Weak Type Theory : a formal language for mathematics

Weak Type Theory:
a formal language for mathematics

Rob Nederpelt

Abstract

In this report we propose a formal language for expressing mathematics, called Weak Type Theory ('WTT'). This language, as we claim, is close to the usual way in which mathematicians express themselves in writing. Therefore it is relatively easy to use and has a high degree of reliability in the sense that it gives a true image of what the mathematician intends to say.

As a formal language, WTT has a precise syntax, which is based on linguistic categories. A derivation system for WTT helps in restricting the usage of WTT to cases which obey a number of 'natural' linguistic requirements. Both the syntax and the derivation systems are extensively exposed in this report, with a number of illustrative examples.

It must be emphasized that WTT is still very weak as regards correctness: the rules of WTT only concern *linguistic* correctness, its types are of a pure linguistic character, ensuring that the formal translation into WTT is satisfactory *as a readable, well-organized text*. On the other hand, all information concerning the real essence of correctness, namely its *logical-mathematical aspects* of truth, trustworthiness and applicability, are disregarded. Only in a later stage, when extending a WTT-text to, e.g., full Type Theory, logical and mathematical correctness start playing their leading parts. We discuss a number of more or less problematic topics which have to be solved in order to design such desirable refinements of WTT.

Dedicated to N.G. de Bruijn

Chapter 1

Introduction

The way in which mathematical ideas are usually expressed in writing (books, papers and the like) is *informal*, in the sense that there exists no prescribed syntax for the presentation of the mathematical content. We give the name *Common Mathematical Language* (or *CML*) to this linguistic machinery which mathematicians preferably use to express mathematical content and to communicate with their fellow mathematicians. We concentrate hereby on *written* specimens of CML.

We think that it is useful for several applications to also have a *formal* language at hand for the same communication purposes. Such a language may act as a substitute for CML. In section 1.4 we discuss possible uses and the benefits thereof.

We propose a system called *Weak Type Theory* (abbreviated *WTT*), intended to be a formal language for expressing mathematical content. What we aim at in developing WTT is simply to keep as much as possible of the advantages, and to remove as much as possible of the disadvantages of CML.

In doing so, we inevitably had to make design decisions. We defend our decisions below, knowing that it is likely that our choices can be disputed.

1.1 A comparison between Common Mathematical Language and Weak Type Theory

We start with an overview of what we see as the main advantages and disadvantages connected with CML.

First we list the most salient *ADVANTAGES*:

Expressive CML is suited for expressing all kinds of mathematical mental constructs, ranging from mathematical entities and relations between them, to mathematical reasonings and theories. It also accommodates the use of relevant mathematico-linguistic categories, such as definitions, theorems and proofs.

Time-honoured It has been polished and refined by intensive use and it is rooted in a long tradition.

Satisfactory It has the tacit approval of a large community of mathematicians, since it has proved and still proves to be an adequate communication medium.

Universal It is used all over the world and it offers a recognizable format and style to all users.

Flexible Not only can it accommodate many branches of mathematics, but it is also easily adaptable to new developments in mathematics.

However, there are also *DISADVANTAGES* in using CML:

Informal Since it is based on natural language – albeit mixed with mathematical symbols and formulas – it has no well-defined formal basis and hence it suffers of imprecision, only overcome by the understanding of the average user.

Ambiguous It has inherited the ambiguities present in natural language, such as unspecified 'discourse references' or unsolved anaphoric relations[1], including the overloading of frequently used words. Moreover, it introduces new ambiguities since the mathematical extensions to the natural language, as used in CML, are sometimes ill-defined.

Incomplete Much is left undetermined, implicit or is neglected, the writer making appeal to the intuition or common sense of the mathematical reader.

Poorly organised A common mathematical text is only partly structured in textual units, but many structural aspects are omitted or only hinted at.

Automation-unfriendly Since it is a plain text only, its mathematical contents cannot be exploited to its full extent by invoking computer assistance for many different purposes.

We revisit the above lists of advantages and disadvantages, this time comparing the formal WTT with the Common Mathematical Language. First we consider the advantages:

Expressive The expressivity of WTT is comparable to that of CML.

Time-honoured WTT is novel, but it respects the mathematical traditions.

Satisfactory Though we have ample experience with WTT, its satisfactory usability of course still has to be demonstrated.

Universal At the moment only potentially.

Flexible WTT has flexibility to a great extent, but not as much as CML. This is due to the fact that both language and meta-language are allowed (and often mixed) in the latter, whereas WTT, by its formal nature, only allows 'language'. So meta-language must be pushed down to the language level, which is not always easy or even impossible.

As regards the disadvantages:

Informal The main gain of WTT over CML is its formality. WTT has a well-defined syntax, consisting of a grammar and a derivation system, which provide for uniformity of expression.

Ambiguous Most of the ambiguities in the CML-texts disappear in the translation into WTT. For example, the anaphoric obscurities in language (hence also in CML) are resolved in WTT, in particular by the requirements of the strict context management of WTT.

Incomplete A text in WTT-format is still incomplete, but less so than CML. However, WTT can be translated further into stronger theories, which are more and more complete.

Poorly organised A WTT-text is clearly organized.

[1] Anaphora regard 'the use of words referring to or replacing a word used earlier in a sentence, to avoid repetitions' (the Concise Oxford Dictionary).

Automation-friendly WTT is by its formal character already more or less fit for computerization. In translating WTT into stronger versions, the potentials for automation grow fast and promising.

1.2 The safety criterion

In our design of WTT, we try to keep close to the Common Mathematical Language. Our main reason for doing this is *reliability*: it is of uttermost importance that the formal version of a piece of mathematics covers exactly the intended contents of the original CML-version. Since the latter is informal – already in the mind of the mathematician who devised it, but also in a written version – it is impossible to have an objective check on the correspondence between the original and the formalized text. Hence it is necessary to invoke a *human* judgement on this correspondence, for each piece of mathematics under consideration. This judgement can be left to the one who devised the text, but also to others. Although a hundred percent reliability cannot be attained, it is our opinion that this judgement is optimally trustworthy if the formal version is as close as possible to the informal one. We call this the *safety criterion*.

As soon as a reliable formal translation of a piece of informal mathematical text has been made, we are on solid ground. Such a translation will still be far from complete, but since it is formal, it is possible to define further translations into more complete versions. Since these subsequent translations are from formal texts to formal texts, it is much easier to check reliability: one only needs to confirm oneself of the reliability of the translation *procedure*, not once and again, but once *and for all*.

1.3 Features of WTT

Some of the features of WTT making it useful as a formal language for mathematics are the following:

- WTT respects all linguistic categories which a mathematician usually employs, so not only the notions *set* and *element* of a set, taken from the *mathematical* language, but also the notions *noun* and *adjective*, already present in *natural* language and amply used in CML. This brings a WTT-text nearer to the intuition of a mathematician, since the fine details of a piece of mathematics are better accounted for than in a purely set-theoretic setting.

- In this respect it should be observed that the average mathematician does *not* have a purely set-theoretic view on mathematics! A collective notion (a 'type') is not always identifiable with a 'set'. Moreover, the combination notion-subnotion differs in meaning and content from the combination set-subset. Also, predicates can be identified with subsets, but again, there are subtle differences between the two.

  In order to respect the fine details of a CML-text, we preserve categories like 'noun' and 'adjective', albeit that, on first view, they seem superfluous. Another reason for maintaining these linguistic categories is, that we want to minimize communication losses in this first formalization step (from CML to WTT). We do this by keeping

as much of the linguistic and structural features of CML as possible, thus leaving all possibilities open. Sieving possibly superfluous ingredients can always happen in a later stage, for example in a further translation into Type Theory (see below).

