k'\}\cdots\{A_1'\}c'$. So $\mathcal{B}; \xi \vdash_d A$ is translated into a single expression $\mathcal{B}'\xi'A'$, where $\mathcal{B}'$ is a string of abstractors *and applicators*, and ξ' consists solely of abstractors.

For expressions the translation is quite similar to the translation ' in 6.2.1. In particular we have (as in 6.2.4) $\mathrm{typ}(A') \geq_\beta \mathrm{typ}(A)'$. However, w.r.t. to δ-reduction the correspondence is not too close: it is not possible to eliminate occurrences of d' one at a time. So in order to establish $A \downarrow B \Rightarrow A' \downarrow B'$ we need a partial δ-normal form again.

Anyhow, it is indeed possible to prove $\mathcal{B}; \xi \vdash_d A \Leftrightarrow \vdash_N \mathcal{B}'\xi'A'$.

VIII SOME RESULTS ON AUT-Pi

VIII.1 Introduction and summary

1.1 There are two languages of the Automath family that have been developed for practical (in contrast with, say, language theoretical) purposes and have actually been applied in extensive formalization projects. On the one hand there is AUT-QE, used by L.S. Jutting in his Landau translation [37]. The latter reference also contains an informal introduction to the language [27]. The theory of AUT-QE is to be found in Chs. IV to VI of this thesis. On the other hand there is AUT-Pi, invented by J. Zucker, and employed by Zucker and A. Kornaat for the formalization of classical analysis and some related topics. In [77] one finds a short account of both the language and the formalization project. This chapter is devoted to the theory of AUT-Pi, which is not quite as complete as the theory of AUT-QE. Some work remains to be done, notably on the extensional version of the language (see sec. 6).

1.2 What AUT-QE and AUT-Pi have in common

In IV.1 we described AUT-QE as a *first-order pure, regular, generalized typed* λ-calculus system. Using the same terminology, AUT-Pi is a first-order *extended*, regular, generalized typed λ-calculus system. So both languages have much in common and, in some sense, AUT-QE can be considered a sublanguage of AUT-Pi.

We resume: both languages are regular, i.e. they have just expressions of degree 1 (*supertypes*), 2 (*types* and *typevalued functions*) and 3 (*terms*). They are first-order, i.e.there is only quantification and λ-abstraction over term variables, not over type-variables. Further, they have generalized type structure, i.e. the types are constructed along with the terms. Besides, AUT-Pi and AUT-QE have the *book-and-context* structure in common. Books to introduce primitive and defined constants, depending on variables, for which substitution (instantiation) is permitted. Contexts for the introduction of variables.

Here we want to emphasize that, just like AUT-QE, AUT-Pi is a non-arithmetical system, i.e. it has no recursion constant with the corresponding reduction.

1.3 The additional operations of AUT-Pi

But, where AUT-QE belongs to pure typed λ-calculus (abstraction, application and instantiation as the only term-forming operations), AUT-Pi is a typical extended system, with the additional kinds of terms: *pairs* $\langle P,A,B\rangle$, *projections* $A_{(1)}$ and $A_{(2)}$, *injections* $i_1(A,\beta)$ and $i_2(B,\alpha)$ and ⊕-functions (or: ⊕-terms) $A \oplus B$. Here the P of the pair, and the β and α of the injections are mere type-labels to guarantee uniqueness of types.

Corresponding with these new terms there are new type-constructs: first the *sum-type* ΣP containing the pairs $\langle P,A,B\rangle$ as elements, where P is a type-valued function with domain α, A belongs to α and B is of type $\{A\}P$. In case P (as a type-valued function) is constant, i.e. $\{A\}P$ does not depend on A, the pair and the sum type can be considered to degenerate to $\langle A,B\rangle$ and $\alpha \otimes \beta$ respectively, where ⊗ is the ordinary cartesian product and β is the type of B. Secondly, there is the *disjoint union* or ⊕-*type* $\alpha \oplus \beta$, containing the injections $i_1(A,\beta)$ and $i_2(B,\alpha)$, where A and B are of types α and β respectively.

The pairs get their meaning by the presence of the projections and the associated reductions: if A is a pair, i.e. element of a sum-type, say ΣP, then $A_{(1)}$ is an element of the domain of P and $A_{(2)}$ is element of $\{A_{(1)}\}P$. Now $\langle P,A,B\rangle_{(1)}$ *π-reduces* to A and $\langle P,A,B\rangle_{(2)}$ *π-reduces* to B. In the extensional version of AUT-Pi, $\langle P,A_{(1)},A_{(2)}\rangle$ *σ-reduces* to A, provided A belongs to ΣP (otherwise the type would vary under reduction).

Similarly, the injections get their meaning by the ⊕-terms and the associated reduction. Let us first explain what a ⊕-term is. Roughly speaking, when f is a function on α and g is a function on β, then - under certain conditions - $f \oplus g$ is a function defined on $\alpha \oplus \beta$, acting on (injections of terms of type) α as f and on (injections of terms of type) β like g. So the reductions are as follows: $\{i_1(A,\beta)\}(f \oplus g)$ +-reduces to $\{A\}f$ and $\{i_2(B,\alpha)\}(f \oplus g)$ +-reduces to $\{B\}g$. The corresponding extensional reduction is ε-*reduction*: $[x{:}\alpha]\{i_1(x)\}f \oplus [x{:}\beta]\{i_2(x)\}f$ ε-*reduces* to f, provided f does not contain x as a free variable (i.e. does not depend on x).

Please note the use of parentheses: ⊕ is supposed to bind more loosely than the other term forming operations.

A more precise definition of AUT-Pi follows in sec. 2.

1.4 The connection with natural deduction systems

By the well-known formulae-as-types, derivations-as-terms interpretation, systems of typed λ-calculus can be brought into close correspondence with certain natural deduction systems for intuitionistic logic (including the usual proof theoretic reduction relations). Thus, pure systems correspond to logical systems with → and ∀ only, and extended systems correspond to systems with more connectives. In particular, the Σ, the pairs and the projections of AUT-Pi may provide the interpretation of "strong" existential quantification with its introduction and elimination rules (though this has not been exploited in Zucker's book, see [77]). And ⊗, the degenerate form of Σ, corresponds precisely to conjunction.

As for the interpretation of ∨ (disjunction) by ⊕-types, the introduction rules of ∨ do correspond to injection, but the elimination rule of ∨ differs slightly from its counterpart in AUT-Pi. The usual elimination rule of ∨ (see, e.g., Prawitz [59]) operates on three arguments: from (1) a derivation of $\alpha \vee \beta$, (2) a derivation of γ under the assumption α, (3) a derivation from γ under the assumption β, one can form a derivation with conclusion γ. The assumptions α and β of the derivations (2) and (3) are discharged.

The AUT-Pi operation representing this rule must be constructed in several steps: first (2) and (3) are transformed into derivations of $\alpha \to \gamma$ and $\beta \to \gamma$ respectively. These two derivations are combined into a derivation of $(\alpha \vee \beta) \to \gamma$ (by using ⊕). Then the conclusion γ follows from modus ponens (by (1)).

Here we stick to the AUT-Pi variant of the rule. For a discussion of the alternatives see Pottinger [56, 57].

Because AUT-Pi is still non-arithmetical, it cannot represent natural-deduction systems for arithmetic (in the sense intended above).

1.5 Product formation versus type inclusion

Now we discuss a specific difference between AUT-QE and AUT-Pi, that prevents AUT-QE from being an actual sublanguage of AUT-Pi. In AUT-QE there is no difference in notation between type-valued functions and function types. I.e. the expression $[x{:}\alpha]\beta$, with β an expression of degree 2, stands for the function that to arguments A in α assigns

types $\beta[\![A]\!]$, but also for the type of the functions which, when applied to A in α, produce a value in $\beta[\![A]\!]$. And, to make things even more complicated, it is possible that β allows such multiple interpretations as well.

In AUT-Pi there is reserved a special symbol for referring to the function type, viz. Π (for cartesian *product* formation): by prefixing with Π the type-valued function $[x{:}\alpha]\beta$ is turned into the corresponding function type $\Pi[x{:}\alpha]\beta$. More general, if P is a type-valued function, then ΠP is the corresponding *product type*, containing those functions as elements which, when applied to arguments A of the right type, produce values in $\{A\}P$.

The language AUT-Pi is named after the Π of product formation.

In AUT-QE the expression $[x{:}\alpha]\beta$ can get (at least) two possible types, viz. $[x{:}\alpha]\tau$ and τ, according to which interpretation is intended. This is implemented by the rule of type inclusion. As a consequence, *uniqueness of types* is valid for terms only. Some problems arise from this in connection with defined constants (see V.1.9 and V.3.3.10). In AUT-Pi uniqueness of types is valid for types as well: e.g. if β is a type, then $[x{:}\alpha]\beta$ has type $\Pi[x{:}\alpha]\tau$ and $\Pi[x{:}\alpha]\beta$ has type τ.

Note here the use of Π again which makes the (constant) "super-type valued function" $[x{:}\alpha]\tau$ into a super-type $\Pi[x{:}\alpha]\tau$.

At first sight it seems that the here-indicated difference is a trifle, and that AUT-QE can be made into a subsystem of AUT-Pi by simply inserting Π's at the right places. However, as noted by the Bruijn, the correspondence is not that close: the rule of type-inclusion (of AUT-QE) is somewhat stronger than the product formation rule (of AUT-Pi). See sec. 6.1, [15] and [17].

1.6 Some features of AUT-Pi not discussed here

For completeness we mention two important, more or less syntactical, features that enrich the language used by Zucker and Kornaat in their AUT-Pi book. First, there is the use of AUT-Synt, a kind of Automath shorthand, as documented in Jutting [37]. Secondly, there is the use of *strings-and-telescopes* (see [77]).

However, these features do not belong specifically to AUT-Pi; they rather can be attached to any Automath language, but were not yet available when Jutting started his Landau translation. On the contrary, the

strings-and telescopes generalize (and, hence, *duplicate*) in some sense the pairs-and-sums of AUT-Pi. These two features are not discussed in this thesis.

In [77] Zucker describes how the whole language is divided into a t-part (for terms and types) and a p-part (for proofs and propositions). This division originates with the distinction between the two degree 1 basic constants, τ (or type) and π (or prop). Connected with this is the principle of *equality of proofs* (two proofs of the same proposition are considered to be definitionally equal; only consistent with classical logic). Here we just use τ as our basic constant. As a consequence we do not discuss equality of proofs.

1.7 Section 2 below contains a more precise definition of AUT-Pi. In section 3 we prove the closure property: Correctness is preserved under reduction. In section 4 we first define two systems of normable expressions, AUT-Pi_0 and AUT-Pi_1, which have the same "connectives" and reductions as AUT-Pi but a simplified type structure. We study SN for these two systems. First we show that the methods of proving β-SN directly apply to the situation with $\beta\pi$-reduction. In sec. 5 we give some different proof methods for SN in presence of +-reduction. Then we extend the AUT-Pi_1 results to AUT-Pi. Section 6 just contains some remarks on the connection between AUT-Pi and AUT-QE (type-inclusion vs. product formation), and on the particular problems posed by the addition of ε-reduction.

VIII.2 A short definition of AUT-Pi

2.1.1 We give an E-definition of AUT-Pi, along the lines of the AUT-QE definition in V.2. For the formation of *books* and *contexts* we refer to IV.3, and for their correctness to the requirements in V.2.1.3. However, the *inhabitable degree condition*, to the effect that correct expressions can be of degree 1, 2 and 3 only, has to be restricted further, to an *inhabitability condition*: Expressions acting as the typ of a variable or a constant have to be *inhabitable*. Where we define α to be *inhabitable* when degree(α) = 1, or: degree(α) = 2 and α E τ (or α E π).

2.1.2 But first we must define the degree (and, implicitly, the notion of degree correctness) of the typical AUT-Pi expressions:

$$\mathrm{degree}(A) = 1 \text{ or } 2 \Rightarrow \mathrm{degree}(\Pi(A)) = \mathrm{degree}(A)$$

$$\mathrm{degree}(A) = 2 \Rightarrow \mathrm{degree}(\Sigma(A)) = 2$$

$$\mathrm{degree}(A) = 3 \Rightarrow \mathrm{degree}(A_{(1)}) = 3,\ \mathrm{degree}(A_{(2)}) = 3$$

$$\mathrm{degree}(A) = \mathrm{degree}(B) = 2 \text{ or } 3 \Rightarrow \mathrm{degree}(A \oplus B) = \mathrm{degree}(A)$$

$$\mathrm{degree}(A) = 3,\ \mathrm{degree}(B) = 2 \Rightarrow \mathrm{degree}(i_1(A,B)) = \mathrm{degree}(i_2(A,B)) = 3$$

$$\mathrm{degree}(A) = 2,\ \mathrm{degree}(B) = \mathrm{degree}(C) = 3 \Rightarrow \mathrm{degree}(\langle A,B,C\rangle) = 3$$

2.1.3 Correctness of expressions, E-formulas (for typing) and Q-formulas (for equality) is defined simultaneously. For the notational conventions and abbreviations we refer to V.2.1 and V.2.2. E.g., we display degrees as superscripts to the correctness symbol $\vdash$, we freely omit books and contexts (or parts of contexts) not relevant to the rule under consideration, and we sometimes omit $\vdash$ as well (viz. in front of a formula when context and degree are not shown).

2.2 The general rules

2.2.1 We start with the rules, which AUT-Pi has in common with AUT-QE. We assume a correct book $\mathcal{B}$ and a correct context ξ. First the general rules for correctness of expressions and E-formulas.

(i) type and prop: $\vdash^1 \tau$ and $\vdash^1 \pi$

(ii) variables: $\cdots, x\ \mathsf{E}\ \alpha, \cdots \vdash x(\mathsf{E}\alpha)$

(iii) instantiation: if c is introduced in $\mathcal{B}$, with context $\bar{y}\ \mathsf{E}\ \bar{\beta}$, then $\bar{B}\ \mathsf{E}\ \bar{\beta}[\![\bar{B}]\!] \Rightarrow c(\bar{B})\ (\mathsf{E}\ \mathrm{typ}(c)[\![\bar{B}]\!])$

For our language theoretical purposes we need not distinguish between τ and π. So in the sequel we just use τ, intending to cover π as well.

2.2.2 Then the remaining general rules: for Q, for type-modification and strengthening.

(iv) Q-reflexivity: $\vdash A \Rightarrow A\ Q\ A$

(v) Q-propagation: $A\ Q\ B, \vdash C,\ (B > C \text{ or } C > B) \Rightarrow A\ Q\ C$

(vi) type-conversion: $A\ E\ B\ Q\ C \Rightarrow A\ E\ C$

(vii) strengthening: if $(x\ E\ \alpha,\eta)\vdash B\ (E/Q\ C)$, x does not occur free in $\eta(,C)$ and B then $\eta\vdash B\ (E/Q\ C)$

The Q-propagation rule still depends on an assumed reduction relation, e.g. either with or without the extensional reductions η, ε, σ. The rule of strengthening is only included for technical reasons associated with η and ε, so can be omitted in the non-extensional case.