- As a consequence of the above, we believe that most of the ambiguities in CML are representable and distinguishable in WTT. This is important for any attempt to obtain a successful disambiguation.
- WTT also accommodates all sorts of *binders* (such as $\bigcup$, $\sum$ or $\forall$).
- Incorporated in WTT are basic notions as: *assumption*, *declaration of a fresh variable*, *definition* and *statement* (covering notions like *theorem* and *step in a proof*).
- WTT is consistently structured and uses *contexts*, which are lists of assumptions and variable declarations 'setting the stage' for a statement or a definition. These contexts reflect the introductory statements usually expressed in mathematics as e.g. *'Let ...'* or *'Assume ...'*.
- The overall form of a text in WTT is a so-called *book*, being a sequence of so-called *lines*, which are statements and definitions, each embedded in their own context. Each WTT-line can be seen as the translation of a mathematical expression stating that something 'holds' in a certain context. Hence, one finds in a book a – probably connected – fragment of mathematical knowledge, consisting of lines expressing mathematical facts (like theorems, lemma's, but also steps in a reasoning or in a proof) and lines expressing definitions (possibly in a context).
- A text transposed in WTT gives an accurate image of the structure of the original CML-text. The line-by-line development of a theory is reflected in the sequence of lines in a WTT-book. These lines are interdependent in the sense that each one may call upon previous ones (and most often does so).

1.4 The usefulness of WTT

Of course, the author of this report does not expect that mathematicians convert themselves to using WTT exclusively. On the contrary, the common mathematical language, as it can be found in books and papers, is good enough and, moreover, usable and familiar to a high degree. However, WTT is formal and so close to the usual linguistic format used by mathematicians, that it can easily be adopted by mathematicians as 'a second language'. This can be advantageous for the following reasons:

- In complex situations, the 'second language' WTT can help the mathematician to precisely identify the (logico-mathematical) structure in which he or she works.
- It can also help a person to be fully aware of the complexity of a mathematical notion, structure or reasoning, in order to better understand the situation. This holds for experts, but also for teachers and students.

- By its uniformity, WTT provides an excellent basis for communication. For example, it enables that many persons work productively on the same task. Also, the text administration of a mathematical project can get a firm basis by the use of WTT.

- WTT does not restrict the mathematician to a purely set-theoretic environment, since the full linguistic apparatus of mathematics is faithfully and respectfully represented.

  In this respect it is interesting to observe that – although many mathematicians 'believe' that their mathematics is based on (or can be formally expressed in) set theory – the actual formalization into set theory is more problematic than expected. Such a formalization requires a set-theoretical 'coding', which causes a certain distance between CML and its set-theoretical translation. Moreover, a choice for the set-theoretical framework has consequences concerning content and therefore restricts the user in several ways. In both aspects, WTT is nearer to the original mathematical content: (1) their mutual distance is smaller, and (2) by using WTT, the choice for a foundational framework (set-theoretical or other) has not yet been made.

  Therefore, it is the author's conviction that WTT obeys better to the safety criterion as exposed in section 1.2.

- WTT may also act as a *lingua franca* or a *mathematical vernacular*[2] for mathematicians, since WTT enables a mathematician to express mathematical contents in a uniform way. The advantage of WTT over the ordinary way of expressing mathematics (half in formal language, half in a natural language like English), is that WTT is more clearly structured. Moreover, its use forces the mathematics writer to think better about the interdependencies of the notions used, in particular as regards contexts and instantiations of constants.

- One can also say that WTT is a *specification language* for mathematics, since it enables the mathematician to explicitly specify which mathematical notions, definitions, statements, theorems and proofs he or she likes to use and/or communicate to others. In this respect, WTT resembles specification languages in computer science, which also enable one to formally represent the 'requirements'. In computer science, a specification is 'realized' in a computer program. In mathematics, a WTT-book can be 'realized' in a 'type-theoretic program', being a sequence of lines obeying the rules of some system of Type Theory.

- Yet another view on WTT is, that a WTT-book acts as a kind of *(mathematical) discourse representation structure*, bringing a number of structural relations to surface which are only implicitly present in CML. In this respect, WTT is a distant relative of Kamp's Discourse Representation Theory ([Kamp and Reyle 93]), well-known in the field of linguistics.

[2]The name 'mathematical vernacular' was coined by de Bruijn, who was the first to develop such 'a language for mathematics' (see [De Bruijn 87]). 'Vernacular' means: 'the language of a particular group' (in this case: the mathematicians).

- WTT is *easy to learn and easy to write*, as shown by experiences, over a long period of time, with mathematics and computer science students at Eindhoven University of Technology.

Moreover, as said before, WTT may open the possibility for the mathematical researcher to get all kinds of computer help in the development of his or her ideas. One can think of the following aspects:

- *Verification* of mathematical theories, e.g. by a type-theory-based computer program. This requires a translation in one or more steps of a WTT-text into Type Theory and the use of some typechecker. This translation is greatly a technical matter and may be done by a Type Theory expert, not necessarily the mathematician who wrote the WTT-text. This leads to 'separation of concerns', thus relieving the mathematician from the cumbersome task to fill in the (possibly uninteresting) details. Another possibility is that an intelligent computer programme does (part of) the translation, in interaction either with the mathematician who delivered the WTT-text, or with a type-theoretic expert.

- *Documentation* of bodies of checked mathematical texts in an archive, a data base which is publicly accessible. One of the 'views' available for the inspection of such a database could be WTT.

- Computer assistance in the *development* of mathematics. In this case, a mathematician may use WTT as a 'rough' formal language in which he/she expresses ideas and conjectures. A computer program translates the WTT-text into Type Theory, in direct communication with the mathematician. The computer program keeps track of all 'holes' (open places in definitions and in the reasoning) and 'proof obligations'. So the mathematician knows exactly what the status is of the theory-in-construction and where it must be adapted and/or extended to make it a complete and reliable theory.

1.5 Contents of this report

In chapter 2 we introduce WTT, a formal grammar for informal mathematics. The syntax describes the systematic construction of a book from its atoms (variables, constants, binding symbols), via phrases (viz. terms, sets, nouns and adjectives) and sentences (viz. statements and definitions) up to contexts, lines and books. (The syntax rules are listed in Appendix A.)

In chapter 3 we give a derivation system for WTT, following the syntax described in chapter 2. (The derivation rules are summarized in Appendix B.)

A number of instructive examples are given in chapter 4. In chapter 5 we discuss the relation between WTT and Type Theory.

The final remarks in Chapter 6 include a comparison with other work and a discussion of future work.

Part of this report, in particular the syntax given in Chapter 2, has been published before ([Nederpelt and Kamareddine 200X]). Another part, concerning Chapter 3, has been submitted for publication ([Kamareddine and Nederpelt 2001]).

Chapter 2

Abstract syntax for WTT

In this chapter, we develop a syntax for Weak Type Theory, based on linguistic categories. These categories include nouns and adjectives, which usually are not incorporated in formalizations of mathematics. Constants are treated as 'first-class citizens', as are binders like $\sum$ and $\bigcup$. With a view to the categories 'nouns' and 'adjectives', we introduce a number of extra binders, to facilitate linguistic constructions. On the sentence level, definitions play a prominent role in WTT. They come in various forms and formats, reflecting the habitual ways in which mathematicians use the definition-mechanism.

Like in Type Theory, contexts are an important ingredient of WTT, giving the necessary immediate background for statements and definitions. This background information concerns the types of the occurring free variables and the necessary assumptions. The notion of 'line' is used to express a context and a statement or definition dependent on this context. The final entity in the WTT-syntax is the 'book', being a sequence of lines. The book is the formal counterpart of a 'mathematical text' (e.g. a theory).

2.1 Linguistic categories

In Weak Type Theory (or *WTT*) we have the following linguistic categories:

- On the *atomic* level: *variables*, *constants* and *binders*,
- On the *phrase*[1] level: *terms*, *sets*, *nouns* and *adjectives*,
- On the *sentence* level: *statements* and *definitions*,
- On the *discource*[2] level: *contexts*, *lines* and *books*.

There is a hierarchy between the four levels: atoms are used to construct phrases; both atoms and phrases are part of sentences; and discourses are built from sentences.

[1] A phrase is according to the Concise Oxford Dictionary 'a group of words forming a conceptual unit, but not a sentence'.

[2] A discourse is 'a connected series of utterances'.

The syntax given in the following sections, establishes *well-formedness* conditions for these categories. We assume that the sets of variables, constants and binders are given beforehand, hence fixed, and that they are mutually disjoint. The syntax decides about the well-formedness of phrases, statements, definitions, and also of contexts, lines and books. Hence, a phrase, statement (etc.), constructed with the syntax, can be considered a *well-formed* phrase, statement (etc.). For convenience, however, we leave this implicit in the following sections, hence we suppress the word 'well-formed' in the syntactic description of all categories mentioned.

A list of all metasymbols and abstract syntax rules introduced in this chapter, can be found in Appendix A.

2.2 Abstract syntax

We describe each of these categories below, first variables (see section 2.3) and constants (section 2.4), then binders (section 2.5), phrases (divided in terms, sets, nouns and adjectives, see section 2.6), statements (section 2.7), definitions (section 2.8) and finally contexts, lines and books (sections 2.9, 2.10 and 2.11).

We use the following standard *metasymbols* for the categories mentioned above:

level	category	symbol	representative
atomic level	*variables*	$\mathtt{V}$	x
	constants	$\mathtt{C}$	c
	binders	$\mathtt{B}$	b
phrase level	*terms*	$\mathtt{t}$	t
	sets	σ	s
	nouns	ν	n
	adjectives	α	a
sentence level	*statements*	$\mathcal{S}$	S
	definitions	$\mathcal{D}$	D
discourse level	*contexts*	$\mathbf{\Gamma}$	Γ
	lines	$\mathbf{l}$	l
	books	$\mathbf{B}$	B

In this syntax we make use of binders $\mathtt{B}$ (think of e.g. $\sum$ or $\forall$), in the following abstract format:

$\mathtt{B}_{\mathcal{Z}}(\mathcal{E})$,

where the *subscript* $\mathcal{Z}$ is a *declaration* introducing a (bound) variable and its type, e.g. $x : \mathbb{N}$ (for: 'x has type $\mathbb{N}$'). See section 2.7.1 for a formal definition of a declaration. The symbol $\mathcal{E}$ stands for an *expression*, to be specified in section 2.5. (See also below.)

Examples of formulas with binding symbols in the above format are: $\sum_{x:\{0,1,\ldots,10\}}(x^2)$ and $\forall_{x:\mathbb{N}}(x \geq 0)$.

The binding symbol for set comprehension, $\{\ldots \mid \ldots\}$, fits in this format after a slight modification, for example: write $\{x \in \mathbb{R} \mid x > 5\}$ as $\mathtt{Set}_{x:\mathbb{R}}(x > 5)$. *Because of uniformity, our standard for set notation in this report will be the latter one.*

We use abstract syntax for the description of the various categories. For example, in section 2.11 we describe the collection of all books, **B**, in abstract syntax as:

$\mathbf{B} \;=\; \emptyset \mid \mathbf{B} \circ \mathbf{l}$

Hence, a book is either 'the empty book' or a book **B** followed by a line **l**, with an open circle as separation marker. (By convention, $\emptyset \circ \mathbf{l}$ is written as **l**.) Otherwise said, a book is a finite (length ≥ 0) sequence of lines.

Notation 2.2.1 *In the abstract syntax used in this report, upper indices and lower indices play different roles.* Upper indices *are part of the symbol, but* lower indices *belong to the abstract syntax. For example, with* $\mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E})$*, we mean all constructs composed of a binder in the set* $\mathtt{B}^{\mathtt{t}}$ *(e.g. '*$\lim$*'), subscripted with a declaration from* $\mathcal{Z}$ *(e.g. '*$n : \mathbb{N}$*') and followed by an expression in* $\mathcal{E}$ *(e.g. '*$\frac{1}{n}$*') between parentheses. The superscript* $\mathtt{t}$ *attached to* $\mathtt{B}$ *says that the binders in* $\mathtt{B}^{\mathtt{t}}$ *are* term-forming*. Hence, '*$\lim_{n:\mathbb{N}}(\frac{1}{n})$*' is a* term *belonging to* $\mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E})$*.*

Other metasymbols used in this report are:

category	symbol	representative
expressions	$\mathcal{E}$	E
parameters	$\mathcal{P}$	P
typings	$\mathcal{T}$	T
declarations	$\mathcal{Z}$	Z

We give a short description of the last-mentioned metasymbols:

— *Expressions* are used in section 2.5 and represent – as we already mentioned above – the various categories that can follow a binder. An expression is a kind of *collective* category, to be precise:

$\mathcal{E} \;=\; \mathtt{t} \mid \sigma \mid \nu \mid \mathcal{S}$,

so after a binder we may find a term, a set, a noun or a statement.

— *Parameters* are discussed in section 2.4. They represent the categories on which constants may depend. Parameters also have a collective character, but without the ν of $\mathcal{E}$:

$\mathcal{P} \;=\; \mathtt{t} \mid \sigma \mid \mathcal{S}$.

— *Typings* and *declarations* are special statements, see section 2.7.1.

2.3 Variables

The set $\mathtt{V}$ of *variables* is given beforehand, infinite, and divided into three disjoint subsets:

($\mathtt{V}^{\mathtt{t}}$) Variables ranging over *terms*,

($\mathtt{V}^{\sigma}$) Variables ranging over *sets*,

($\mathtt{V}^{\mathcal{S}}$) Variables ranging over *statements*.

In abstract syntax:

$\mathtt{V} \;=\; \mathtt{V}^{\mathtt{t}} \mid \mathtt{V}^{\sigma} \mid \mathtt{V}^{\mathcal{S}}$

2.4 Constants

Apart from variables, there are *constants* in WTT. Again, this set is given beforehand and infinite. Moreover, the set is disjoint from the set of variables.

Constants play an important role in mathematical language. They are either 'primitive'[3] or they act as an abbreviation. In the latter case a constant is introduced in the left hand side of a *definition*, being a special kind of sentence (see section 2.8). Both primitive and defined constants can be *used* after having been introduced. 'Doing' mathematics *without* constants (hence without definitions) is theoretically possible but practically unfeasible.

Below we discuss the various categories of constants. We give examples in section 2.4.1.

The set $\mathtt{C}$ of constants is divided in the following five disjoint subsets:

($\mathtt{C}^{\mathtt{t}}$) Constants forming *terms*,

($\mathtt{C}^{\sigma}$) Constants forming *sets*,

($\mathtt{C}^{\nu}$) Constants forming *nouns*,

($\mathtt{C}^{\alpha}$) Constants forming *adjectives*,

($\mathtt{C}^{\mathcal{S}}$) Constants forming *statements*.

Hence, $\mathtt{C} \;=\; \mathtt{C}^{\mathtt{t}} \mid \mathtt{C}^{\sigma} \mid \mathtt{C}^{\nu} \mid \mathtt{C}^{\alpha} \mid \mathtt{C}^{\mathcal{S}}$.

Note: In this report, we use the word 'noun' for what is known in linguistics as *indefinite noun phrase*. Moreover, we use 'adjective' for *adjective phrase*.

A constant is always followed by a *parameter list*. We denote this as $\mathtt{C}(\vec{\mathcal{P}})$. This list has for each constant a fixed length ≥ 0, the *arity* of the constant. Parameters $\mathcal{P}$ are either terms, sets or statements:

$\mathcal{P} \;=\; \mathtt{t} \mid \sigma \mid \mathcal{S}$

(If the parameter list is empty we write c instead of $c(\ )$.)

2.4.1 Examples of constants

Below, we give examples for each of the six kinds of constants, with parameter lists.

Remark 2.4.1 *We can see from the examples that we often use 'sugared' versions of the combination 'constant followed by parameter list'. For example, instead of '*the centre (C)*' we write '*the centre of C*', and instead of '*$+(3,6)$*' we write the infix formula '*$3+6$*'.*

We also often write things like '$x \in \mathbb{N}$' instead of '$x : \mathbb{N}$'. In doing this, we confuse 'is element of' with 'has type'. Again, this is for easy understanding and we hope that the reader does not take offence at this obvious abuse of notation.

[3] *Primitive* constants are introduced axiomatically, they are not defined in terms of other notions. Think of the primitive set $\mathbb{N}$ of the natural numbers, the primitive function s ('successor') from $\mathbb{N}$ to $\mathbb{N}$ or the primitive element 0 in $\mathbb{N}$.