Notice that the rule of type-inclusion of AUT-QE has been left out here. Its role, viz. of transforming (type-valued) functions into types, is to be played here by the product rule for 2-expressions of the next section.

2.3 The specific rules I

Now we come to the rules specific for AUT-Pi. They are divided into three groups. Each consists of one (or more) introduction rule(s) one (or more) elimination rule(s) and a type formation rule to provide the introduction expression(s) with a type. With each group an IE-reduction rule (i.e. introduction-elimination reduction rule) and its extensional counter part can be associated.

I Abstraction, application and products

1. Product rule 1: $x\ E\ \alpha\vdash^1 B \Rightarrow \vdash^1 \Pi([x:\alpha]B)$
2. Product rule 2: $B\ E\ \Pi([x:\alpha]\tau) \Rightarrow \Pi(B)\ E\ \tau$
3. Abstr. rule: $\vdash^2\alpha,\ x\ E\ \alpha\vdash^{i+1} B(E\ C) \Rightarrow \vdash^{i+1}[x:\alpha]B(E\ \Pi([x:\alpha]C))$
4. Appl. rule 1: $A\ E\ \alpha,\ \vdash^2 B\ E\ \Pi([x:\alpha]\beta)\Rightarrow\ \vdash^2\{A\}B(E\ \beta[\![A]\!])$
5. Appl. rule 2: $A\ E\ \alpha,\ B\ E\ \Pi(C),\ C\ E\ \Pi([x:\alpha]\tau) \Rightarrow \{A\}B(E\ \{A\}C)$

The associated reduction relations are β and η:

$$\{A\}[x:\alpha]B >_\beta B[\![A]\!],\quad [x:\alpha]\{x\}A >_\eta A \quad \text{if} \quad x \notin FV(A)$$

It is in the above group of rules that the difference between AUT-QE and AUT-Pi becomes explicit. For a discussion of the rule of Π see 1.5, and 6.1.

Notation: In case $x \notin FV(B)$ we abbreviate $\Pi([x{:}\alpha]B)$ by $\alpha \to B$.

Using this convention, product rule 2 and appl rule 2 become

$$B \mathrel{E} \alpha \to \tau \Rightarrow \Pi(B) \mathrel{E} \tau$$

and

$$A \mathrel{E} \alpha,\ B \mathrel{E} \Pi(C),\ C \mathrel{E} \alpha \to \tau \Rightarrow \{A\}B \mathrel{E} \{A\}C$$

2.4 A possible extension concerning 1-expressions

Notice that all compound correct 1-expressions have a Π in front, or possibly (when 1-abbreviation constants are present) δ-reduce to an expression starting with Π. In fact, each correct 1-expression δ-reduces to an expression like $\Pi([x_1{:}\alpha_1]\Pi([x_2{:}\alpha_2]\Pi(\cdots\Pi([x_n{:}\alpha_n]\tau)\cdots)))$.

As a consequence all 1-expressions are inhabitable (see 2.1), just like in AUT-QE, but they generally contain parts which are *not correct*, e.g. the part $[x{:}\alpha]\tau$ in $\Pi([x{:}\alpha]\tau)$. If we do not like this we can easily extend the language by

(1) restricting the notion of inhabitable 1-expressions: 1-expressions are said to be inhabitable according to: (i) τ inhabitable, (ii) (ii) if B inhabitable then $\Pi([x{:}\alpha]B)$ inhabitable, (iii) if B inhabitable, $B \mathrel{Q} C$ then C inhabitable.

(2) restricting product rule 1:

$$x \mathrel{E} \alpha \vdash^1 B,\ B \text{ inhabitable} \Rightarrow \vdash \Pi([x{:}\alpha]B)$$

(3) dropping the restriction to degree i + 1 in the abstr rule. Then, we can further extend AUT-Pi to a +-language (i.e. all value degrees are also function degrees, see V.2.7) by

(4) adding a new appl rule:

$$A \mathrel{E} \alpha,\ B \mathrel{Q} [x{:}\alpha]C \Rightarrow \vdash \{A\}B$$

These changes are relatively unimportant, of course.

2.5 The specific rules II

2.5.1 The rules of group I can be considered as just rephrasing the corresponding rules of AUT-QE. Now, however, we come to rules which have no counterpart in AUT-QE.

II Pairs, projections, sums

Let $\phi \mathrel{E} \alpha \to \tau$. Then

1. Sum rule: $\vdash \Sigma(\phi)\,(E\ \tau)$
2. Pair rule: $A \mathrel{E} \alpha,\ B \mathrel{E} \{A\}\phi \Rightarrow \vdash <\phi,A,B>(E\ \Sigma(\phi))$
3. Projection rules: $C \mathrel{E} \Sigma(\phi) \Rightarrow \vdash C_{(1)}\,(E\ \alpha),\ \vdash C_{(2)}\,(E\ \{C_{(1)}\}\phi)$

The reduction rules associated with group II are π and σ:

$$<\phi,A,B>_{(1)} >_{\pi} A,\quad <\phi,A,B>_{(2)} >_{\pi} B$$

$$A \mathrel{E} \Sigma(\phi) \Rightarrow <\phi,A_{(1)},A_{(2)}> >_{\sigma} A$$

2.5.2 Notice that here, for the first time, reduction ceases to be a purely syntactical matter. The condition $A \mathrel{E} \Sigma(\phi)$ is inserted here because we want to maintain *preservation of types*

$$A \mathrel{E} \alpha,\ A > B \Rightarrow B \mathrel{E} \alpha$$

Otherwise, we come in trouble with $\phi \mathrel{E} \alpha \to \tau$, $A \mathrel{E} \alpha$, $\psi \equiv [x{:}\alpha]\{A\}\phi$, $B \mathrel{E} \{A\}\phi$, where $C \equiv <\phi,A,B> \mathrel{E} \Sigma(\phi)$ and $<\psi,C_{(1)},C_{(2)}> \mathrel{E} \Sigma(\psi)$ and not $\phi\ Q\ \psi$.

As a consequence we must modify one of the monotonicity rules into: if $x \mathrel{E} \alpha \Rightarrow A > B$ then $[x{:}\alpha]A > [x{:}\alpha]B$.

2.5.3 Notation: in case $x \notin FV(\beta)$ we abbreviate $\Sigma([x{:}\alpha]\beta)$ by $\alpha \otimes \beta$. For pairs $<\phi,A,B>$ in such a degenerate sum we can omit the type label ϕ and just write $<A,B>$ (because it is intended that ϕ can be constructed from A and B in this case).

The degenerate versions of pair rule and projection rules are:

$$A \mathrel{E} \alpha,\ B \mathrel{E} \beta \Rightarrow <A,B> \mathrel{E} \alpha \otimes \beta$$

$$C \mathrel{E} \alpha \otimes \beta \Rightarrow C_{(1)} \mathrel{E} \alpha,\ C_{(2)} \mathrel{E} \beta$$

For degenerate pairs the typing condition for σ-reduction can be omitted.

Notice that, in contrast with products, only degree 2 sums are formed, and consequently only degree 3 pairs. Besides, the two components of a pair are 3-expressions too.

2.6 The specific rules III

See the discussion in 1.4. The rules concern

III Binary unions, injections and plus-terms

Let $\alpha \mathrel{E} \tau$, $\beta \mathrel{E} \tau$. Then

1. Binary union: $\vdash \alpha \oplus \beta (E\ \tau)$
2. Injection 1: $A \mathrel{E} \alpha \Rightarrow \vdash i_1(A,\beta)(E\ \alpha \oplus \beta)$
3. Injection 2: $B \mathrel{E} \beta \Rightarrow \vdash i_2(B,\alpha)(E\ \alpha \oplus \beta)$
4. Plus rule: $\gamma \mathrel{E} \tau,\ B \mathrel{E} \alpha \to \gamma,\ C \mathrel{E} \beta \to \gamma \Rightarrow \vdash B \oplus C(E(\alpha \oplus \beta) \to \gamma)$

The associate reductions are + and ε:

$$\{i_1(\beta,A)\}(C \oplus D) >_+ \{A\}C,\ \{i_2(B,\alpha)\}(C \oplus D) >_+ \{A\}D$$

$$[x{:}\alpha]\{i_1(x)\}F \oplus [x{:}\beta]\{i_2(x)\}F >_\varepsilon F \quad \text{if} \quad x \notin \mathrm{FV}(F).$$

Notation: ⊕ is supposed to bind more loosely than the other connectives. This is why the function parts of the +-redices *are*, and the left- and right part of the ε-redex are *not* put inside parentheses.

We mention also the alternative form of +, +' (which is in fact + followed by β):

$$\{i_1(A,\beta)\}\ [x{:}\alpha]B \oplus C) >_{+'} B[\![A]\!]$$

and an alternative form of ε, εalt:

$$[x{:}\alpha]B[\![y/i_1(x,\beta)]\!] \oplus [x{:}\beta]B[\![y/i_2(x,\alpha)]\!] >_{\varepsilon\mathrm{alt}} [y{:}\alpha \oplus \beta]B$$

We clearly have $>_{+'} \Rightarrow >_+ >_\beta$ (see II.7.1.2 for the notation). Further

$$\{i_1(A,\beta)\}(B \oplus C) <_\eta \{i_1(A,\beta)\}([x{:}\alpha]\{x\}B \oplus C) >_{+'} \{A\}B$$

etc. i.e. $>_+ \Rightarrow <_\eta >_{+'}$. So, as far as equality Q is concerned, we have (in the sense of II.0.4.3) $(\beta,+ \Rightarrow +')$ and $(\eta,+' \Rightarrow +)$. Since we always include β, and η is optional, we prefer the rule + in our definition.

Similarly we have $>_{\varepsilon} \Rightarrow >_{\varepsilon alt} >_{\eta}$ and $>_{\varepsilon alt} \Rightarrow >_{\beta} >_{\beta} <_{\varepsilon}$, so (w.r.t. Q) $(\eta, \varepsilon alt \Rightarrow \varepsilon)$ and $(\beta, \varepsilon \Rightarrow \varepsilon alt)$. Thus we prefer rule ε.

Binary unions always have degree 2, injections always have degree 3. Only $\oplus$-functions of degree 3 are formed.

2.7 A possible extension concerning $\oplus$-functions

We can, however, define an extension of the language by also admitting degree 2 $\oplus$-functions, i.e. glueing type-valued functions together into a single type-valued function. To this end we put: Let α E τ, β E τ. Let ϕ E $\alpha \to \tau$, ψ E $\beta \to \tau$. Then

4'. Plus rule 1: $\vdash \phi \oplus \psi(E(\alpha \oplus \beta) \to \tau)$

5. Plus rule 2: B E $\Pi(\phi)$, C E $\Pi(\psi) \Rightarrow \vdash B \oplus C(E\ \Pi(\phi \oplus \psi))$

The old plus can be considered as a special case of rule 5, by using ε or εalt:

$$[x{:}\alpha]\gamma \oplus [x{:}\beta]\gamma >_{\varepsilon alt} [x{:}\alpha \oplus \beta]\gamma$$

We *do not discuss* this extension here, because it really complicates the normability problem (see 4.6).

2.8 Elementary properties

As in V.2.7 - V.2.9 we can infer some nice properties. First, concerning the degrees:

$\vdash A \Rightarrow A$ degree correct

A Q $B \Rightarrow$ degree(A) = degree(B)

A E $B \Rightarrow$ degree(A) = degree(B) + 1

Then, concerning contexts, *renaming* (see V.2.9.2) and *weakening* (V.2.9.3). Further, the *simultaneous* and the *single substitution theorem* (V.2.9.4-5), and *correctness of categories* (V.2.10): A E $B \Rightarrow \vdash B$.

Analogously to the abstr and appl properties in V.2.10 and V.2.1 (which m.m. hold as well in AUT-Pi) we have properties like

$\vdash \langle \phi, A, B \rangle \Rightarrow (A$ E α, ϕ E $\alpha \to \tau$, B E $\{A\}\phi)$ etc.

i.e. the "inversion of the correctness rules".

An important additional property (to be proved in the next section) is *uniqueness of types*:

$$A \mathrel{E} B,\ A \mathrel{E} C \Rightarrow B \mathrel{Q} C$$

which in AUT-QE did not hold for A of degree 2, because of type inclusion.

VII.3 A short proof of closure for AUT-Pi

3.1 Proving closure for AUT-Pi is not very different from proving it for AUT-QE. So we just sketch how to modify the proof in V.3.2.

We start with a version without the extensions mentioned in 2.4 and 2.7, but we include all reductions (also δ^1-reduction).

3.2 For the terminology see V.3.1. Let $>$ denote disjoint more step reduction. By the properties in II.7.4.3 we have

$$A > B \Rightarrow \delta\text{-nf}(A) > \delta\text{-nf}(B)$$

By the substitution theorem we have δ-CLPT. The δ-nf's of 1-expressions are of the form $\Pi([x{:}\alpha]A)$ or τ. Reductions of these expressions can only be internal, so by induction on Q we get (including what might be called UD^1 here):

$$\vdash^1 \Pi([x{:}\alpha]A) \mathrel{Q} \Pi([x{:}\beta]B) \Rightarrow \alpha \mathrel{Q} \beta \quad \text{and} \quad (x \mathrel{E} \alpha \vdash A \mathrel{Q} B)$$

3.3 From this follows SA^2 (whence β-outside-CL_1^2) and β-outside-PT_1^2. Viz. let $A \mathrel{E} \alpha$, $\vdash^2 [x{:}B]C \mathrel{E} \Pi([x{:}\alpha]D)$, with conclusion $\vdash \{A\}[x{:}B]C$. Then, for some E, $x \mathrel{E} B \vdash C \mathrel{E} E$ and $\vdash \Pi([x{:}B]E) \mathrel{Q} \Pi([x{:}\alpha]D)$. So $\alpha \mathrel{Q} B$ and $x \mathrel{E} B \vdash E \mathrel{Q} D$ whence $A \mathrel{E} B$ (i.e. SA^2) and $x \mathrel{E} B \vdash C \mathrel{E} D$. So

$$C[\![A]\!] \mathrel{E} D[\![A]\!] \quad \text{(i.e. } \beta\text{-outside-CLPT}_1^2\text{)}.$$

The proofs of UT^2 and the inside cases of PT_1^2 are by ind. on $\vdash$.