We explicitly state that these kinds of sugaring are only for the human reader, they are not part of the syntax of WTT and should be 'undone' whenever formal accuracy is at stake.

Hence, we do not incorporate a formal counterpart in WTT of what this sugaring 'is', since we want to keep our basic-WTT simple and straightforward: to decide about the instances where 'sugar' is useful and advantageous, is not easy and adding rules for sugaring therefore introduces arbitrariness. We are aware of the fact that this policy of ours undermines our claim that WTT is as close as possible to CML. We leave this dilemma to further research.

($\mathtt{C}^{\mathtt{t}}$) Constants for *terms*, with parameter lists, are e.g.:

π, the centre of C, $3+6$, the arithmetic mean of 3 and 6, $d(x,y)$, ∇f.

The constants in this example are the terms π, 'the centre', $+$, 'the arithmetic mean', d and ∇.

The parameter lists are: ( ), (C), $(3,6)$, $(3,6)$, (x,y) and (f), respectively.

($\mathtt{C}^{\sigma}$) For *sets*:

$\mathbb{N}$, A^{C}, $V \to W$, $A \cup B$.

Constants: $\mathbb{N}$, $^{\mathrm{C}}$, $\to$, $\cup$.

Parameter lists: ( ), (A), (V,W), (A,B).

($\mathtt{C}^{\nu}$) For *nouns*:

a triangle, an eigenvalue of A, a reflection of V with respect to l, an edge of $\triangle ABC$.

Constants: 'a triangle', 'an eigenvalue', 'a reflection', 'an edge'.

Parameter lists: ( ), (A), (V,l), $(\triangle ABC)$.

($\mathtt{C}^{\alpha}$) For *adjectives*:

prime, surjective, Abelian, continuous on $[a,b]$.

Constants: prime, surjective, Abelian, continuous.

Parameter lists: ( ), ( ), ( ), $([a,b])$.

($\mathtt{C}^{\mathcal{S}}$) For *statements*:

'P lies between Q and R', '$5 \geq 3$', $p \wedge q$, $\neg\forall_{x\in\mathbb{N}}(x > 0)$.

Constants: 'lies between', $\geq$, $\wedge$, $\neg$.

Parameter lists: (P,Q,R), $(5,3)$, (p,q), $(\forall_{x\in\mathbb{N}}(x > 0))$.

Note that the parameters in parameter lists are usually either *terms* or *sets*. Only in the case of statements the parameters may be *statements* as well, as is shown in the last mentioned two examples.

2.4.2 Special constants

We introduce two special constants in order to switch between the categories 'noun' and 'set'. These categories are both present and frequently used in CML and it turns out to be useful to be able to easily change from the one to the other. Note that nouns and sets are in a sense interchangeable and one could restrict oneself to only one of these categories, without losing expressive power (as is actually done in the set-theoretic formalization). In section 1.3 we explained why we nevertheless admit *both* categories.

The first special constant is $\uparrow$, belonging to the category $\mathtt{C}^{\sigma}$. The second one is $\downarrow$, in the category $\mathtt{C}^{\nu}$. They have complementary roles. The unary constant $\uparrow$ 'lifts' a noun to the corresponding set, $\downarrow$ does the opposite.

Examples:

($\mathtt{C}^{\sigma}$) (a natural number)$\uparrow = \mathbb{N}$, (a divisor of 6)$\uparrow = \mathtt{Set}_{n\in\mathbb{N}}(n \text{ is a divisor of } 6) = \{1,2,3,6\}$,[4] $(\mathtt{Noun}_{x\in\mathbb{R}}(x>5))\uparrow = \mathtt{Set}_{x\in\mathbb{R}}(x>5)$.

($\mathtt{C}^{\nu}$) $\mathbb{Z}\downarrow$ is 'an integer', $(\mathtt{Set}_{x\in\mathbb{R}^2}(|x|=1))\downarrow$ is $\mathtt{Noun}_{x\in\mathbb{R}^2}(|x|=1)$ or 'a point on the unit circle'.

2.5 Binders

As a third set given beforehand and infinite, we have the set of *binders*. This set is disjoint from both the set of variables and the set of constants. We divide the set $\mathtt{B}$ of binders into five subcategories, depending on the resulting category of the bound expression $\mathtt{B}_{\mathcal{Z}}(\mathcal{E})$ in which the binder occurs:

($\mathtt{B}^{\mathtt{t}}$) Binders forming *terms*,

($\mathtt{B}^{\sigma}$) Binders forming *sets*,

($\mathtt{B}^{\nu}$) Binders forming *nouns*,

($\mathtt{B}^{\alpha}$) Binders forming *adjectives*,

($\mathtt{B}^{\mathcal{S}}$) Binders forming *statements*.

Hence, $\mathtt{B} = \mathtt{B}^{\mathtt{t}} \mid \mathtt{B}^{\sigma} \mid \mathtt{B}^{\nu} \mid \mathtt{B}^{\alpha} \mid \mathtt{B}^{\mathcal{S}}$.

In the body $\mathcal{E}$ of bound expressions $\mathtt{B}_{\mathcal{Z}}(\mathcal{E})$ various linguistic categories can occur:

$\mathcal{E} = \mathtt{t} \mid \sigma \mid \nu \mid \mathcal{S}$,

depending on what $\mathtt{B}$ is. We recall that $\mathcal{Z}$ is a declaration, e.g. $x : \mathbb{N}$. (See also section 2.7.1.)

We give some examples of bound expressions, specifying the appropriate body category $\mathcal{E}$. The bound expressions are listed according to the category of the binder:

[4] Here again, we used sugaring. We write, as usual, $\{1,2,3,6\}$ for $\mathtt{Set}_{n\in\mathbb{N}}(n=1 \vee n=2 \vee n=3 \vee n=6)$. However, the notation with Set is the only 'official' WTT-format.

- $\mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E}) \;=\; \min_{\mathcal{Z}}(\mathtt{t}) \mid \sum_{\mathcal{Z}}(\mathtt{t}) \mid \lim_{\mathcal{Z}}(\mathtt{t}) \mid \lambda_{\mathcal{Z}}(\mathtt{t}) \mid \lambda_{\mathcal{Z}}(\sigma) \mid \iota_{\mathcal{Z}}(\mathcal{S}) \mid \ldots$
- $\mathtt{B}^{\sigma}_{\mathcal{Z}}(\mathcal{E}) \;=\; \mathtt{Set}_{\mathcal{Z}}(\mathcal{S}) \mid \bigcup_{\mathcal{Z}}(\sigma) \mid \iota_{\mathcal{Z}}(\mathcal{S}) \mid \ldots$
- $\mathtt{B}^{\nu}_{\mathcal{Z}}(\mathcal{E}) \;=\; \mathtt{Noun}_{\mathcal{Z}}(\mathcal{S}) \mid \mathtt{Abst}_{\mathcal{Z}}(\mathtt{t}) \mid \mathtt{Abst}_{\mathcal{Z}}(\sigma) \mid \mathtt{Abst}_{\mathcal{Z}}(\nu) \mid \ldots$
- $\mathtt{B}^{\alpha}_{\mathcal{Z}}(\mathcal{E}) \;=\; \mathtt{Adj}_{\mathcal{Z}}(\mathcal{S}) \mid \ldots$
- $\mathtt{B}^{\mathcal{S}}_{\mathcal{Z}}(\mathcal{E}) \;=\; \forall_{\mathcal{Z}}(\mathcal{S}) \mid \ldots$

In these lists there are several (more or less) new binding symbols: λ, ι, `Noun`, `Abst` and `Adj`. They are explained below. For the binder `Set`, see section 2.2.

2.5.1 The λ-binder

Church's λ introduces *function abstraction*. The format for an expression bound by the λ-binder is: $\lambda_{\mathcal{Z}}(\mathtt{t}/\sigma)$. Here $\lambda_{\mathcal{Z}}(\mathtt{t})$ is a term-valued function and $\lambda_{\mathcal{Z}}(\sigma)$ is a set-valued function.

Examples:

($\mathcal{E} \equiv \mathtt{t}$) The term $\lambda_{x\in\mathbb{R}}(x^2)$ denotes the squaring function on the reals.

($\mathcal{E} \equiv \sigma$) The term $\lambda_{n\in\mathbb{N}}\mathtt{Set}_{k\in\mathbb{N}}(k \leq n)$ sends a natural number n to the set $\{0, 1, \ldots, n\}$.

2.5.2 The ι-binder

Russell's ι is used for a *definite description*: '*the* such and such, such that ...'. The general format for an expression bound with the ι-binder is: $\iota_{\mathcal{Z}}(\mathcal{S})$.

The result of the binding of a sentence by means of ι can either be a term or a set (therefore we find $\iota_{\mathcal{Z}}(\mathcal{S})$ both in the $\mathtt{B}^{\mathtt{t}}$- and in the $\mathtt{B}^{\sigma}$-list).

Examples:
The term $\iota_{n\in\mathbb{N}}(2 < n < \pi)$ describes natural number 3.
The set $\iota_{U:\ \mathrm{SET}}\ (3 \in U \wedge |U| = 1)$ describes the singleton set $\{3\}$. (The declaration $U :\ \mathrm{SET}$ expresses that U *is a set*. See also section 2.7.1.)

2.5.3 The `Noun`-binder

Since nouns (to be precise: indefinite noun phrases) are first-class citizens in WTT, they are treated similarly as sets. Consequently, next to set comprehension, we allow *noun comprehension*, i.e. the construction of a noun describing a type in noun-format. For noun comprehension we introduce the binder `Noun`. It is used for an *indefinite description*: '*a* such and such, such that ...'. Hence, the general format of a phrase with `Noun`-binder is: $\mathtt{Noun}_{\mathcal{Z}}(\mathcal{S})$, i.e. 'a noun saying of $\mathcal{Z}$ that $\mathcal{S}$'.

Examples: The noun $\mathtt{Noun}_{x\in\mathbb{R}}(5 < x < 10)$ represents 'a real number between 5 and 10'. And $\mathtt{Noun}_{V:\ \mathrm{SET}}\ (|V| = 2)$ is 'a set with two elements'.

2.5.4 The Abst-binder

The Abst-binder *abstracts* from a term t, a set σ or a noun ν and delivers a noun. It is the formal counterpart of the modifier 'for some ...'.

Examples of the three kinds of nouns $\mathtt{Abst}_{\mathcal{Z}}(\mathtt{t}/\sigma/\nu)$:

($\mathcal{E} \equiv \mathtt{t}$) $\mathtt{Abst}_{n\in\mathbb{N}}(n^2)$ represents 'a term n^2 *for some* natural number n', i.e. 'the square of some natural number'.[5]

($\mathcal{E} \equiv \sigma$) $\mathtt{Abst}_{n\in\mathbb{N}}\mathtt{Set}_{x\in\mathbb{R}}(x > n)$ represents 'a set $\{x \in \mathbb{R} \mid x > n\}$ *for some* natural number n', i.e. 'an interval of the form (n, ∞)', with $n \in \mathbb{N}$.

($\mathcal{E} \equiv \nu$) $\mathtt{Abst}_{n\in\mathbb{N}}\mathtt{Noun}_{x\in\mathbb{R}}(10n \leq x < 10n+1)$ represents 'a real number in the interval $[10n, 10n+1)$ *for some* n', i.e. 'a non-negative real number which, written in decimal notation, has a zero at the position just before the decimal point'.

Notes.

(1) The Abst-binder is useful and compact. It enables one to put the abstraction quantification on the outside of the expression. However, a noun constructed with the Abst-binder can always be rewritten into one without it. We show this by rewriting the examples in a form without Abst, viz.:
($\mathcal{E} \equiv \mathtt{t}$) $\mathtt{Abst}_{n\in\mathbb{N}}(n^2) \rightsquigarrow \mathtt{Noun}_{k\in\mathbb{N}}\exists_{n\in\mathbb{N}}(k = n^2)$,
($\mathcal{E} \equiv \sigma$) $\mathtt{Abst}_{n\in\mathbb{N}}\mathtt{Set}_{x\in\mathbb{R}}(x > n) \rightsquigarrow \mathtt{Noun}_{V:\ \mathrm{SET}}\ \exists_{n\in\mathbb{N}}(V = \mathtt{Set}_{x\in\mathbb{R}}(x > n))$,
($\mathcal{E} \equiv \nu$) $\mathtt{Abst}_{n\in\mathbb{N}}\mathtt{Noun}_{x\in\mathbb{R}}(10n \leq x < 10n+1) \rightsquigarrow \mathtt{Noun}_{x\in\mathbb{R}}\exists_{n\in\mathbb{N}}(10n \leq x < 10n+1)$.

Note that in each of these examples, an 'inside' $\exists$ takes over the role of the 'outside' Abst.

(2) In the third case, the Abst-binder, transforming a noun into a noun, closely corresponds to the $\bigcup$-binder, transforming a set into a set. This can be expressed by the following abstract transformation:
$(\mathtt{Abst}_{\mathcal{Z}}(\nu))\uparrow = \bigcup_{\mathcal{Z}}((\nu)\uparrow)$.

See the third example: $\bigcup_{n\in\mathbb{N}} \mathtt{Set}_{x\in\mathbb{R}}(10n \leq x < 10n+1)$ is the *set* of all real numbers in *some* interval $[10n, 10n+1)$. This set can also be written in a form without $\bigcup$, viz.:
$\bigcup_{n\in\mathbb{N}} \mathtt{Set}_{x\in\mathbb{R}}(10n \leq x < 10n+1) \rightsquigarrow \mathtt{Set}_{x\in\mathbb{R}}\exists_{n\in\mathbb{N}}(10n \leq x < 10n+1)$.

Note again the 'inside' $\exists$, this time in the place of the 'outside' $\bigcup$.

2.5.5 The Adj-binder

Adjectives, being first-class citizens as well, can be constructed with the Adj-binder. One can read $\mathtt{Adj}_{\mathcal{Z}}(\mathcal{S})$ as: 'the adjective saying of $\mathcal{Z}$ that S'.

Example: $\mathtt{Adj}_{n\in\mathbb{N}}(\exists_{k\in\mathbb{N}}(n = k^2 + 1))$ is an adjective saying of a natural number that it is a square plus 1. One could give this adjective a name, say 'oversquare' and consequently say things like '5 is oversquare' or 'Let m be an oversquare number'.

[5] Note the difference between Abst and λ: the term $\lambda_{n\in\mathbb{N}}(n^2)$ is the *function* mapping n to n^2, whereas $\mathtt{Abst}_{n\in\mathbb{N}}(n^2)$ is an arbitrary element of the set $\{n^2 \mid n \in \mathbb{N}\}$. Note also that '$\mathtt{Noun}_{n\in\mathbb{N}}(n^2)$' is syntactically incorrect, since n^2 is not a statement, and also absurd: its natural reading 'an n in $\mathbb{N}$ such that n^2' makes no sense.

2.6 Phrases

Now that variables, constants and binders have been treated, we can give an abstract syntax for the various *phrase* categories. Phrases can be terms, sets, nouns or adjectives:

$$\begin{aligned}
\mathtt{t} &= \mathtt{C}^{\mathtt{t}}(\vec{\mathcal{P}}) \mid \mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\mathtt{t}} \\
\sigma &= \mathtt{C}^{\sigma}(\vec{\mathcal{P}}) \mid \mathtt{B}^{\sigma}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\sigma} \\
\nu &= \mathtt{C}^{\nu}(\vec{\mathcal{P}}) \mid \mathtt{B}^{\nu}_{\mathcal{Z}}(\mathcal{E}) \mid \alpha\nu \\
\alpha &= \mathtt{C}^{\alpha}(\vec{\mathcal{P}}) \mid \mathtt{B}^{\alpha}_{\mathcal{Z}}(\mathcal{E})
\end{aligned}$$

Examples of $\mathtt{C}^{\mathtt{t}}(\vec{\mathcal{P}})$, $\mathtt{C}^{\sigma}(\vec{\mathcal{P}})$, $\mathtt{C}^{\nu}(\vec{\mathcal{P}})$ and $\mathtt{C}^{\alpha}(\vec{\mathcal{P}})$ were given in section 2.4.1, examples of $\mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E})$, $\mathtt{B}^{\sigma}_{\mathcal{Z}}(\mathcal{E})$, $\mathtt{B}^{\nu}_{\mathcal{Z}}(\mathcal{E})$ and $\mathtt{B}^{\alpha}_{\mathcal{Z}}(\mathcal{E})$ in section 2.5.