3.4 The strengthening rule gives η-outside-CL_1. Here follows a proof of η-outside-PT_1^2 different from the proof in V.3.2.5. Viz. let $\vdash^2[x{:}\alpha]\{x\}A \mathrel{E} \gamma$, $x \notin FV(A)$. Then, for some C, $[x{:}\alpha]\{x\}A \mathrel{E} \Pi([x{:}\alpha]C[\![y/x]\!]) \mathrel{Q} \gamma$, where $x \mathrel{E} \alpha \vdash A \mathrel{E} \Pi([y{:}\alpha']C)$, $\alpha' \mathrel{Q} \alpha$. So, as well, $x \mathrel{E} \alpha \vdash A \mathrel{E} \Pi([y{:}\alpha]C)$. By weakening $x \mathrel{E} \alpha$, $y \mathrel{E} \alpha \vdash A \mathrel{E} \Pi([y{:}\alpha]C)$ and $x \mathrel{E} \alpha$, $y \mathrel{E} \alpha \vdash \{y\}A \mathrel{E} C$ so $x \mathrel{E} \alpha \vdash [y{:}\alpha]\{y\}A \mathrel{E} \Pi([y{:}\alpha]C)$. Again by weakening $x \mathrel{E} \alpha \vdash [x{:}\alpha]\{x\}A \mathrel{E} \gamma$, so by UT^2 $x \mathrel{E} \alpha \vdash \gamma \mathrel{Q} \Pi([y{:}\alpha]C)$. Hence $x \mathrel{E} \alpha \vdash A \mathrel{E} \gamma$ and by strengthening $A \mathrel{E} \gamma$, q.e.d.

3.5 This completes the proof of PT_1^2. Then PT^2 and LQ^2 follow by ind. on $\geq$ and Q respectively. Now we come to PTCL^3. For properties like SA^3 we need

$$\Sigma(\phi) \mathrel{Q} \Sigma(\psi) \Rightarrow \phi \mathrel{Q} \psi$$

$$\Pi(\phi) \mathrel{Q} \Pi(\psi) \Rightarrow \phi \mathrel{Q} \psi$$

$$(\alpha \oplus \beta) \mathrel{Q} (\gamma \oplus \delta) \Rightarrow \alpha \mathrel{Q} \gamma,\ \beta \mathrel{Q} \delta$$

3.6 To this end we study β^2-reduction and, in particular, β^2-head-reduction, for short β_h^2 (for the definitions see V.3.3.3 and V.4.4.5). We know already β^2-outside-CLPT_1 (this is β-outside-CLPT_1^2). From this follows β^2-CLPT_1 by ind. on $\vdash$, and β^2-CLPT by ind. on $\geq$. Now we use the fact that 3 is the only argument degree and that, hence, β^2-reduction does not create new β^2-redices. Compare V.3.3.4, VI.2.4.

As a consequence, β^2-SN is quite easily provable (for degree correct expressions) even without using norms: namely, if A β^2-SN, B β^2-SN then $A[\![B]\!]$ β^2-SN, by ind. on (1) $\theta_\beta^2(B)$, (2) length (B). So, as usual, β^2-SN by ind. on length (see IV.2.4.1). A fortiori, β_h^2-SN.

Besides β_h^2 satisfies CR, so we can speak about β_h^2-nf's. E.g.,

$$\text{degree}(B) = 2,\ \beta_h^2\text{-nf}(B) \equiv [x{:}\alpha]C \Rightarrow \beta_h^2\text{-nf}(\{A\}B) \equiv C[\![A]\!]$$

Clearly β_h^2 and δ commute, so $\beta_h^2\delta$-CR and $\beta_h^2\delta$-nf's are defined too.

3.7 Theorem: $\vdash^2 A \mathrel{Q} \Sigma(\phi) \Rightarrow \beta_h^2\delta\text{-nf}(A) \equiv \Sigma(\psi),\ \phi \mathrel{Q} \psi$

Sketch of proof: Ind. on Q. For the induction step we need the following property: $\vdash^2 A$, $\beta_h^2\delta\text{-nf}(A) \equiv \Sigma(\phi)$, $A > C$ or $C > A$, $\vdash C \Rightarrow \beta_h^2\delta\text{-nf}(C) \equiv \Sigma(\psi)$, $\phi \mathrel{Q} \psi$. If $C > A$ it is eacy, (β_h^2)-i-pp holds here for all kinds of

reduction i (see II.7.3), so $\beta^2_h\delta$-nf$(C) \equiv \Sigma(\psi)$, $\psi > \phi$. Otherwise, $A > C$. Now $\beta^2_h\delta$ commutes with all other kinds of reduction, except η^2_k (see II.7.2). And it even commutes with the latter, except for "outside" domains. Where we define the latter to be the α_i, β_j, etc. in $\{\bar{A}\}[\bar{x}:\bar{\alpha}]\{\bar{B}\}[\bar{y}:\bar{\beta}]\cdots$, with $\{\bar{A}\}$ possibly empty. But there are no "outside" domains left in $\Sigma(\phi)$. So, in any case, $\beta^2_h\delta$-nf$(C) \equiv \Sigma(\psi)$, $\phi > \psi$. In fact, if $A >^2_{\eta h} C$ then $\phi \equiv \psi$.

By $\beta^2_h\delta$-CL we know that both $\Sigma(\phi)$ and $\Sigma(\psi)$ are correct so from ($\phi > \psi$ or $\psi > \phi$) we can conclude ϕ Q ψ. This proves the wanted property. □

Corollary: $\Sigma(\phi)$ Q $\Sigma(\psi) \Rightarrow \phi$ Q ψ

3.8 Both the theorem and the corollary can be proved in precisely the same manner for Π and $\oplus$, yielding the properties in 3.5.

Remark: The theorem above is a kind of minimal result for the desired properties. E.g., we can, alternatively, prove a kind of weak CR^2-result as in VI.2.4, or prove a similar but stronger theorem in the spirit of V.3.3, V.3.4.

3.9 Now we are able to prove the outside cases of $CLPT^3_1$. E.g. for +-reduction. Let $\{i_1(A,\beta)\}(F \oplus G)$ E γ. Then $i_1(A,\beta)$ E δ, $F \oplus G$ E $\Pi(\phi)$, ϕ E $\delta \to \tau$, $\{i_1(A,\beta)\}\phi$ Q γ. And A E α, $\alpha \oplus \beta$ Q δ, F E $\alpha' \to \gamma'$, G E $\beta' \to \gamma'$, $(\alpha' \oplus \beta') \to \gamma'$ Q $\Pi(\phi)$. So $[x:\alpha' \oplus \beta']\gamma'$ Q ϕ, and $[x:\alpha' \oplus \beta']\gamma'$ E $\delta \to \tau$. So $(\alpha' \oplus \beta')$ Q δ Q$(\alpha \oplus \beta)$, whence α Q α', β Q β'. So $\{A\}F$ E γ'. Further γ' Q $\{i_1(A,\beta)\}[x:\alpha' \oplus \beta']\gamma'$ Q $\{i_1(A,\beta)\}\phi$ Q γ, whence $\{A\}F$ E γ too. Similarly for the other variant of +.

3.10 Then follows full $CLPT_1$ by ind. on $\vdash$ and CLPT by ind. on $\geq$. Besides, we have of course UT and LQ. And we can freely make the language definition somewhat more liberal, as follows.

First we can change the Q-propagation rule into

$$A \text{ Q } B,\ B \twoheadrightarrow C,\ \vdash C \Rightarrow A \text{ Q } C$$

Secondly we can add the appl rule, with i ≥ 1

$$A \text{ E } \alpha,\ \vdash^{i+1} B \text{ Q } [x:\alpha]C \Rightarrow \vdash \{A\}B$$

and drop the degree restriction in the appl rule 1 (i.e. rule I.4).

3.11 Now we shall say something about proving CL for AUT-Pi with the extension of sec. 2.4. Just adding abstr expressions of degree 1 does not matter at all, we still can get UD^1 without any difficulty.

Making the language into a +-language (i.e. adding appl-1-expressions too) causes some trouble with the domains in case η reduction is present. Which can however be circumvented as in V.3.3: First leave η^1 out, then prove β^1-CL and add η^1 again.

3.12 Finally the extension of sec. 2.7, i.e. where $\oplus$-2-expressions are present. If there is also ε^2-reduction the situation is essentially more complicated, because β and ε interfere nastily. But without ε^2 the proofs of 3.3-3.8 just need some modification: $(\beta+)^2$-SN can be proved as easy as β^2-SN, $+^2$-CLPT is not difficult either. Then theorem 3.7 can be proved for $(\beta+)^2$-δ-head-nf's instead.

3.13 Requirements for the pp-results in II.9 were:

(1) The result of outside-δ-reduction is never a $\oplus$-, an inj- or on abstr-expression

(2) The result of outside η or ε is never an inj-expression or a pair.

Now we can easily verify them for AUT-Pi using the results of this section. First let $\langle \phi, A_{(1)}, A_{(2)} \rangle >_\delta A$. I.e. degree$(A) = 3$, $A \in \Sigma(\phi)$. If A were an abstr-term then $A \in \Pi(\psi)$ for some ψ. UT states that $\Pi(\psi) \subseteq \Sigma(\phi)$. Theorem 3.7 states that $\Pi(\psi) \geq \Sigma(\chi)$ for some χ. This is impossible. Similarly for inj- or $\oplus$-expressions. Or let $[x{:}\alpha]\{x\}A >_\eta A$. By PT $A \in \Pi(\phi)$ for some ϕ. If A were an inj-expression then degree$(A) = 3$, $A \in (\beta \oplus \gamma)$ for some β, γ. By UT $\Pi(\phi) \subseteq (\beta \oplus \gamma)$. Use the suitable variant of theorem 3.7 again (sec. 3.8), this gives a contradiction.

VIII.4 A first SN-result for an extended system

4.1 Introduction

The word "extended" in the title of this section refers to the presence of other formation rules than just abstr and appl (and possibly instantiation) and other reduction rules than just β and η (and possibly

δ). In the case of AUT-Pi we are concerned with the additional presence of:

(1) pairs and projections, with reductions π and σ

(2) injections and $\oplus$-terms, with reductions + and ε

In IV.2.4 we gave some versions of a "simple" (as compared to a proof using computability) proof of β-SN. Then we extended it to $\beta\eta$ using $\beta\eta$-pp. Afterwards we included δ as well.

Here we stick to the separation of δ from the other reduction rules. Below we first show (4.6) that addition (1) mentioned above does not cause any trouble: the first version of the "simple" proof of β-SN immediately covers the $\beta\pi$-case. And afterwards, we can include δ and η by a postponement result again.

However the second addition essentially complicates matters. The presence of + makes the first β-SN proof fail here, because the important induction on functional complexity (norm) goes wrong.(see sec. 5.1.2). We add new, socalled *permutative reductions* (sec. 4.3.1, III) in order to save the idea of the proof (5.1.3). These permutative reductions, in turn, complicate the SN-condition, and a way to keep them manageable consists of adding (in 5.1.5) still another kind of reduction, viz. *improper* reductions (sec. 4.3.1, IV).

Our second β-SN proof of Ch. IV can fairly easy be adapted for the present situation however. We just have to add improper reductions to make the proof work (see sec. 5.2). For completeness we also include a proof based on the computability method (sec. 5.3).

However, these three proofs just cover the situation with $\beta + \pi$-reduction and can, by ext-pp be extended to $\beta + \pi\delta\eta$. Alas, we *have not been able to handle* ε too. We cannot use pp anymore, so we have to include ε from the start of the proof on. And none of our methods can cope with this situation.

The problems with $\oplus$ (or $\vee$) are well-known from proof theory. E.g. Prawitz in [59] first proves normalization for classical propositional logic, where he avoids the problem with $\vee$, by defining $\vee$ in terms of "negative" connectives. Then, when studying intuitionistic propositional logic, he also needs permutative reductions for proving normalization. By the way, our improper reductions turn out to be identical with the semi-proper reduction used in the SN proof for arithmetic by Leivant in [40].

4.2 The system AUT-Pi$_0$

4.2.1 For brevity and clarity we study a system of terms with the same "connectives" and reductions as AUT-Pi (so the essential problems with SN become clear) but with a simplified type-structure. It can be compared with the *normable expressions* of Ch. IV. Later (sec. 5.4) we extend our results to AUT-Pi.

4.2.2 Reduced type structure

The *reduced types* or *norms* (syntactical variables α, β, γ, ν) are inductively given by:

(1) τ is a norm

(2) if α and β are norms then also $\alpha \otimes \beta$, $\alpha \to \beta$ and $\alpha \oplus \beta$

Note: If we write $[\alpha]\beta$ instead of $\alpha \to \beta$ it is clear that the norms of Ch. IV form a subset of the present norm system. We write $\alpha \to \beta$ with the purpose to show that our norms form a simple type structure over a single fixed type, τ. This is also true of the norms in Ch. IV. Hence normability results (as in Ch. IV, or as given earlier by Jutting and Nederpelt [36,51] for certain Automath variants) can alternatively be proved as follows: the generalized systems under consideration are not essentially richer than simple, non-generalized type theory, in the sense that they do provide the same set of terms of free λ-calculus with a type as does a simple, non-generalized system. Compare Ben-Yelles [6].

4.2.3 Terms of AUT-Pi$_0$

All *terms* (syntactical variables A, B, C, $\cdots$) have a norm. The norm of A is denoted $\mu(A)$. We also write $A \in \alpha$ for $\mu(A) \equiv \alpha$. Terms are constructed according to:

(i) variables x, y, z, $\cdots$ of any norm

(ii) $x \in \alpha$, $A \in \alpha$, $B \in \beta \Rightarrow [x{:}A]B \in \alpha \to \beta$

(iii) $C \in \alpha \to \beta$, $A \in \alpha$, $B \in \beta \Rightarrow \langle C,A,B\rangle \in \alpha \otimes \beta$

(iv) $A \in \alpha$, $B \in \beta \Rightarrow i_1(A,B) \in \alpha \oplus \beta$, $i_2(A,B) \in \beta \oplus \alpha$

(v) $B \in \alpha \to \beta,\ A \in \alpha \Rightarrow \{A\}B \in \beta$

(vi) $B \in \alpha \otimes \beta \Rightarrow B_{(1)} \in \alpha,\ B_{(2)} \in \beta$

(vii') $[x{:}A]C \in \alpha \to \gamma,\ [y{:}B]D \in \beta \to \gamma \Rightarrow ([x{:}A]C \oplus [y{:}B]D) \in (\alpha \oplus \beta) \to \gamma$

These terms can be compared with the 3-expressions of AUT-Pi. However there are no constants, no instantiation (and no δ), it has simpler type structure and it has only $\oplus$-terms of the form $[x{:}A]C \oplus [y{:}B]D$. Below we also consider a variant AUT-Pi_1 which has general $\oplus$-terms. Instead of rule (vii') it has rule

(vii) $B \in \alpha \to \gamma,\ C \in \beta \to \gamma \Rightarrow B \oplus C \in (\alpha \oplus \beta) \to \gamma$

Below, we often omit type-labels in $[x{:}A]B$, $i_1(A,B)$, $i_2(A,B)$ and $\langle C,A,B\rangle$, just writing $[x]B$, $i_1(A)$, $i_2(A)$ and $\langle A,B\rangle$.

4.3 The reduction rules

4.3.1 We consider four groups of reduction rules

I The introduction-elimination rules (IE-reductions) β, π and +' (see 2.6).

Rule +' is particularly appropriate for AUT-Pi_0, i.e. in connection with rule (vii'). For AUT-Pi_1 we rather use rule +.

II The ext-reductions η, σ and ε

Here we use the simple unrestricted version of σ: $\langle C,A_{(1)},A_{(2)}\rangle > A$.