The combination $\alpha\nu$ gives a (new) noun which is a combination of an adjective and a noun. Examples: 'isosceles triangle', 'convergent series'.

Note that variables ranging over nouns or adjectives are missing in this scheme, just as in section 2.3: such variables are not required in either CML or WTT.

2.7 Statements

Abstract syntax for the category of *statements* is:

$$\mathcal{S} = \mathtt{C}^{\mathcal{S}}(\vec{\mathcal{P}}) \mid \mathtt{B}^{\mathcal{S}}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\mathcal{S}}$$

Examples of $\mathtt{C}^{\mathcal{S}}(\vec{\mathcal{P}})$ were given in section 2.4.1. An example of $\mathtt{B}^{\mathcal{S}}_{\mathcal{Z}}(\mathcal{E})$ (with the $\forall$-binder for $\mathtt{B}^{\mathcal{S}}$) was given in section 2.5.

2.7.1 Typings and declarations

A *typing statement* or *typing*, expresses the relation between something and its type. In WTT we have five kinds of typings, depending on the nature of the type. This type can be: SET (the type of all sets), STAT (the type of all statements), a set, a noun or an adjective. Each of these statements relates a *subject* (the left hand side) with its *type*[6] (the right hand side). Abstract syntax for the set $\mathcal{T}$ of typing statements (a subcollection of $\mathcal{S}$) is:

$$\mathcal{T} = \sigma : \text{ SET } \mid \mathcal{S} : \text{ STAT } \mid \mathtt{t} : \sigma \mid \mathtt{t} : \nu \mid \mathtt{t} : \alpha$$

In words: 'σ is a set', '$\mathcal{S}$ is a statement', '$\mathtt{t}$ is an element of σ', '$\mathtt{t}$ is ν', '$\mathtt{t}$ is α'.

Examples of these five cases: '$\mathtt{Set}_{n\in\mathbb{N}}(n \leq 2)$: SET ', '$p \wedge q$: STAT ', '$3 \in \mathbb{N}$',[7] 'AB : an edge of $\triangle ABC$', '$\lambda_{x\in\mathbb{R}}(x^2)$: differentiable'.

Clearly, $\mathcal{T}$ is a subcollection of $\mathtt{C}^{\mathcal{S}}(\vec{\mathcal{P}})$, the set of relational statements, with symbol ':' as special element in $\mathtt{C}^{\mathcal{S}}$. See also section 3.7.

[6] Also called the *predicate*.

[7] As this example (again) shows, we often replace $\mathtt{t} : \sigma$ by $\mathtt{t} \in \sigma$, with abuse of notation.

In its turn, a subcollection of the typings is formed by the *declarations*, $\mathcal{Z}$, where the subject is a variable. A declaration is meant to *introduce a new variable* of a certain type.[8]

$\mathcal{Z} \;=\; \mathtt{V}^{\sigma} :\ \mathrm{SET} \;\mid\; \mathtt{V}^{\mathcal{S}} :\ \mathrm{STAT} \;\mid\; \mathtt{V}^{\mathtt{t}} : \sigma \mid \mathtt{V}^{\mathtt{t}} : \nu$

Here the variable $\mathtt{V}^{\sigma}$, $\mathtt{V}^{\mathcal{S}}$ or $\mathtt{V}^{\mathtt{t}}$ is the *introduced* or *declared* variable.
Subscripts of binders (see section 2.5) can be taken from $\mathcal{Z}$.

2.8 Definitions

An important category in WTT is the category $\mathcal{D}$ of *definitions*. Definitions introduce a new constant.

Remark 2.8.1 *We only regard non-recursive definitions. This is comparable to the situation in systems for Type Theory like the pioneering system Automath (see [De Bruijn 70]) and the most prominent system of today, Coq, which is based on the Calculus of Constructions (see [Coquand and Huet 88]). In both systems, recursion is not a basic operation and must be simulated if necessary. However, in more recent versions of Coq (e.g. [Coq 96]), inductive types have been incorporated.*

Similarly, WTT as it is now has no recursion operator or direct possibilities for recursive definitions. It is a subject for future research to extend WTT with recursion.

We distinguish between *phrase definitions* $\mathcal{D}^{\varphi}$ and *statement definitions* $\mathcal{D}^{\mathcal{S}}$:
$\mathcal{D} \;=\; \mathcal{D}^{\varphi} \mid \mathcal{D}^{\mathcal{S}}$.

Phrase definitions fix a constant representing a phrase (section 2.8.1). Statement definitions also introduce a constant, but embedded in a statement (section 2.8.2).

In definitions, the newly defined constant is separated from the phrase or statement it represents by the symbol ':='.

Remark 2.8.2 *We have decided not to include definitions for binders. The main reason is, that the set of binders used in mathematics is relatively stable, new binders are only seldom necessary. On the other hand, it is not very hard to include binder definitions in the syntax.*[9]

2.8.1 Phrase definitions

There are four kinds of phrase definitions:

$\mathcal{D}^{\varphi} \;=\; \mathtt{C}^{\mathtt{t}}(\overrightarrow{\mathtt{V}}) := \mathtt{t} \mid \mathtt{C}^{\sigma}(\overrightarrow{\mathtt{V}}) := \sigma \mid \mathtt{C}^{\nu}(\overrightarrow{\mathtt{V}}) := \nu \mid \mathtt{C}^{\alpha}(\overrightarrow{\mathtt{V}}) := \alpha$

The newly *defined constants* in these definitions are $\mathtt{C}^{\mathtt{t}}$, $\mathtt{C}^{\sigma}$, $\mathtt{C}^{\nu}$ or $\mathtt{C}^{\alpha}$, respectively.

[8] There are no declarations with an adjective as type. This seems strange, at first sight, since it is usual to write things like: 'Let f be differentiable'. It is our opinion, however, that such a sentence is used in either one of the following two cases: (1) as an elliptic version of the introduction of a new variable with a *noun* (not an adjective) as type: 'Let f be *a differentiable function*', or (2) as an *assumption* about an f which is already known (not new), and hence as a typing statement, *not* a declaration.

[9] Abstract syntax for binder definitions could be:
$\mathcal{D}^{\mathtt{B}} \;=\; \mathtt{B}^{\mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}}_{\mathcal{Z}}(\mathcal{E}) := \mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}.$

Note that the parameters occurring after the $\mathtt{C}$ in the left hand side of each definition must be *variables*. The reason is of course, that a definition should be as general as possible and hence may 'depend' on a list of variables. Later, when *using* the definition in a certain situation, all these variables must be 'instantiated' according to that situation.

Examples of the four kinds of phrase definitions are:

$(\mathtt{C} \equiv \mathtt{C}^{\mathtt{t}})$ *the arithmetic mean* of a and $b := \frac{1}{2}(a+b)$,

$(\mathtt{C} \equiv \mathtt{C}^{\sigma})$ $\mathbb{R}^+ := \mathtt{Set}_{x\in\mathbb{R}}(x > 0)$,

$(\mathtt{C} \equiv \mathtt{C}^{\nu})$ *a unit* of G with respect to $\cdot := \mathtt{Noun}_{e\in G}(\forall_{a\in G}(a\cdot e = e\cdot a = a))$[10],

$(\mathtt{C} \equiv \mathtt{C}^{\alpha})$ *prime* $:= \mathtt{Adj}_{n\in\mathbb{N}}(n > 1 \wedge \forall_{k,l\in\mathbb{N}}(n = k\cdot l \Rightarrow k = 1 \vee l = 1))$.[11]

The variable lists in the four examples are: (a,b), $(\ )$, $(G,\cdot)$, $(\ )$. As we will see in chapter 3, these variables must be introduced ('declared') in a context (see also section 2.9). For the first definition, such a context can be e.g. $a:\mathbb{R}, b:\mathbb{R}$. For the third definition the context is: $G:$ SET , $\cdot : G \to G$. Both contexts consist of *declarations* only.