III Permutative reductions (p-reductions)

($\to$) $\{A\}\{B\}([x]C \oplus [y]D) > \{B\}([x]\{A\}C \oplus [y]\{A\}D)$

($\otimes$) $(\{A\}([x]C \oplus [y]D))_{(1)} > \{A\}([x]C_{(1)} \oplus [y]D_{(1)})$ - similarly for (2)-projection

($\oplus$) $D \equiv E \oplus F \Rightarrow \{\{A\}([x]B \oplus [x]C)\}D > \{A\}([x]\{B\}D \oplus [x]\{C\}D)$

The general pattern of these rules looks like

$$\mathcal{O}(\{A\}([x]B \oplus [y]C)) > \{A\}([x]\mathcal{O}(B) \oplus [y]\mathcal{O}(C))$$

where $\mathcal{O}$ is an operation on expressions, given in one of the following

ways: $\mathcal{O}(B) \equiv \{A\}B$, $\mathcal{O}(B) \equiv \{B\}(E \oplus F)$, $\mathcal{O}(B) \equiv B_{(1)}$ or $\mathcal{O}(B) \equiv B_{(2)}$. The norms of these B's are respectively $\alpha \to \beta$, $\alpha \oplus \beta$ and $\alpha \otimes \beta$. That is why the rules are coded $(\to)$, $(\oplus)$ and $(\otimes)$.

In case the argument of $\mathcal{O}$ allows outside (i.e. $\oplus$-reduction), the p-step does not produce a new equality: $\mathcal{O}(\{i_1(A)\}[x]B \oplus [y]C) > \mathcal{O}(B[\![A]\!]) \equiv \mathcal{O}(B)[\![A]\!] < \{i_1(A)\}([x]\mathcal{O}(B) \oplus [y]\mathcal{O}(C))$. Below (6.2), it turns out that, generally, p-equality is generated by $\beta\eta+\varepsilon$-reduction.

The above mentioned rules are the standard ones from proof theory. There it is formulated like this: if the conclusion of an $\vee$-elimination rule forms the major premise of an elimination rule, then the latter rule can be pushed upward through the $\vee$-elimination rule. E.g. our $\to$-rule can be compared with the following proof theoretic reduction:

$$\dfrac{\dfrac{\begin{matrix}A\\ \gamma\end{matrix}\quad \dfrac{\begin{matrix}\alpha \vee \beta\\ B\end{matrix}\quad \begin{matrix}[\alpha]\\ \gamma \to \delta\\ C\end{matrix}\quad \begin{matrix}[\beta]\\ \gamma \to \delta\\ D\end{matrix}}{\gamma \to \delta}\vee E}{\delta}\to E}{} \quad > \quad \dfrac{\begin{matrix}B\\ \alpha \vee \beta\end{matrix}\quad \dfrac{\begin{matrix}[\alpha]\\ \gamma\end{matrix}\quad \begin{matrix}\gamma \to \delta\\ A\end{matrix}\quad \begin{matrix}[\alpha]\\ C\end{matrix}}{\delta}\quad \dfrac{\begin{matrix}[\beta]\\ \gamma\end{matrix}\quad \begin{matrix}\gamma \to \delta\\ A\end{matrix}\quad \begin{matrix}[\beta]\\ D\end{matrix}}{\delta}}{\delta}$$

Both here and in proof theory the p-reductions are primarily introduced for technical reasons. However, as Pottinger [56] points out there is some intuitive justification for them too. Part of it, that in some cases they do not extend the equality relation is stated above.

It has been suggested to allow other permutative reductions as well (Pottinger [56], Leivant [40]). However, Zucker [76] has shown that this spoils SN.

IV Improper reductions (im-reductions)

(im) $\{A\}([x{:}B]C \oplus [y{:}D]E) > C$,

$\{A\}([x{:}B]C \oplus [y{:}D]E) > E$

Notice that the set of free variables of the expression can be enlarged by performing an im-reduction. If an inside im-reduction takes place inside the scope of some bound variable, the latter variables have to be renamed in order to avoid any confusion.

These reductions can be compared with Leivant's [40] *semi-proper*

reductions. They degenerate to what Prawitz calls *immediate simplifications*, when $x \notin \mathrm{FV}(C)$, resp. $y \notin \mathrm{FV}(E)$.

4.3.2 One step and many-step reduction

One-step reduction $>_1$ is, as well, generated from the *main* or *outside* reductions given above, by the monotonicity rules. Then follows many-step reduction $\geq$ from *reflexivity* and *transitivity*.

4.3.3 The usual substitution properties are valid, e.g.,

$$B >_1 B' \Rightarrow B[\![A]\!] >_1 B'[\![A]\!] \quad \text{and}$$

$$A >_1 A' \Rightarrow B[\![A]\!] \geq B[\![A']\!] \quad \text{etc.}$$

4.4 Closure for AUT-Pi_0

4.4.1 First notice that AUT-Pi_0 is certainly not closed under η, because of the restrictive rule (vii'). So the proof below is intended *for the η-less case*.

4.4.2 Due to the simple type structure it is quite easy to show that norms are preserved under substitution and reduction and hence that AUT-Pi_0 is closed under reduction.

4.4.3 Substitution lemma for the norms: $x \in \alpha$, $A \in \alpha$, $B \in \beta \Rightarrow B[\![x/A]\!] \in \beta$ (and $B[\![x/A]\!]$ a term).

Proof: Ind. on length of B. □

4.4.4 Reduction lemma for norms: $A \in \alpha$, $A > A' \Rightarrow A' \in \alpha$ (this includes CL_1).

Proof: Ind. on the definition of $>$. For β and $+'$ use the substitution lemma. E.g. $+'$: let $A \equiv \{i_1(A_1)\}([x]A_2 \oplus [y]A_3)$, $A \in \alpha$, $A' \equiv A_2[\![A_1]\!]$. Then, for some α_1, α_2, $A_1 \in \alpha_1$, $([x]A_2 \oplus [y]A_3) \in (\alpha_1 \oplus \alpha_2) \to \alpha$, so $[x]A_2 \in \alpha_1 \to \alpha$, $x \in \alpha_1$, $A_2 \in \alpha$. So $A_2[\![A_1]\!] \in \alpha$, q.e.d. Or a permutative reduction: $A \equiv (\{A_1\}([x]A_2 \oplus [y]A_3))_{(1)}$,

$A \in \alpha$, $A' \equiv \{A_1\}([x]A_{2(1)} \oplus [y]A_{3(1)})$. Then for some β, α_1, α_2, $\{A_1\}([x]A_2 \oplus [y]A_3) \in \alpha \otimes \beta$, $x \in \alpha_1$, $y \in \alpha_2$, $A_1 \in \alpha_1 \otimes \alpha_2$, $A_2 \in \alpha \otimes \beta$, $A_3 \in \alpha \otimes \beta$. So $A' \in \alpha$. □

4.4.5 Theorem: (closure) $A \in \alpha$, $A \geq A'$ (without η) $\Rightarrow A' \in \alpha$

Proof: Ind. on $\geq$.

4.5 The system AUT-Pi$_1$

4.5.1 Instead of rule (vii') it has the rule

$$B \in \alpha \to \gamma,\ C \in \beta \to \gamma \Rightarrow B \oplus C \in (\alpha \oplus \beta) \to \gamma$$

and it has + instead of +'.

Of course (vii') $\Rightarrow$ (vii), so indeed AUT-Pi$_1$ contains AUT-Pi$_0$. We can define a translation ϕ from AUT-Pi$_0$ to AUT-Pi$_1$ such that $\phi(A) \geq_\eta A$ and which shows that AUT-Pi$_1$ is not a very essential extension of AUT-Pi$_0$.

The translation is given by ind. on length. The only nontrivial clause is $\phi(C_1 \oplus C_2) \equiv [x{:}M_\alpha]\{x\}\phi(C_1) \oplus [x{:}M_\beta]\{x\}\phi(C_2)$, where $C_1 \oplus C_2 \in (\alpha \oplus \beta) \to \gamma$ and M_α, M_β are suitable fixed expressions of norms α, β and x, y are chosen of norm α, β such that $x \notin \mathrm{FV}(C_1)$, $y \notin \mathrm{FV}(C_2)$, respectively, $\wedge$. On variables, ϕ acts like identity. For the rest, ϕ just commutes with the formation rules. Clearly, ϕ leaves the norm invariant and is indeed a translation into AUT-Pi$_0$.

4.5.2 We have the following properties

(1) $\phi(B[\![x/A]\!]) \equiv \phi(B)([\![x/\phi(A)]\!])$, if $\mu(x) \equiv \mu(A)$

(2) For IE-reduction: $A >_1 B \Rightarrow \phi(A) >_1 \phi(B)$

(3) For (IE-ext)-reduction: $A >_1 B \Rightarrow \phi(A)$ properly reduces to $\phi(B)$.

Proofs: By induction on length. The β-case of (2) uses (1):

$\phi(\{A_1\}[y]A_2) \equiv \{\phi(A_1)\}[y]\phi(A_2) >_1 \phi(A_2)[\![\phi(A_1)]\!] \equiv \phi(A_2[\![A_1]\!])$, q.e.d.

The +-case of (2): $\phi(\{i_1(A_1)\}(A_2 \oplus A_3)) \equiv$

$\{i_1(\phi(A_1))\}([x{:}M_\alpha]\{x\}\phi(A_2) \oplus [x{:}M_\beta]\{x\}\phi(A_3)) >_{1,+'} \{\phi(A_1)\}\phi(A_2) \equiv \phi(\{A_1\}A_2)$. The ε-case of (3): $\phi([x{:}A]\{i_1(x)\}B \oplus [x{:}C]\{i_2(x)\}B) \equiv [x{:}M_\alpha]\{x\}[x{:}\phi(A)]\{i_1(x)\}\phi(B) \oplus [x{:}M_\beta]\{x\}[x{:}\phi(C)]\{i_2(x)\}\phi(B) \geq_\beta [x{:}M_\alpha]\{i_1(x)\}\phi(B) \oplus ([x{:}M_\beta]\{i_2(x)\}\phi(B) >_\varepsilon \phi(B)$. We particularly investigate the case of η which is not allowed in AUT-Pi_0: $\phi([x{:}A]\{x\}B \oplus C) \equiv ([x{:}M_\alpha]\{x\}[x{:}\phi(A)]\{x\}\phi(B) \oplus [y{:}M_\beta]\{y\}\phi(C) >_\beta [x{:}M_\alpha]\{x\}\phi(B) \oplus [y{:}M_\beta]\{y\}\phi(C) \equiv \phi(B \oplus C)$ - if $x \notin FV(B)$ -.

4.5.3 In the sequel we prove SN for some versions (i.e. with and without p-red. etc.) of AUT-Pi_0. By the above properties we can easily extend the p- and im-less case to AUT-Pi_1:
AUT-Pi_0 SN (with +') $\Rightarrow$ AUT-Pi_1 SN (with +).

Proof: Let A be an AUT-Pi_1 term. Use ind. on $\theta(\phi(A))$. □

But, from SN with + follows SN with + and +', because each +'-step can be simulated by a + a β-step, so θ_+ decreases under +'-reduction. And, because AUT-Pi_1 contains AUT-Pi_0 we also get SN for AUT-Pi_0 with + and +'.

4.5.4 The postponement requirements

For AUT-Pi_0- and AUT-Pi_1-expressions it is quite straightforward to show the requirements (1), (2) of 3.13. E.g. let $\langle A_{(1)}, A_{(2)}\rangle A$. Then $A \in \alpha \otimes \beta$. So A is not an inj-term, a $\oplus$-term, or an abstr-term. Etc.

4.6 The first-order character of the systems

4.6.1 In IV.1.5 we emphasized the importance of the property

$$\mu(\{A_1\}B) \equiv \mu(\{A_2\}B), \text{ in particular } \mu(\{A_1\}[x]A_2) \equiv \mu(A_2)$$

i.e. the functional complexity of $\{A\}B$ does not depend on the argument A. Alternatively stated: it is of course possible that the different values of B have different types, but apparently there is a strong uniformity in these types, for the functional complexity of all the values

is the same. In fact, we defined a system to be *first-order* if this property was present.

4.6.2 Generally, the introduction of ⊕-types and ⊕-terms might spoil this uniformity: we might be able to define functions completely different on both parts of their domain. So, by "general" ⊕-functions the first-order property above gets lost. However, in AUT-Pi_0, AUT-Pi_1 and in AUT-Pi the domain of ⊕-functions is explicitly restricted in such a way, that the first-order property can be maintained, viz. by requiring

(1) in AUT-Pi_0 that $\mu(B) \equiv \mu(C)$ when forming $(x)B \oplus (y)C$

(2) in AUT-Pi_1 that $B \in \alpha \to \gamma$, $C \in \beta \to \gamma$ when forming $B \oplus C$

(3) in AUT-Pi that $B \in \alpha \to \gamma$, $C \in \beta \to \gamma$ when forming $B \oplus C$

As a consequence we still have $\mu(\{A_1\}B) \equiv \mu(\{A_2\}B)$ and in particular $\mu(\{A\}([x]B \oplus [y]C)) \equiv \mu(B) \equiv \mu(C)$.

4.6.3 Now it will be clear that the generalized ⊕-rules of 2.7 would spoil the first-order character. Example: let $A \mathsf{E} \tau$, $B \mathsf{E} \tau$, $C \mathsf{E} \tau$, $D \mathsf{E} \tau$ then $[x{:}A]C \mathsf{E} A \to \tau$, $[x{:}B]D \mathsf{E} B \to \tau$. So $[x{:}A]C \oplus [x{:}B]D \mathsf{E} (A \oplus B) \to \tau$. So, if $E \mathsf{E} A \to C$, $F \mathsf{E} B \to D$ then $(E \oplus F) \mathsf{E} \Pi([x{:}A]C \oplus [x{:}B]D)$. Clearly the functional complexity of $\{i_1(G)\}(E \oplus F)$ for $G \mathsf{E} A$ and $\{i_2(H)\}(E \oplus F)$ for $H \mathsf{E} B$ can be completely different, viz. that of C and D respectively.

4.6.4 It is possible that a notion of norm (i.e. simplified type) can be defined which is manageable and measures functional complexity of these general ⊕-terms, but the present norm (and the corresponding SN proof) is certainly not suitable for this situation.

4.6.5 Remark: Strictly speaking, the suggested correction between the typing relation in AUT-Pi and the norms in AUT-Pi_0 has not yet been accounted for. The preceding statements have to be understood on an intuitive, heuristic level.

4.7 A proof of βπησ-SN

4.7.1 Here we show that the first β-SN proof of Ch.IV straightforwardly carries over to the case of βπησ-SN. As our domain of expressions we take, e.g., the terms of AUT-Pi_1.

4.7.2 SN-conditions for βπ

For non-main-reducing expressions (also called *immune forms* or IF's) it is sufficient for SN if all their proper subexpressions are SN. Incidentally this is also true for projection expressions (because main π-reduction amounts to picking a certain subexpression). So we have: A SN $\Leftrightarrow A_{(1)}$ SN, and the funny property: $A_{(1)}$ SN $\Leftrightarrow A_{(2)}$ SN.

We recall the SN condition for appl expressions in this case:

$$\{A\}B \text{ SN} \Leftrightarrow A \text{ SN},\ B \text{ SN} \quad \text{and} \quad (B \geq [x]C \Rightarrow C[\![A]\!] \text{ SN})$$

4.7.3 Heuristics: the dead end set for β

So, the substitution theorem for SN is again sufficient for proving SN (see IV.2.4). The crucial case of the substitution theorem for β-SN was where A is SN, $B \equiv \{B_1\}B_2$ is SN, $B_2[\![A]\!] \geq [y]C$, but $B_2 \not\geq [y]C_0$. I.e. the reduction to square brackets form depends essentially on the substitutions. Then we used the square brackets lemma: $B_2 \geq \{\bar{F}\}x$, $(\{\bar{F}\}x)[\![A]\!] \geq [y]C$.

We define the set E_x of these expression $\{\bar{F}\}x$ symbolically by a recursion equation $E_x = x + \{U\}E_x$, where U stands for the set of all expressions and it is of course understood that all expressions in E_x are in AUT-Pi_1 again.

The expressions $\{\bar{F}\}x$ can be considered as *dead ends* when one tries to copy in B_2 the contractions leading from $B_2[\![A]\!]$ to $[y]C$, i.e. when one tries to come "as close as possible" to an abstr expression. We do not bother to make the concept of dead end more precise, or more general, but just give this informal explanation for naming E_x the *dead end set* w.r.t. x, β-reduction, and abstr expressions.

4.7.4 The dead end set for $\beta\pi$

When one tries to copy a $\beta\pi$-reduction sequence of $B[\![A]\!]$ in B one need not end up with an expression in E_x, but, e.g., can also end in $x_{(1)}$. The following theorem states that F defined by

$$F = x + F_{(1)} + F_{(2)} + \{U\}F$$

is the dead end set w.r.t. x, $\beta\pi$ and immume forms (IF's). Let $\geq$ stand for $\geq_{\beta\pi}$, and let * stand for $[\![x/A]\!]$.

Theorem: If B SN, $B^* \geq C$, $C \in$ IF then $B \geq C_0$, $C_0^* \geq C$ with either (i) C_0^* non-main reduces to C, or (ii) $C_0 \in F$.

Proof: Just like the square brackets lemma (second proof, IV.2.4.3), by ind. on (1) $\theta(B)$, (2) $\ell(B)$. Let B^* main-reduce to C (otherwise take $B \equiv C_0$). Then $B \equiv x$, (and take $C_0 \equiv B$, $C_0 \in F$), $B \equiv D_{(1)}$, $B \equiv D_{(2)}$ or $B \equiv \{D_1\}D_2$. E.g. let $B \equiv D_{(1)}$. Then $D^* \geq \langle D_1, D_2\rangle$, $D_1 \geq C$. Apply ind. hyp. (2) to D. In case (i), $D \geq \langle E_1, E_2\rangle$, $E_1^* \geq D_1$, $E_2^* \geq D_2$, so $B \geq E_1$, $E_1^* \geq C$. Then apply ind. hyp. (1) to E_1. In case (ii), $D \geq E_0$, $E_0 \in F$, $E_0^* \geq \langle D_1, D_2\rangle$ and $B \geq E_{0(1)} \in F$, $E_{0(1)}{}^* \equiv E_{0(1)}^* \geq C$, so case (ii) holds for B too. □

Remark: (1) Similarly we can prove a more general outer-shape lemma (see II.11.5.4) for $\beta\pi$, where the condition "$C \in$ IF" simply has been dropped.

(2) It is probable that such "standardization-like" theorems can also be proved without using SN (as in II.11).

4.7.5 Heuristics: the norms of dead ends

The point of the β-SN proof is:

$$B \in E_x \Rightarrow \ell(\mu(B)) \leq \ell(\mu(x))$$

- where ℓ is the length of the norm -. So, if $B[\![A]\!] \geq [y]C$ then $\ell(\mu(y)) < \ell(\mu(x))$, and we can use ind. on norms in the crucial case of the substitution theorem.

We are lucky that the same method works for $\beta\pi$-reduction too. Namely

$$B \in F \Rightarrow \ell(\mu(B)) \leq \ell(\mu(x))$$

So, if

$$B[\![A]\!] \geq_{\beta\pi} [y]C \quad \text{then} \quad \ell(\mu(y)) \leq \ell(\mu(x)).$$

4.7.6 The substitution theorem for $\beta\pi$-SN

Theorem: A $\beta\pi$-SN, B $\beta\pi$-SN $\Rightarrow$ $B[\![x/A]\!]$ $\beta\pi$-SN

Proof: Ind. on (1) $\mu(A)$, (2) $\theta_{\beta\pi}(B)$, (3) $\ell(B)$. Let $\geq$ be $\geq_{\beta\pi}$. If $B \equiv x$ then $B[\![A]\!] \equiv A$ so SN. If $B \in$ IF or $B \equiv C_{(1)}$ or $B \equiv C_{(2)}$ use ind. hyp. (3). If $B \equiv \{B_1\}B_2$ proceed as for β-SN, using the norm properties of the dead end set F. □

4.7.7 $\beta\pi$-SN and $\beta\pi\eta\sigma$-SN

An immediate corollary of the substitution theorem for $\beta\pi$-SN is $\beta\pi$-SN itself. Now we can extend this to $\beta\pi\eta\sigma$-SN (as in II.7.2.5) using $(\beta\pi)$-$(\eta\sigma)$-pp, a case of ext-pp (see II.9.2). The requirement for pp is indeed fulfilled (see 4.5.4).

VIII.5 Three proofs of $\beta\pi$+-SN, with application to AUT-Pi

5.1 A proof of $\beta\pi$+-SN using p- and im-reductions

5.1.1 Here we show how the preceding SN-proof (based on the first version of the simple β-SN proof in Ch. IV) has to be modified in order to cope with + (or +'). First we shall see how the norm considerations of that proof do not go through.

5.1.2 The dead end set for $\beta\pi$+

Let $\geq$ be $\geq_{\beta\pi+}$. The following theorem states that the set G defined by

$$G = x + G_{(1)} + G_{(2)} + \{U\}G + \{G\}(U \oplus U)$$

is the dead end set w.r.t. x, $\beta\pi$+ and IF's. Let * stand for $[\![x/A]\!]$.

Theorem: Let B be SN, $B^* \geq C$, $C \in$ IF then $B \geq C_0$ with either (1) C_0^* non-main reduces to C, or (2) $C_0^* \geq C$, $C_0 \in$ G

Proof: As in 4.7.4, by ind. ond (i) $\theta(B)$, (ii) $\ell(B)$ □

Similarly, we can prove the corresponding outer shape lemma.

The problem is now that the norm of the expressions in G is not related to the norm of x. E.g. consider the typical +-dead end $\{x\}(B \oplus C)$.

5.1.3 Improving the dead end set by p-reduction

We restrict our domain of consideration to AUT-Pi_0. Instead of rule + we choose rule +'. Besides we add permutative reductions. Then a great deal of the "bad guys" among the dead ends, i.e. whose norm is not related to that of x, can be main reduced by a p-reduction. This will (in the next section) result in an *improved dead end set* H defined by

H = F + {F}(U ⊕ U) with F as in 4.7.4.

5.1.4 Let ≥ be β+'πp-reduction. The direct reducts of a p-main step are of the form $\{A\}([x]\mathcal{O}(B) \oplus [y]\mathcal{O}(C))$ (see 4.3.1 for the definition of $\mathcal{O}$), so never are in one of the immume forms (abstr, inj, pair, plus).

Lemma: p-main reduction steps in a reduction to IF can be circumvented

Proof: The last p-main step in a reduction to IF must be followed by a +'-main step. However this combination can be replaced by a single internal +'-step. □

Corollaries:

(1) $\{B\}([x]C_1 \oplus [x]C_2) \geq D$, $D \in$ IF $\Rightarrow B \geq i_j(A)$, $C_j[\![A]\!] \geq D$ (j+1 or 2)

(2) $\{B\}C \geq D$, $D \in$ IF ⇒ Either (i) $C \geq [y]E$, $E[\![D]\!] \geq D$ or
(ii) $B \geq i_j(A)$, $C \geq ([x]C_1 \oplus [x]C_2)$, $C_j[\![A]\!] \geq D$, j=1 or j=2.

(3) $B_{(j)} \geq D$, $D \in$ IF ⇒ $B \geq C_1, C_2$, $C_j \geq D$ (j=1,2).

Proof: Each of these reductions to IF can be replaced by one without p-main steps. □

Part of the two corollaries can be summarized (with $\mathcal{O}$ as in 4.3.1) by:

if $\mathcal{O}(B) \geq D$, $D \in$ IF then $B \geq C$, $C \in$ IF, $\mathcal{O}(C) \geq D$.

This gives another lemma.

Lemma: If $\mathcal{O}(\{B\}([x]C_1 \oplus [x]C_2)) \geq D$, $D \in$ IF then
$\{B\}([x]\mathcal{O}(C_1) \oplus [x]\mathcal{O}(C_2)) \geq D$.

Proof: $\{B\}([x]C_1 \oplus [x]C_2) \geq E$, $E \in$ IF, $\mathcal{O}(E) \geq D$. So $B \geq i_j(A)$, $C_j[\![A]\!] \geq E$. But then $\{B\}([x]\mathcal{O}(C_1) \oplus [x]\mathcal{O}(C_2)) \geq \mathcal{O}(C_j[\![A]\!]) \geq \mathcal{O}(E) \geq D$, q.e.d. □

This proof amounts to: if an expression allows both p-main and IE-main reduction then we can insert p-main followed by +'-main before performing the IE-main step. Now we prove the theorem about the improved dead end set H. Let * stand for $[\![x/A]\!]$.

Theorem: If B SN, $B^* \geq C$, $C \in$ IF then $B \geq C_0$, $C_0^* \geq C$ with either (1) C_0^* non-main reduces to C, or (2) $C_0 \in$ H

Proof: As in 4.7.4, by ind. on (i) $\theta(B)$, (ii) $\ell(B)$. Here θ refers to the current reduction $\beta\pi$+'p. Let B^* main reduce to C, $B \not\equiv x$. If the first main step can be mimicked in B use ind. hyp. (i). Otherwise, by ind. hyp. (ii) $B \geq \mathcal{O}(D)$, $D \in$ H, $\mathcal{O}(D)^* \geq C$. If $D \in$ F then $\mathcal{O}(D) \in$ H and we are done. Otherwise $D \equiv \{D_3\}([y]D_1 \oplus [y]D_2)$, $D_3 \in$ F. Then B properly reduces to $E \equiv \{D_3\}([y]\mathcal{O}(D_1) \oplus [y]\mathcal{O}(D_2))$, $E \in$ H, and by the previous lemma $E^* \geq C$, q.e.d. □

5.1.5 Improving the SN-conditions by im-reduction

The crucial SN-conditions for $\beta\pi$ +' (in AUT-Pi$_0$) is

If (1) A SN, B SN, (2) $B \geq [x]C \Rightarrow C[\![A]\!]$ SN and for j=1,2
(3) $B \geq [x]C_1 \oplus [x]C_2$, $A \geq i_j(D) \Rightarrow C_j[\![D]\!]$ SN, then is $\langle A\rangle B$ SN.

Now the p-reductions have improved our dead end set, but the problem is that they make the SN-conditions quite complicated. E.g. in order to prove that $\{A\}\{B\}([x]C_1 \oplus [x]C_2)$ is SN we need that $\{A\}C_1$ is SN, in particular if $C_1 \geq [y]E$ we need that $E[\![A]\!]$ is SN etc. I.e. the SN-condition of $\{A\}B$ ceases to be easily expressible in terms of direct subexpressions of reducts of A and B.

In order to solve this problem we add im-reduction. But at first we show that the dead end set is not changed by this addition.

5.1.6 The dead end set of $\beta\pi+'p,im$

Luckily the dead end set remains H. Let $\geq$ stand for $\geq_{\beta\pi+'p,im}$. The first lemma of 5.1.4 can be maintained. For let a p-main step be follwed by an im-main step. Then we can skip the main p-step and just apply the im-step internally.

The next corollaries need an obvious modification, in particular:

If $\{B\}([x]C_1 \oplus [x]C_2) \geq D$, D IF then either (1) $B \geq i_j(A)$, $C_j[\![A]\!] \geq D$ (for j=1 or j=2), or (2) $C_j \geq D$ (for j=1 or j=2).

And the property thereafter becomes:

If $\partial(B) \geq D$, $D \in$ IF then either (1) $B \geq C$, $C \in$ IF, $\partial(C) \geq D$, or (2) $\partial(B) \equiv \{B\}([x]C_1 \oplus [x]C_2)$, $C_j \geq D$ (for j=1 or 2)

But the second lemma of 5.1.4 remains unchanged. Namely, if an expression allows p-main reduction but also im-main reduction, then we can insert p-main followed by im-main before performing the im-main step.

E.g. $\{\{B_1\}([x]C_1 \oplus [x]C_2)\}([y]D_1 \oplus [y]D_2) >_p$

$\{B_1\}([x]\{C_1\}([x]D_1 \oplus [y]D_2) \oplus \cdots) >_{im} \{C_1\}([y]D_1 \oplus [y]D_2) >_{im} D_1$.

So, the theorem of 5.1.4, that the dead end set is still H, carries over too.

5.1.7 The new SN-conditions

The point of the im-reduction is that the SN-conditions for $\beta\pi+'p,im$ are identical with those for $\beta\pi+'$ (see 5.1.5). First we give the SN-conditions of $\{B\}([x]C_1 \oplus [x]C_2)$. These are (1) B SN, C_1 SN and C_2 SN, and (2) $B \geq i_j(A) \Rightarrow C_j[\![A]\!]$ SN (for j=1 and 2).

Proof: Let the above condition be fulfilled. Use ind. on (1) $\theta(B)$, (2) $\ell(B)$, The interesting case is when the first main step in a reduction is a p-step. So let $B \geq \{B_3\}([y]B_1 \oplus [y]B_2)$, to prove that $\{B_3\}([y]\{B_1\}C \oplus [y]\{B_2\}C)$ is SN, with $C \equiv [x]C_1 \oplus [x]C_2$. By ind. hyp. (1) or (2) we just need that B_3 is SN (trivial) that $\{B_j\}C$ SN for j=1,2 and that $\{B_j[\![D]\!]\}C$ is SN, where $B_3 \geq i_j(D)$. Since B properly reduces to both B_j and $B_j[\![D]\!]$ (in case $B_3 \geq i_j(D)$) we can use ind. hyp. (1) and get what we want. □

Theorem: The SN-conditions for $\beta\pi+'p,im$ are identical with those of $\beta\pi+'$ (see 5.1.5).