However, definitions may also depend on *assumptions*. This is reflected in section 2.9, where it is stated that a context consists of a list of declarations *and* assumptions. For an example, take the definition of the natural logarithm (this is again case $\mathtt{C} \equiv \mathtt{C}^{\mathtt{t}}$):

$\ln(x) := \iota_{y\in\mathbb{R}}(e^y = x)$.

Here variable x has to be declared in a context, for example: $x:\mathbb{R}, x > 0$. This is a declaration: $x:\mathbb{R}$, introducing x of type $\mathbb{R}$, followed by an assumption: $x > 0$, stating that the introduced x is positive.

In general, definitions are not complete without such a context. That is to say, the 'ground has to be prepared' before the actual definition is stated.

In the following chapter 3 it turns out that in 'weakly well-typed' definitions, the variables in variable list $\vec{\mathtt{V}}$ are the same as the declared variables in the context, and listed in the same order. (The *assumptions* occurring in the context are not accounted for in the parameter list of a WTT-constant.)[12] E.g., in the first example, the parameter list of 'the arithmetic mean' is '(a,b)', which is the exactly the same as the list of the declared variables occurring in the context '$a\in\mathbb{R}, b\in\mathbb{R}$'.[13]

An *instantiation* of a defined notion is the *use* of a defined constant, thereby replacing the variables occurring in the variable list, by actual terms or sets.

Examples of instantiations of the first example definition are: 'the arithmetic mean of 3 and 6', or, for given x: 'the arithmetic mean of x and x^2'.

[10] Of course, $a\cdot e = e\cdot a = a$ is a sugared version of e.g. $a\cdot e = a \wedge e\cdot a = a$.

[11] Here obviously $\forall_{k,l\in\mathbb{N}}\ldots$ acts as a syntactic sugaring of $\forall_{k\in\mathbb{N}}\forall_{l\in\mathbb{N}}\ldots$.

[12] In Type Theory, however, variables 'inhabiting' assumptions are added to the variable list.

[13] Since such a parameter list can be reconstructed from the context in which the definition is embedded, these parameter lists (or parts of it) are often omitted.

2.8.2 Statement definitions

We introduce the following category of statement definitions:

$\mathcal{D}^{\mathcal{S}} \quad = \quad \mathtt{C}^{\mathcal{S}}(\overrightarrow{\mathtt{V}}) \ := \ \mathcal{S}$

The newly *defined constant* in this statement definition is $\mathtt{C}^{\mathcal{S}}$.

Example:
$(\mathtt{C} \equiv \mathtt{C}^{\mathcal{S}})$ *a is parallel to b* $:= \ \neg\exists_{P\text{: a point}}(P$ lies on $a \wedge P$ lies on $b)$.

Again, we need a context to make the definition self-contained. This context is, for example:
a : a line, b : a line
(i.e., 'Let a and b be lines').

Note that the notion 'is parallel to' can only be considered as a two-place *relation*, and therefore its definition must be a statement definition. In other cases, things are not so clear and the WTT-user has to make a choice.

For example, the definition of 'x is the opposite of y' can be treated as a statement definition, similarly to the one above:

(1) *x is the opposite of y* $:= \ x + y = 0$,

with context consisting of e.g. $x : \mathbb{R}, y : \mathbb{R}$. Here we have the case $\mathtt{C} \equiv \mathtt{C}^{\mathcal{S}}$.

But it is also possible to define the same notion in a *phrase* definition (to be precise: a term definition), as follows:

(2) *the opposite of y* $:= \iota_{x\in\mathbb{R}}(x = -y)$.

Now the context is only $y : \mathbb{R}$ and we have the case $\mathtt{C} \equiv \mathtt{C}^{\mathtt{t}}$.

The difference is whether one considers '(is) the opposite of' to be a *relation* or a *function*. The latter choice allows more freedom, since a phrase definition can always be used as part of a sentence, but not the other way round. For example, if one chooses for (2), then it is possible to instantiate this definition in a phrase:

'the opposite of 5',

but also in a sentence:

'-5 = the opposite of 5'.

Likewise, it is more flexible to define the phrase (in this case: the noun) 'a unit of $(G, \cdot)$' than to define the statement 'e is a unit of $(G, \cdot)$'.

2.9 Contexts

A *context* Γ is a list of declarations (from $\mathcal{Z}$) and statements (from $\mathcal{S}$):

$\boldsymbol{\Gamma} \quad = \quad \emptyset \mid \boldsymbol{\Gamma}, \mathcal{Z} \mid \boldsymbol{\Gamma}, \mathcal{S}$

A declaration Z occurring in a context represents the *introduction of a variable* of a certain (already known) type. A statement S in a context stands for an *assumption*.[14]

We repeat two examples of contexts which we gave before:

$x : \mathbb{R}, x > 0$,

a : a line, b : a line .

For more examples we refer to chapter 4.

2.10 Lines

A *line* $\mathbf{l}$ contains either a statement or a definition, relative to a context:

$\mathbf{l} \;=\; \mathbf{\Gamma} \triangleright \mathcal{S} \mid \mathbf{\Gamma} \triangleright \mathcal{D}$

The symbol $\triangleright$ is used as separation marker between the context and the statement or definition (in that context).

Examples of lines:

$x : \mathbb{N}, y : \mathbb{N}, x < y \;\triangleright\; x^2 < y^2$,

$x : \mathbb{R}, x > 0 \;\triangleright\; ln(x) := \iota_{y \in \mathbb{R}}(e^y = x)$.

The first line above contains a statement (it is of the form $\mathbf{\Gamma} \triangleright \mathcal{S}$), the second contains a definition (and is of the form $\mathbf{\Gamma} \triangleright \mathcal{D}$). Again, more examples can be found in chapter 4.

2.11 Books

A *book* $\mathbf{B}$ is a list of lines:

$\mathbf{B} \;=\; \emptyset \mid \mathbf{B} \circ \mathbf{l}$

A simple example of a book consisting of two lines is the following:

$x : \mathbb{R}, x > 0 \;\triangleright\; ln(x) := \iota_{y \in \mathbb{R}}(e^y = x)$
$\emptyset \;\triangleright\; ln(e^3) = 3$.

In chapter 4 we give several examples of (short) WTT-books.

[14] According to our syntax, such an assumption – being a statement – can also be a declaration – being a statement with *a variable* as subject. However, our typing rules for contexts (see chapter 3) will ensure that there is no *newly* introduced variable in an assumption, so it is always clear whether a context statement represents the introduction of a new variable, or an assumption.

Chapter 3

A derivation system for WTT

In this chapter we develop a derivation system for WTT. A WTT-book constructed with this derivation system obeys the syntax given in the previous chapter, hence such a book is well-formed (cf. 2.1). However, the derivation rules only give a *subset* of the well-formed constructs obtained with the abstract syntax of chapter 2, since the rules enforce that those constructs obey certain (weak) *typing requirements*. Constructs obtained by repeated application of the derivation rules, we therefore call *weakly well-typed*. The overall properties of weakly well-typed constructs can be summarized as follows:

- All constructs obtained with the derivation system have a weak type, which corresponds to a linguistic category.
- The derivation system is *syntax-driven* in the sense that for each (sub-)goal in a derivation, only one rule is applicable.

On a smaller scale, the derivation system has the following properties:

- There are no free variables in a book which has been constructed with the derivation system.
- Each occurrence of a variable and each occurring constant has a fixed weak type.
- Each occurring binder has a fixed weak *input*-type and a fixed weak *output*-type.
- Each occurrence of a (sub-)formula in a weakly well-typed book has a fixed weak type.

A book which has been constructed with the derivation system of this chapter, is transparently structured. Yet, it has a great resemblance to an ordinary mathematical text: see the examples in chapter 4. Therefore, a weakly well-typed book can be seen as a natural formalization of mathematics, highlighting a number of characteristic features of mathematical texts. Among these features are:

- A WTT-book reflects the line-for-line development of the original mathematical text, in the order of the lines which form the WTT-book.