Proof: Let $\{A\}B$ fulfill the SN-conditions (1), (2), (3) of 5.1.5. We use ind. on $\theta(B)$. The interesting case is when the first main step is p. The case that $B \geq [x]B_1 \oplus [x]B_2$ has been done before, so let $B \geq \{B_3\}([x]B_1 \oplus [x]B_2)$, to prove that $\{B_3\}([x]\{A\}B_1 \oplus [x]\{A\}B_2)$ is SN. I.e. that B_3 SN, that $\{A\}B_1$ and $\{A\}B_2$ SN and that $\{A\}B_1[\![D]\!]$, $\{A\}B_2[\![D]\!]$ are SN whenever $B_3 \geq i_j(D)$ (j=1 or 2). Now B properly reduces to both B_j and $B_j[\![D]\!]$ (if $B_3 \geq i_j(D)$) so we use the ind. hyp. and get what we want. □

In other words: we just need that the direct subexpressions and the IE-main reducts (not *all* the main reducts) are SN for proving that an expression is SN.

5.1.8 The substitution theorem for SN

Notation: We just write $\mu(A) </\leq \mu(B)$ to abbreviate $\ell(\mu(A)) </\leq \ell(\mu(B))$

Theorem: B SN, A SN, $\mu(x) \equiv \mu(A) \Rightarrow B[\![x/A]\!]$SN

Proof: Ind. on (I) $\mu(A)$, (II) $\theta(B)$, (III) $\ell(B)$. The crucial case is when $B \equiv \{B_1\}B_2$ and $B[\![A]\!]$ IE-main reduces. If this first main step can be mimicked in B use the second ind. hyp. Otherwise we end up with $\{B_1'\}C$ or $\{C\}B_2'$ with $C \in$ H and $B_1 \geq B_1'$ or $B_2 \geq B_2' \equiv [y]D_1 \oplus [y]D_2$, respectively. If $C \in$ G then $\mu(B_1') < \mu(C) \leq \mu(x)$ so a first main reduction of $(\{B_1'\}C)[\![A]\!]$ involves a substitution $[\![z/E]\!]$ with $\mu(z) \leq \mu(B_1') < \mu(x)$. And a first main-IE reduction step of $(\{C\}B_2')[\![A]\!]$ must be a +'-step, so involves a substitution $[\![z/E]\!]$ with $C[\![A]\!] \geq i_j(E)$. So in that case too $\mu(z) \equiv \mu(E) < \mu(C) \leq \mu(x)$. Anyhow if $C \in$ G, we can use ind. hyp. (I). Otherwise $C \equiv \{C_3\}([y]C_1 \oplus [y]C_2)$, with $C_3 \in$ G. Then a p-step is possible and can be inserted before doing the main IE-step. This p-step can be mimicked in the reduction of B, so we can use ind. hyp. (II). □

5.1.9 SN for AUT-Pi_0 and AUT-Pi_1

Like before, an immediate corollary is $\beta\pi+'p,im$-SN for AUT-Pi_0, so $\beta\pi+'$-SN for AUT-Pi_0, whence $\beta\pi+$-SN for AUT-Pi_1. Then by pp we can extend the AUT-Pi_1 result to $\beta\pi+\eta\sigma$-SN. (Not for ε.)

5.1.10 An alternative method

Actually im-reduction can be avoided in this proof. Namely the effect of p-reductions on the SN-conditions can be expressed by means of certain inductively defined sets.

We define a set of expressions $B!$ by

$$B! = B + \{U\}([x](B!) \oplus U) + \{U\}(U \oplus [x](B!)).$$

I.e. $B!$ contains all those expressions that im-reduce to B.

Then the SN-conditions for $\beta\pi+'$ become

If (1) B SN, C SN, (2) $B \geq B' \in A!$, $C \geq C' \in ([y]D)! \Rightarrow D[\![A]\!]$ SN, and (3) $B \geq B' \in (i_j(A))!$, $C \geq ([y]C_1 \oplus [y]C_2)! \Rightarrow C_j[\![A]\!]$ SN $(j=1,2)$ then $\{B\}C$ SN.

5.2 A second proof of $\beta\pi+'$-SN, using im-reduction

5.2.1 This proof is based on the second instead of the first β-SN-proof of Ch. IV (sec. IV.2.5, see also VII.4.5). There we did not use the square brackets lemma, and no dead end set, so we can do without p-reduction. Our language is AUT-Pi$_0$, again, and $\geq$ stands for $\geq \beta\pi+',im$.

5.2.2 Replacement theorem for SN

As explained in VII.4.5, the kernel of this type of proof is a replacement theorem, rather then a substitution theorem, for SN.

Theorem: If B SN, A SN, $\mu(x) \equiv \mu(A)$ then $B[\![x/A]\!]_{LR}$ SN.

Proof: By ind. on (I) $\mu(A)$, (II) $\theta(B)$, (III) $\ell(B)$. We write * for $[\![x/A]\!]_{LR}$. Consider a reduction sequence $B^* >_1 \cdots >_1 F >_1 G$, where the contraction leading from F to G is the first contraction not taking place inside some reduct of one of the inserted ocurrences of A. Realize first that the number of those inside-A contractions is finite, because A is SN. Now we prove that G is SN. Distinguish two possibilities:

(a) The step $F >_1 G$ does not essentially depend on the inserted

A's and can be mimicked in B. I.e. $B >_1 G_0$, $G_0^* \geq G$. In this case we use ind. hyp. (II).

(b) Otherwise some reduct of some inserted A plays a crucial role in the redex contracted. If $F > G$ is a π-step, then, e.g., $B \equiv \cdots x \cdots x_{(1)} \cdots$, $B^* \equiv \cdots A \cdots A_{(1)} \cdots$, $F \equiv \cdots A' \cdots \langle C_1, C_2 \rangle_{(1)} \cdots$, $G \equiv \cdots A' \cdots C_1 \cdots$. Now form $B_0 \equiv \cdots x \cdots y \cdots$ from B by replacing $x_{(1)}$ by a fresh y, with $\mu(y) \equiv \alpha_1$ (where $\alpha \equiv \alpha_1 \times \alpha_2$). And $B \equiv B_0[\![y/x_{(1)}]\!]$ so B_0 is SN, $\theta(B_0) \leq \theta(B)$, $\ell(B_0) < \ell(B)$. So by ind. hyp. (II) or (III), B_0^* is SN and $B_0^* \geq G_0 \equiv \cdots A' \cdots y \cdots$ with $G \equiv G_0[\![y/C_1]\!]_{LR}$. Here G_0 is SN, C_1 is SN, $\mu(y) \equiv \mu(C_j)$, $\ell(\mu(y)) < \ell(\mu(x))$ so we can apply ind. hyp. (I) to get that G is SN. If $F > G$ is a β-step argue as in IV.2.5.3 or VII.4.5.6. If $F > G$ is a +'-step, the redex contracted is, e.g., $\{i_1(D)\}([y]C_1 \oplus [y]C_2)$, reducing to $C_1[\![D]\!]$. Now distinguish (b1) a reduct of an inserted A is crucial in $i_1(D)$, (b2) a reduct of an inserted A is crucial in $([y]C_1 \oplus [y]C_2)$. First case (b1). Then $B \equiv \cdots x \cdots \{x\}C_0 \cdots$, $C_0^* \geq [y]C_1 \oplus [y]C_2$, $A \geq i_1(D)$. By a norm argument the $\oplus$-term must be present in B already, so $C_0 \equiv [y]E_1 \oplus [y]E_2$, $E_1^* \geq C_1$, $E_2^* \geq C_2$. Now form $B_0 \equiv \cdots x \cdots E_1 \cdots$. This is an im-reduct of B, so SN and by ind. hyp. (II) B_0^* SN, reducing to $G_0 \equiv \cdots A' \cdots C_1 \cdots$, where $G \equiv \cdots A' \cdots C_1[\![D]\!] \cdots$. Clearly G_0 SN, D SN and $\ell(\mu(D)) < \ell(\mu(x))$. So $G \equiv G_0[\![y/D]\!]_{LR}$ SN by ind. hyp. (I). In case (b2), argue as in the β-case. Finally, the redex contracted in F is an im-redex, in which A plays a crucial role. I.e. $B \equiv \cdots x \cdots \{C_0\}x \cdots$, $A \geq [y]D_1 \oplus [y]D_2$, $C_0^* \geq C$, $F \equiv \cdots A' \cdots \{C\}([y]D_1 \oplus [y]D_2) \cdots$, $G \equiv \cdots A' \cdots D_1 \cdots$. Form $B_0 \equiv \cdots x \cdots y \cdots$, $B \equiv B_0[\![y/\{C_0\}x]\!]_{LR}$; so either by ind. hyp. (II) or (III) B_0^* is SN, reducing to $G_0 \equiv \cdots A' \cdots y \cdots$. Clearly D_1 SN, $\ell(\mu(D_1)) < \ell(\mu(x))$ so by ind. hyp. (I) $G \equiv G_0[\![y/D_1]\!]_{LR}$ is SN. □

5.2.3 An immediate corollary of this replacement theorem is the ordinary substitution theorem. From this, as before, follows $\beta\pi$+'im-SN for AUT-Pi$_0$. So we get $\beta\pi+\sigma\eta$-SN for AUT-Pi$_1$.

5.3 A proof of $\beta\pi+\eta\sigma$-SN by computability

5.3.1 In this proof we do not include $\eta\sigma$ by a pp-result afterwards, but consider these ext-reductions from the beginning of the proof on.

We must consider AUT-Pi_1 because AUT-Pi_0 is not closed under η. Our definition of *computability* has been strongly inspired by de Vrijer's definition in [70].

De Vrijer's definition is phrased in such a manner that the important properties: (1) computability implies SN, (2) computability is preserved under reduction, follow almost immediately. Then, as usual, we prove by ind. on length that expressions are *computable under substitution*.

Notice that we do not include ε.

5.3.2 The definition of computability

We write C_α for the set of computable terms of norm α. The set C_α is defined by induction on the length of α, as follows:

Let $B \in \alpha$. Then $B \in C_\alpha$ if B SN and the following requirements are fulfilled:

(1) $\alpha \equiv \alpha_1 \to \alpha_2$, $B \geq [y]C$, $A \in C_{\alpha_1} \Rightarrow C[\![A]\!] \in C_{\alpha_2}$

(2) $\alpha \equiv \alpha_1 \otimes \alpha_2$, $B \geq \langle C,D\rangle \Rightarrow C \in C_{\alpha_1}$, $D \in C_{\alpha_2}$

(3) $\alpha \equiv \alpha_1 \oplus \alpha_2$, $B \geq i_j(C) \Rightarrow C \in C_{\alpha_j}$ $(j=1,2)$

(4) $\alpha \equiv (\alpha_1 \oplus \alpha_2) \to \alpha_3$, $B \geq C \oplus D \Rightarrow C \in C_{\alpha_1 \to \alpha_3}$, $D \in C_{\alpha_2 \to \alpha_3}$.

Notice that each clause in the definition of C_α only depends on C_β's with β shorter than α.

5.3.3 We write C for the set of all computable expressions, the union of all the C_α's. By definition: $A \in C \Rightarrow A$ SN. Each condition in the definition of computability of B has the form: $B \geq C \Rightarrow P(C)$, with P some condition on C.

So computability is preserved under reduction.

5.3.4 Now we try to express the computability of an expression in terms of the computability of its subexpressions. First a lemma.

Lemma:

(1) $[x]C \geq [x]D \Rightarrow C \geq D$

(2) $\langle C,D\rangle \geq \langle E,F\rangle \Rightarrow C \geq E,\ D \geq F$

(3) $i_j(C) \geq i_j(D) \Rightarrow C \geq D \quad (j=1,2)$

(4) $C \oplus D \geq E \oplus F \Rightarrow C \geq E,\ D \geq F$

Proof: Without main reduction it is trivial. Otherwise it is η or σ. E.g. if $\langle C,D\rangle \geq \langle E,F\rangle$ then $C \geq \langle E,F\rangle_{(1)} \geq E$, $D \geq \langle E,F\rangle_{(2)} \geq F$ q.e.d. By the way, property (4) even holds in presence of ε. □

Lemma (computability conditions):

(0) variables are in C

(1) A SN, $C \in \mathsf{C}$, $D \in \mathsf{C} \Rightarrow \langle A,C,D\rangle \in \mathsf{C}$

(2) A SN, $C \in \mathsf{C} \Rightarrow i_1(C,A) \in \mathsf{C}$, $i_2(C,A) \in \mathsf{C}$

(3) $C \in \mathsf{C}$, $D \in \mathsf{C} \Rightarrow C \oplus D \in \mathsf{C}$

(4) $C \in \mathsf{C} \Rightarrow C_{(1)} \in \mathsf{C}$, $C_{(2)} \in \mathsf{C}$.

(5) $B \in \mathsf{C}$, $C \in \mathsf{C} \Rightarrow \{B\}C \in \mathsf{C}$

Proof: (0) is clear. (1), (2), (3) by the previous lemma. (4) as follows: Let $C \in \mathsf{C}$ then C SN so $C_{(j)}$ SN. If $C_{(j)} \geq [y]D$ then $C \geq \langle C_1,C_2\rangle$ with $C_j \geq [y]D$. Each of the C_j is in C, so $[y]D$ satisfies the required condition. Similar if $C_{(j)} \leq \langle D_1,D_2\rangle$, $C_{(j)} \geq i_1(D)$ etc. Proof of (5): Let $B,C \in \mathsf{C}$ so B,C SN. Induction on $\mu(B)$. We first check the SN conditions. Let $C \geq [y]D$ then $D[\![B]\!] \in \mathsf{C}$ so SN. Or let $B \geq i_j(D)$, $C \geq C_1 \oplus C_2$, to prove that $\{D\}C_j$ is SN. Well, both C_j's are in C, $D \in \mathsf{C}$ and we can use the ind. hyp. to prove that $\{D\}C_j \in \mathsf{C}$ (so SN). Further, if $\{B\}C \geq [y]E$ (or reduces to $\langle E,F\rangle$ etc.), this is only possible after a main step, so either via some $D[\![B]\!]$ with $C \geq [y]D$ or some $\{D\}C_j$ where $B \geq i_j(D)$, $C \geq C_1 \oplus C_2$. Those expressions were in C so $[y]E$ (and $\langle E,F\rangle$ etc.) satisfy the required conditions. □

5.3.5 Computability under substitution

For expressions $[y]C$ such simple computability conditions cannot be given. We define an even stronger notion than computability.

Definition: B is said to be *computable under substitution (cus)* if $A_1,\cdots,A_n \in \mathsf{C}$, $\mu(x_i) \equiv \mu(A_i)$ for $i=1,\cdots,n \Rightarrow B[\![\bar{x}/\bar{A}]\!] \in \mathsf{C}$

Some easy properties are:

(1) B Cus $\Rightarrow B \in \mathsf{C}$ (e.g. take n=0)

and (2) B Cus, $B \geq C \Rightarrow C \in \mathsf{C}$

Then a lemma: Let $\mu(C) \equiv \alpha_1 \to \alpha_2$ and let $F \in \mathsf{C}_{\alpha_1} \Rightarrow \{F\}C \in \mathsf{C}_{\alpha_2}$. Then

$$C \in \mathsf{C}_{\alpha_1 \to \alpha_2}$$

Proof: Clearly C is SN. We use ind. on $\ell(\alpha_1)$. If $C \geq [y]D$, $F \in \mathsf{C}_{\alpha_1}$ we must prove $D[\![F]\!] \in \mathsf{C}_{\alpha_2}$. This holds because $\{F\}C \geq D[\![F]\!]$. If $C \geq D \oplus E$ we must prove that $D,E \in \mathsf{C}$. For $i_1(F) \in \mathsf{C}_{\alpha_1}$, $\{i_1(F)\}C \in \mathsf{C}$ so $\{F\}D \in \mathsf{C}$. Now use the ind. hyp. Similar for E. □

5.3.6 Lemma: B Cus, C Cus $\Rightarrow [y{:}B]C$ Cus

Proof: Let C Cus, B Cus, $\bar{A} \in \mathsf{C}$ of the right norms. Abreviate $[\![\bar{x}/\bar{A}]\!]$ by *. We must prove that $[y{:}B^*]C^* \in \mathsf{C}$. Well, $B^* \in \mathsf{C}$, $C^* \in \mathsf{C}$ so $[y{:}B^*]C^* \in \mathsf{SN}$. If $[y{:}B^*]C^* \geq [y{:}D]E$, $F \in \mathsf{C}$ of the right norm then we need that $E[\![F]\!] \in \mathsf{C}$. Because C is Cus, $C[\![\bar{x},y/\bar{A},F]\!] \in \mathsf{C}$, which expression reduces to $E[\![F]\!]$, q.e.d. In particular, if $C^* \geq \{y\}(E_1 \oplus E_2)$, $y \notin \mathrm{FV}(E_1 \oplus E_2)$, we have that $\{F\}(E_1 \oplus E_2) \in \mathsf{C}$, so by the lemma $E_1 \oplus E_2 \in \mathsf{C}$, $E_1 \in \mathsf{C}$, $E_2 \in \mathsf{C}$, q.e.d. □

Theorem: All AUT-Pi$_1$ expressions are Cus

Proof: Variables are Cus by definition. Further use induction on length. For the abstr case use the previous lemma. For all the other cases use the lemma in 5.3.4. E.g. to prove that $\{B\}C$ is Cus. Let * be as in the previous lemma. By ind. hyp. $B^* \in \mathsf{C}$, $C^* \in \mathsf{C}$, so $\{B^*\}C^* \in \mathsf{C}$.

Corollaries: (1) All AUT-Pi$_1$ expressions are computable

(2) All AUT-Pi$_1$ expressions are $\beta\pi{+}\eta\sigma$-SN

5.4 Strong normalization for AUT-Pi

5.4.1 The normability of AUT-Pi

In order to extend our results from AUT-Pi to AUT-Pi we must first extend our definition of norm (see 4.2.3), and implicitly, of normability, as follows:

$$\mu(\tau) \equiv \tau$$

$$\mu(A) \equiv \alpha \to \beta \Rightarrow \mu(\Pi(A)) \equiv \alpha \to \beta$$

$$\mu(A) \equiv \alpha \to \beta \Rightarrow \mu(\Sigma(A)) \equiv \alpha \otimes \beta$$

$$A, B \text{ of degree } 2 \Rightarrow \mu(A \oplus B) = \mu(A) \oplus \mu(B)$$

And we must say what the norms of the variables are

$$\mu(x) :\equiv \mu(\mathsf{typ}(x)).$$

Our definition of normability, here, is modelled after the normability definition of AUT-QE (weak normability), in particular as far as the handling of 2-variables is concerned. For details see IV.4.4-IV.4.5. First we define *norm inclusion* $\subset$:

(1) α a norm $\Rightarrow \alpha \subset \tau$

(2) $\alpha \subset \beta \Rightarrow (\gamma \to \alpha) \subset (\gamma \to \beta)$

Then we say that A *fits in* B (notation A fin B) if:

$$\mathrm{degree}(A) = 3 \Rightarrow \mu(A) \equiv \mu(B)$$

$$\mathrm{degree}(A) = 2 \Rightarrow \mu(A) \subset \mu(B)$$

Now we define the norm of constant expressions

$$\bar{A} \text{ fin } \bar{C}[\![\bar{A}]\!] \Rightarrow \mu(c(\bar{A})) :\equiv \mu(\mathsf{typ}(c)[\![\bar{A}]\!])$$

$$\bar{A} \text{ fin } \bar{C}[\![\bar{A}]\!] \Rightarrow \mu(d(\bar{A})) := \mu(\mathsf{def}(d)[\![\bar{A}]\!])$$

where $\bar{x} \in \bar{C}$ is the context of the scheme, in which c (resp. d) was introduced.

We want to show that correct expressions are normable, and of course that whenever $A \in B$, A fits in B. In view of the instantiation rule and the fact that norms can change under substitution (for 2-variables) we prove, as in Ch. IV.4.5 a kind of *normability under substitution*.

Theorem: If $\bar{A}$ fin $\bar{B}[\![\bar{A}]\!]$, $\bar{y} \in \bar{B} \vdash C \in D$ then $C[\![\bar{A}]\!]$ fin $D[\![\bar{A}]\!]$ (note that "fitting in" implies the normability of the expressions involved)

Proof: Ind. on correctness.

Corollary: $\vdash C \ \mathsf{E}\ D \Rightarrow C$ fin D (so C, D normable)

5.4.2 Note: By the above defined concept of normability lots of expressions become normable which are certainly not correct in AUT-Pi. E.g. $\{A\}(\Pi([x{:}B]C))$, with $\mu(A) \equiv \mu(B)$, and $(\Sigma(B))_{(1)}$, with $\mu(B) \equiv \beta_1 \to \beta_2$. This is a consequence of the fact that AUT-Pi is handled just like AUT-QE: Π's are (as regards norms) ignored, and Σ's are in some sense identified with pairs.

5.4.3 Extending the SN-result to AUT-Pi

Clearly the presence of non-reducing constants such as Σ, Π, (for 2-expressions) and τ does not harm the SN-results of the previous sections. We just have to add δ-reduction. The *substitution* (resp. *replacement theorem for* SN can easily be extended because δ-contractions in $B[\![x/A]\!]_{(LR)}$ either take place inside A or can be mimicked in B already. Then we can proceed as in IV.4.6 or directly prove B normable $\Rightarrow$ B SN, by ind. on (1) date(B), (2) $\ell(B)$. The new case is when $B \equiv d(\bar{C})$. The C_i's are SN by ind. hyp. (2). Further we want that def(d)$[\![\bar{C}]\!]$ is SN. Well, def(d) is SN by ind. hyp. (1) and def(d)$[\![\bar{C}]\!] \equiv$ def(d)$[\![C_1]\!]\cdots[\![C_n]\!]$. So by iterated use of the substitution theorem we are done. Later we can add $\sigma\eta$, by pp.

Alternatively we can extend the SN proof by *computability* to the present case, viz. by leaving the definition of computability unmodified and prove *computability under substitution* by ind. on (1) date, (2) length. In particular let $A_1,\cdots,A_k \in C$ of the right norms, let * stand for $[\![\bar{x}/\bar{A}]\!]$, let $B_1^*,\cdots,B_n^* \in C$. Then we must prove that $d(\bar{B})^* \in C$. The B_i^*'s are SN. By ind. hyp. (1) def(d) is cus, so def(d)$[\![\bar{B}^*]\!] \in C$, so SN. Further, if $d(B^*) \geq [y]E$ (or $\langle E,F\rangle$ etc.) then this reduction passes through def(d)$[\![B^*]\!]$ (which was in C).

So, finally we have $\beta\pi+\sigma\eta\delta$-SN for AUT-Pi.

VIII.6 Some additional remarks on AUT-Pi

6.1 The connection between AUT-QE and the abstr part of AUT-Pi

Here the abstr part of AUT-Pi is the part generated by the general rules (2.2.1, 2.2.2) and the specific rules group I (2.3). If it were not for the role of Π, and the rule of product formation, this part of AUT-Pi would be identical to AUT-QE.

In the introduction to this chapter we mentioned already that the rule of type-inclusion is somewhat stronger than the rule of product formation. This means that the obvious translation of AUT-Pi, viz. just skipping the Π's produces correct AUT-QE, but not all of AUT-QE. Namely, without Π, the rule of product formation becomes

$$\phi \ E \ [x{:}\alpha]\tau \Rightarrow \phi \ E \ \tau \qquad \text{(I)}$$

which is just a specific instance of the type-inclusion rule

$$\phi \ E \ [\bar{y}{:}\bar{\beta}][x{:}\alpha]\tau \Rightarrow \phi \ E \ [\bar{y}{:}\bar{\beta}]\tau \qquad \text{(II)}$$

Let us see whether sensible use of (I) can yield something like (II). So let $\phi \ E[\bar{y}{:}\bar{\beta}][x{:}\alpha]\tau$. Then $\bar{y} \ E \ \bar{\beta} \vdash \{\overleftarrow{\bar{y}}\}\phi \ E \ [x{:}\alpha]\tau$ (where $\overleftarrow{\bar{y}}$ consists of the y_i's in the reversed order). So by (I) $\bar{y} \ E \ \bar{\beta} \vdash \{\overleftarrow{\bar{y}}\}\phi \ E \ \tau$, and by iterated use of the abstr rule we get $\vdash \phi^+ \ E \ \tau$ with $\phi^+ \equiv [\bar{y}{:}\bar{\beta}]\{\overleftarrow{\bar{y}}\}\phi$. Clearly

$$\phi^+ \geq^2_\eta \phi$$

which indicates that AUT-QE is not a very essential extension of the image of AUT-Pi under the translation. Compare De Bruijn [15, 17].

6.2 The CR problem caused by ε

In Ch. II we gave a counter example for $\beta\varepsilon$-CR. Namely $[x]x$ and $[y]i_1(y) \oplus [y]i_2(y)$ are distinct $\beta\varepsilon$-equal normal forms (just two different ways to write identity on a $\oplus$-type). This suggests to save CR by adding εalt (see 2.6)

$$[x]B[\![i_1(x)]\!] \oplus [x]B[\![i_2(x)]\!] > [x]B$$

However, εalt and + interfere in a nasty way:

$[x](\cdots\{x\}F\cdots) \oplus [x](\cdots\{x\}G\cdots) <_{++} [x](\cdots\{i_1(x)\}(F \oplus G)\cdots) \oplus$

$[x](\cdots\{i_2(x)\}(F \oplus G)\cdots) >_{\varepsilon} [x](\cdots\{x\}(F \oplus G)\cdots)$, so this does not help.

In principle, CR is not too important for our purpose, we rather need a good decision procedure for definitional equality. Just like (in V.4) we suggested to implement η-equality by the rule

$$\{x\}F\ Q\ G \Rightarrow F\ Q\ [x]G$$

we *conjecture* here that we could generate full equality (including ε) by adding

$$\{i_1(x)\}F\ Q\ \{x\}G,\ \{i_2(x)\}F\ Q\ \{x\}H \Rightarrow F\ Q\ G \oplus H$$

But in order to guarantee the well-foundedness of such an algorithm, we need of course some kind of strong normalization result, which applies in the present situation.

The general pattern of the counterexample to +εalt-CR reads

$$[x]\mathcal{O}(\{x\}F) \oplus [x]\mathcal{O}(\{x\}G)\ Q\ [x]\mathcal{O}(\{x\}(F \oplus G))$$

where $\mathcal{O}$ is a very general operation on expressions. This shows that extensional equality generates the equality induced by permutative reductions (sec. 4.3) $\mathcal{O}(\{A\}([x]B \oplus [x]C))\ Q\ \{A\}[x]\mathcal{O}(\{x\}([x]B \oplus [x]C))$

$Q\ \{A\}([x]\mathcal{O}(\{x\}[x]B) \oplus [x]\mathcal{O}(\{x\}[x]C))\ Q\ \{A\}([x]\mathcal{O}(B) \oplus [x]\mathcal{O}(C))$. E.g.,

$\{D\}\{A\}([x]B \oplus [x]C) <_{\beta} \{A\}[x]\{D\}\{x\}([x]B \oplus [x]C)$

$>_{\varepsilon alt} \{A\}([x]\{D\}\{i_1(x)\}([x]B \oplus [x]C) \oplus [x]\{D\}\{i_2(x)\}([x]B \oplus [x]C)) >_{+'+'}$

$\{A\}([x]\{D\}B \oplus [x]\{D\}C)$, q.e.d.

Conversely, we might generate part of the ε-equality by adding *general* permutative reductions, paying due attention to the thus arising SN problem.

6.3 The SN-problem caused by ε

We strongly believe that SN holds for the full AUT-Pi reduction (including ε), and that there are just some technical problems which prevent the proofs of the preceding section to apply to that situation. We briefly sketch why each of the three proofs fails in presence of ε.

The problem with the first proof (5.1) is that the dead end set for, e.g., βε-reduction is not so easy to describe. E.g. $[y]\{\{i_1(y)\}x\}F \oplus [y]\{i_2(y)\}F$ is a typical dead end for βε. Of course βη- or βσ-dead ends are not manageable either, but ση can be included afterwards, using pp.

Then the second proof (5.2). An ε-redex $[y]\{i_1(y)\}F \oplus [y]\{i_2(y)\}F$ can be created by substitution $[\![x/A]\!]$ in two different ways: (1) from $x \oplus [y]\{i_2(y)\}F$, $A \equiv [y]\{i_1(y)\}F$ (and similar with the right hand part), (2) from $[y]\{i_1(y)\}F_1 \oplus [y]\{i_2(y)\}F_2$, $F_1[\![A]\!] \equiv F$, $F_2[\![A]\!] \equiv F$. In case (1) we are suggested to replace $x \oplus [y]\{i_1(y)\}F$ by a single variable z, and to introduce a new substitution $[\![z/F]\!]$. However, $\ell(\mu(z)) > \ell(\mu(x))$, which does not fit in the proof at all. But we can remove this case by just considering AUT-Pi_0. Case (2) does not pose a problem: the substitution plus reduction can be simulated by reduction plus substitution, starting from $[y]\{i_1(y)\}F_0 \oplus [y]\{i_2(y)\}F_0$, where both F_1 and F_2 can be constructed from F_0 by substituting A for *some of* free x's. Besides, the second proof is based on *replacement*. This means that the ε-redex above can also be created from, e.g., (3) $[y]\{x\}F \oplus [y]\{i_2(y)\}F$, with $A \equiv i_1(y)$, or (4) $[y]\{i_1(x)\}F \oplus [y]\{i_2(y)\}F$. These two expressions do not reduce, unless we switch to a generalized form of ε_{alt} (which does not solve the problem, though - see below).

Finally the computability method (5.3) fails because the property: $F \in C$, $G \in C \Rightarrow F \oplus G \in C$ is not so easy anymore. For, let $F \geq [x]\{i_1(x)\}[y]D$, $G \geq [x]\{i_2(x)\}[y]D$. Then we just know that $A \in C \Rightarrow D[\![i_1(A)]\!] \in C$, $D[\![i_2(A)]\!] \in C$, but we want that $D[\![A]\!] \in C$ for general $A \in C$.

We have tried to adapt the second SN-proof to this situation, viz. by restricting to AUT-Pi_0, and by introducing a liberal version of ε_{alt}, named ε'.

$$\varepsilon'\colon\ [y]F[\![i_1(y)]\!] \oplus G > [y]F,\ G \oplus [y]F[\![i_2(y)]\!] > [y]F$$

This can be considered a kind of improper reduction in the sense that it identifies expressions which in the intuitive interpretation do correspond to different objects. A typical way of creating a new ε'-redex is, e.g., from $[y]x \oplus G$ by the replacement $[\![x/i_1(y)]\!]_{LR}$, reducing to $[y]y$. One can indeed mimick this by first reducing to $[y]x$, and then apply a new replacement, viz. $[\![x/y]\!]$. But the norm of this new x is longer than that of the old one.

REFERENCES

[1] P. Andrews, Resolution in type theory, Journ. of Symb. Logic, 36 (1971), p. 414-432

[2] H.P. Barendregt, Some extensional term models for combinatory logics and λ-calculi, Ph.D. Thesis, Utrecht 1971

[3] H.P. Barendregt, Pairing without conventional restraints, Zeitschr. f. math. Logik u. Grundl. d. Math. 20 (1974), p. 289-306

[4] H.P. Barendregt, The type free λ-calculus, in: Handbook of Math. Logic, Barwise (ed.), North Holland, Amsterdam 1977

[5] H.P. Barendregt, J. Bergstra, J.W. Klop and H. Volkema, Representability in lambda algebras, Indag. Math. 38 (1976, p. 377-387

[6] Ch. Ben-Yelles, Article to be published in Zeitschr. f. Math. Logik u. Grundl. d. Math. 1980

[7] S. de Boer, De ondefinieerbaarheid van Church' δ-functie in de λ-calculus en Barendregt's lemma, stageverslag, Eindhoven 1975

[8] N.G. de Bruijn, The mathematical language AUTOMATH, its usage and some of its extensions, in: Symposium on Automatic Demonstration (IRIA, Versailles 1968), Lect. Notes in Math., 125, p. 29-61, Springer, 1970

[9] N.G. de Bruijn, AUT-SL, a single-line version of AUTOMATH, AUT20*), 1971

[10] N.G. de Bruijn, Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem, Indag. Math. 34 (1972), p. 381-392

[11] N.G. de Bruijn, AUTOMATH, a language for mathematics, notes (by B. Fawcett) of a series of lectures (Séminaire de Math. Supér., Montréal, 1971), Montréal 1973

[12] N.G. de Bruijn, Set theory with type restrictions, in: Infinite and finite sets I, Hajnal et al. (eds.), p. 205-214, Colloquia Math. Soc. Jan. Bolyoi 10, 1975

[13] N.G. de Bruijn, The AUTOMATH Mathematics Checking Project, in: Proc. of the symp. APLASM I (Braffort volume), Braffort (ed.), AUT34, 1973

[14] N.G. de Bruijn, Some extensions of AUTOMATH: The AUT-4family, AUT44, 1974

[15] N.G. de Bruijn, Some auxiliary operators in AUT-Π, AUT51, 1977

*) Items marked AUT... are reports distributed by the AUTOMATH group, Dept. of Math., Techn. Univ. Eindhoven.

[16] N.G. de Bruijn, Lambda calculus with namefree formulas involving symbols that represent reference transforming mappings, Indag. Math. 40 (1978), p. 348-356

[17] N.G. de Bruijn, AUT-QE without type-inclusion, AUT56, 1978

[18] N.G. de Bruijn, A note on weak diamond properties, AUT57, 1978

[19] N.G. de Bruijn, A namefree λ-calculus with facilities for internal definitions of expressions and segments, AUT59, 1978

[20] N.G. de Bruijn, A survey of the project AUTOMATH, in: Combinatory Logic, lambda calculus and formal systems (Curry Festschrift), Hindley and Seldin (eds.), Ac. Press 1980

[21] J.P. Bulnes-Rozas, GOAL: A goal oriented commend language for interactive proof construction, Ph. D. Thesis, Stanford A.I. Lab., Memo AIM-328, Stanford 1979

[22] M. Coppo, M. Dezani-Ciancaglini and B. Venneri, Functional characters of solvable terms, Zeitschr. f. Math. Logik u. Grundl. d. Math., to appear

[23] M. Coppo and M. Dezani-Ciancaglini, A new type assignment for λ-terms, Archiv. Math. Logik 19 (1978), p. 139-156

[24] A. Church, A formulation of the simple theory of types, J. of Symb. Logic 5 (1940), p. 56-68

[25] H.B. Curry and R. Feys, Combinatory logic I, North Holland, Amsterdam 1958

[26] H.B. Curry, J.R. Hindley and J.P. Seldin, Combinatory logic II, North Holland, Amsterdam 1972

[27] D.T. van Daalen, A description of AUTOMATH and some aspects of its language theory, in: Braffort volume (see [13]) reprinted in [37]

[28] G. Gentzen, Untersuchungen über das logische Schliessen, Math. Zeitschr. 39 (1935), p. 176-210, p. 405-431

[29] G. Gentzen, Die Widersprachsfreiheit der reine Zahlentheorie, Math. Annalen 112 (1936), p. 493-565

[30] J.Y. Girard, Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types, in: Second Scand. Logic Symp. (Oslo Volume), Fenstad (ed.), North Holland, Amsterdam 1971

[31] J.Y. Girard, Interprétation functionnelle et élimination des coupures de l'arithmétique d'ordre supérieure, Thèse, Paris 1972

[32] M. Gordon, R. Milner and C. Wadsworth, Edinburgh LCF, A mechanical logic of computation, Edinburgh 1979, submitted to Springer Lect. Notes in Comp. Sc.

[33] J.R. Hindley, Combinatory reductions and lambda reductions compared, Zeitschr. f. Math. Logik u. Grundl. d. Math. 23 (1979), p. 169-180

[34] W.A. Howard, The formulae-as-types notion of construction, unpubl. 1969, to appear in Curry Festschrift (see [20])

[35] H. Jervell, A normal form in first order arithmetic, in Oslo volume (see [30])

[36] L.S. van Benthem Jutting, A normal form in a λ-calculus with types, in: Mitt. d. Gesellsch. f. Math. u. Datenverarb. Bonn, 17, Tagung üb. form. Sprachen u. Programmiersprachen, Oberwolfach 1971

[37] L.S. van Benthem-Jutting, Checking Landau's "Grundlagen" in the AUTOMATH system, Ph. D. Thesis Eindhoven 1977, Math. Centre Tracts, 83, Amsterdam 1979

[38] L.S. van Benthem-Jutting and R.M.A. Wieringa, Representatie van expressies in het verificatieprogramma YERA 1979, Internal Report, Eindhoven 1980

[39] S.C. Kleene, Introduction to Metamathematics, Van Nostrand, New York 1952

[40] D. Leivant, Strong normalization for arithmetic (variations on a theme of Prawitz), in: Proof theory symposium Kiel 1974, Lect. Notes in Math, 500, p. 182-197, Springer 1975

[41] J.J. Lévy, Réductions sures dans le lambda-calcul, Thèse 3° Cycle, Paris 1974

[42] J.J. Lévy, An algebraic interpretation of the λβκ-calculus and a labelled λ-calculus, in λ-calculus and computer science theory (Rome volume), C. Böhm (ed.), Lect. Notes in Comp. Sc., 37, p. 147-165

[43] C.R. Mann, The connections between proof theory and category theory, Ph. D. Thesis, Oxford 1973

[44] P. Martin-Löf, Hauptsatz for the theory of species, in: Oslo Volume (see [30]), p. 217-234

[45] P. Martin-Löf, An intuitionistic theory of types, Unpubl. 1972

[46] P. Martin-Löf, An intuitionistic theory of types, predicative part, in: Logic Coll. 73, Rose and Sheperdson (eds.), North Holland, Amsterdam 1975

[47] P. Martin-Löf, About models for intuitionistic type theory and the notion of definitional equality, in: Proz. of the third Scand. Logic Symp., Karger (ed.), North Holland, Amsterdam 1975

[48] G. Mitschke, λ-Kalkül, δ-Konversion und axiomatische Rekursions-Theorie, Habilit. Schr., Darmstadt 1976

[49] R.P. Nederpelt, Lambda-Automath, AUT21, 1971

[50] R.P. Nederpelt, Lambda-Automath II, AUT22, 1971

[51] R.P. Nederpelt, Strong normalization for a typed lambda calculus with lambda structured types, Ph. D. Thesis, Eindhoven 1973

[52] H. Osswald, Ein syntaktischer Beweis für die Zulässigkeit der Schnittregel im Kalkül von Schütte für die intuitionistischen Typenlogik, Manuscr. Math. 8 (1973), p. 243-249

[53] P. Penning, Automath bewijzen voor tautologieën, Stageverslag, Eindhoven 1977

[54] G. Plotkin, Lambda-definability in the full type hierarchy, in: Curry Festschrift (see [20])

[55] W. Pohlers, Ein starkes Normalisationssatz für die intuitionistischen Typen, Manuscr. Math. 8 (1973), p. 371-387

[56] G. Pottinger, Letter to Prawitz, April 18, 1977

[57] G. Pottinger, On analysing relevance constructively, Studia Logica 38 (1979), p. 171-185

[58] G. Pottinger, A type assignment to the strongly normalizable λ-terms, in: Curry Festschrift (see [20])

[59] D. Prawitz, Natural Deduction, a proof theoretic study, Almquist and Wiksell, Stockholm 1965

[60] D. Prawitz, Ideas and results in Proof Theory, in: Oslo Volume (see [30]), p. 235-307

[61] L.E. Sanchis, Functionals defined by recursion, Notre Dame J. of Formal logic 8 (1967), p. 161-174

[62] D. Scott, Constructive validity, in: Symp. on Automath. Demonstration (see [8]), p. 237-275

[63] J.P. Seldin, Review of [10], Journal of Symb. logic 40 (1975), p. 470

[64] J.P. Seldin, A theory of generalized functionality I, Unpubl. 1976

[65] J. Staples, Church-Rosser theorems for Replacement Systems, in: Algebra and Logic, Lect. Notes in Math. 450, p. 291-307, Springer 1975

[66] J. Staples, A lambda calculus with naive substitution, Unpubl. Brisbane 1977

[67] S. Stenlund, Combinators, λ-terms and proof theory, Reidel 1972

[68] W.W. Tait, Intentional interpretation of functionals of finite types, Journ. of Symb. Logic. 32 (1967), p. 198-212

[69] A.S. Troelstra et al., Metamathematical Investigation of Intuitionistic Arithmatic and Analysis, Lect. Notes in Math., Springer 1973

[70] R.C. de Vrijer, Big trees in a λ-calculus with λ-expressions as types, in: Rome colume (see [42]), p. 202-221

[71] R.C. de Vrijer, A syntactic model for λ-calculus with surjective pairing, Ph. D. Thesis, Eindhoven, to appear

[72] C. Wadsworth, Semantics and pragmatics of the lambda calculus, Ph. D. Thesis, Oxford 1972

[73] R.W. Weihrauch, A users manual for FOL, Stanford A.I.-lab. memo 235, Stanford 1977

[74] R.M.A. Wieringa, Binaire optelling en vermenigvuldiging in AUT-QE, Stageverslag, Eindhoven 1976

[75] I. Zandleven, A verifying program for AUTOMATH, Braffort volume (see [13]), AUT36, 1973

[76] J. Zucker, Cut-elimination and normalization, Annals of Math. Logic 7 (1974), p. 1-112

[77] J. Zucker, Formalization of classical mathematics in AUTOMATH, Actes du coll. intern. de logic, Guillaume (ed.), Clermont-Ferrand 1975

[78] D.A. Turner, Another algorithm for bracket abstraction, Journ. of Symb. logic 44 (1979), p. 267-270

[79] R.C. de Vrijer, "Stelling" to his [71]

SAMENVATTING

In het Automath project zijn een aantal *wiskundige talen* ontwikkeld die geschikt zijn om grote stukken wiskunde zó weer te geven dat een computer de correctheid van de wiskundige redenering kan controleren. Het programma dat deze controle verzorgt wordt *verificator* genoemd. De belangrijkste Automath talen zijn AUT-68, AUT-QE en AUT-Pi.

De Automath talen zijn gebaseerd op systemen van *gegeneralizeerde getypeerde λ-calculus*. De *taaltheorie* houdt zich bezig met syntaktische kwesties, betreffende de *definitiegelijkheid*, de *reductie-relatie* en de *typerings-relatie* in deze systemen. Drie belangrijke eigenschappen waarop de taaltheorie zich richt zijn: (*sterke*) *normalisatie*, *gesloten-heid* en *Church-Rosser eigenschap*. Deze eigenschappen zijn onder meer van belang om de correcte werking van de verificator te kunnen aantonen.

Dit proefschrift kan worden opgevat als een voortzetting en een aanvulling op taaltheoretisch werk van Nederpelt en de Vrijer. Hoofdstuk I geeft een overzicht van het Automath project, gaat uitvoerig in op de rol van de taaltheorie binnen het project, en wordt besloten met een uitgebreide samenvatting van het proefschrift. Hoofdstuk II bevat de nodige preliminaria. Hoofdstuk III behandelt de theorie van *afkortingen*. In de hoofdstukken IV, V en VI worden achtereenvolgend de drie genoemde belangrijke eigenschappen bewezen voor AUT-68, AUT-QE en nog enige varianten. Hoofdstuk VII gaat in op de theorie van Nederpelt's Automath systeem Λ. De drie belangrijke eigenschappen worden bewezen (dit bevestigt twee vermoedens uit Nederpelt's proefschrift), en tevens wordt de Vrijer's *grote-boom stelling* van een nieuw bewijs voorzien. Hoofdstuk VIII bevat de theorie van AUT-Pi. Geslotenheid wordt bewezen voor het volledige AUT-Pi, alsmede sterke normalisatie en Church-Rosser voor een deelsysteem van AUT.Pi.

Sommige resultaten uit het proefschrift zijn niet alleen van toepassing op Automath maar ook van belang in de λ-calculus, en, door de *formulae-as-types* interpretatie, voor *bewijstheorie*.

CURRICULUM VITAE

De schrijver van dit proefschrift werd in 1949 in Bergeijk geboren. Na het eindexamen gymnasium β aan het Lorentzlyceum te Eindhoven, begon hij in 1966, op aanraden van Prof.Dr. J.J. Seidel, aan de studie voor wiskundig ingenieur aan de Technische Hogeschool Eindhoven. In juni 1972 studeerde hij met lof af, bij Prof.Dr. N.G. de Bruijn. Na zijn afstuderen was hij tot eind 1976 verbonden aan het Project Wiskundige Taal AUTOMATH, als wetenschappelijk medewerker in dienst van de Nederlandse Organisatie voor Zuiver-Wetenschappelijk Onderzoek (Z.W.O.), en onder leiding van Prof. de Bruijn.

Sinds maart 1977 is hij wetenschappelijk medewerker bij Prof. Ir. W. Baarda, op de afdeling Geodesie van de Technische Hogeschool Delft.

Adress of the author:
Department of Geodesy
Technological University
Thijsseweg 11
Delft