- The important role of contexts, both for (mathematical) statements and for (mathematical) definitions, is made explicit in a WTT-book .
- The translation also preserves the modular construction of a mathematical sentence (either a statement or a definition), with constituents of different linguistic character: terms, sets, nouns, adjectives, statements.
- There is an important role for defined constants with parameter lists, which can be instantiated.
- Binders of different sorts are incorporated.

One can also consider a WTT-text to be a first step towards a complete formalization into Type Theory (see chapter 5). It is said that Type Theory is the natural formalism for such a formalization and there are many examples to support this claim: think e.g. of Automath, 'a language for mathematics' ([Nederpelt, Geuvers, de Vrijer 94]), being defined in the late sixties and applied to express numerous mathematical subject matters. Other examples are the well-known theorem provers Coq ([Coq 96]) and LEGO ([Luo and Pollack 92]), developed in the nineties and now in a stage of being widely applicable. These systems, and several others, are firmly embedded into Type Theory.

However, it should be noted that the type restrictions of WTT are only weak. Consequently, a successful formalization of a mathematical text into WTT does not at all guarantee that its mathematical content is in any sense 'meaningful'. A prerequisite for the construction of a sensible WTT-text is that the person writing this text has a mathematical subject in mind, which he chooses to express in this formalism, as faithfully as possible. But only a further translation into Type Theory will give a complete picture, which still has to be 'authorized' by the original text writer: "This is (or is not) what I had in mind".

3.1 Weak types

We now discuss the derivation system in general. The main feature is that the derivation rules enable one to extend a weakly well-typed book B with a line l, in order to form a new weakly well-typed book $B \circ l$. This is the case if the added line l obeys certain weak well-typedness requirements itself, relative to the book B. Since a line l always has the form $\Gamma \triangleright S$ or $\Gamma \triangleright D$, with statement S or definition D (see section 2.10), we also have to consider weak well-typedness of a context Γ *relative to a book* B, and weak well-typedness of a statement S or a definition D *relative to a book* B *and a context* Γ.

For establishing the different forms of weak well-typedness, we need a notion of *weak typing* between an entity and its weak type. This relation is denoted by a bold-faced colon (**:**). As weak types, denoted by **W**, we use the following eight entities, corresponding to a subset of our linguistic categories given in section 2.1:

W = `book`, `cont`, `term`, `set`, `noun`, `adj`, `stat`, `def`.

They stand for 'books', 'contexts', 'terms', 'sets', 'nouns', 'adjectives', 'statements' and 'definitions'.

Hence, for example, B `:` `book` expresses that the weak type of B is `book` (or: B is a weakly well-typed book) and n `:` `noun` expresses that the weak type of n is `noun`.

We also introduce a notion of *(relative) derivability*. We use the bold-faced symbol $\vdash$ for this notion. We distinguish between three formats for derivability, in the form of so-called *judgements*:

(1) The judgement that B is a weakly well-typed book:

$\vdash B$ `:` `book`.

(2) The judgement that Γ is a weakly well-typed context relative to book B:

$B \vdash \Gamma$ `:` `cont`.

(3) The judgements that t is a weakly well-typed term, etc., relative to book B and context Γ:

$B;\Gamma \vdash t$ `:` `term`,

$B;\Gamma \vdash s$ `:` `set`,

$B;\Gamma \vdash n$ `:` `noun`,

$B;\Gamma \vdash a$ `:` `adj`,

$B;\Gamma \vdash S$ `:` `stat`,

$B;\Gamma \vdash D$ `:` `def`.

In the sections below, we first discuss the *preface* of a book B, meant to establish weak type information about all constants occurring in B but not explicitly defined in B.

In the presentation of our derivation rules we follow the construction of the various weak types bottom up. In most of the rules we assume that we already have a weakly well-typed book B and a weakly well-typed context Γ relative to B. We start with variables and constants relative to B and Γ. Next, we discuss the construction of phrases beginning with a binder and of phrases in general. Thereafter, we give derivation rules for sentences (statements and definitions).

Finally, we give rules for contexts (relative to a book) and for books themselves.

We end the chapter with a list of relevant properties of the derivation system.

A list of all derivation rules given in this chapter, can be found in Appendix B.

Notation 3.1.1 *As we said, we often start from the assumption that B is a weakly well-typed book and Γ a weakly well-typed context relative to B. For convenience, we abbreviate the corresponding pair of premisses: $\vdash B$ `:` `book`, $B \vdash \Gamma$ `:` `cont`, by: $OK(B;\Gamma)$.*

3.2 The preface

As we saw before, a book B is a list of lines $\Gamma \triangleright S$ and/or $\Gamma \triangleright D$, where each S in such a line is a statement and each D a definition.

Definition 3.2.1 *We say that $l \in B$ if l is one of the lines constituting B.*

In the case that line l is a definition, it contains exactly one *defined constant* (see section 2.8.1 and 2.8.2).

Definition 3.2.2 *Let $l \in B$ and suppose that l is a definition line $\Gamma \triangleright D$ where D is of the form $c(x_1, \ldots, x_n) := \Phi$. Then the defined constant of the definition line, or* $\texttt{defcons}(D)$*, is c.*

Moreover, $\texttt{defcons}(B) = \{\texttt{defcons}(D) \mid \Gamma' \triangleright D \text{ is a line of } B, \text{ for some } \Gamma'\}$*. We call these constants the internally defined or internal constants of B.*

All parameterized constants occurring in a book B outside a definition, represent defined notions, with instantiated parameter lists. Such a constant may be internally defined, but this is not necessarily the case: B usually is a text-*fragment*, being part of a larger text, the rest of which is omitted. Consequently, a parameterized constant occurring in a book B can 'stand on its own', i.e., needs not have a corresponding definition being part of a line *inside* B. Such constants are called *externally defined* or *external* constants (relative to B).

Example 3.2.3 *– In many books the constants $\mathbb{N}$ and $\mathbb{R}$ will be used without proper definition. (Both $\mathbb{N}$ and $\mathbb{R}$ are parameterized constants with empty parameter list.)*

– One also uses many other well-known constants without recalling their definition, such as $\sqrt{}$. (This constant needs a parameter list of length one, e.g., $\sqrt{5}$). See again the examples in chapter 4.

*– Another well-known constant is '*ln*', which will be an external constant for many WTT-books. It has one declared variable in its context. The corresponding definition line, which will probably not occur in the book B under consideration, is for example,*

$$x \in \mathbb{R},\ x > 0 \quad \triangleright \quad \ln(x) \ := \ \iota_{y \in \mathbb{R}}(e^y = x),$$

with declaration $x \in \mathbb{R}$, so x is the declared variable[1]*. Note that a prerequisite for this definition of '*ln*' is, that 'e' has already been defined beforehand.*

*When using (instantiating) the constant '*ln*', we need a parameter list of length one, e.g.,* $\ln(5)$*. Note that* 5 *matches with x since it belongs to the same linguistic collection* `t` *(of terms).*

In order to be able to judge weak well-typedness of the *externally* defined constants, we extend a book B on the front side with a *preface*, consisting of a list of constants, together with the weak types of its parameters and the resulting weak type of the constant. In fact, it is not essential for the derivation system how the external constant is exactly defined, we can suffice with a description of the weak types connected with such a constant, in particular: which weak *input*-types are expected for the parameters of the constant, and what is the weak *output*-type of the constant.

Definition 3.2.4 *We denote a preface for a book B by* $\texttt{pref}(B)$*. All constants mentioned in this preface, are collected in the set* $\texttt{prefcons}(B)$*.*

If $c \in \texttt{prefcons}(B)$, then $in(c)$ is the list of the weak types of the parameters of c, and $out(c)$ is the resulting weak type of the full construct $c(\ldots)$.

[1] $x > 0$ is an assumption, i.e. a statement which is not a declaration, see section 2.8.1.

Example 3.2.5 *A preface for a book B could look like:*

constant name	*weak types in*	*weak type out*
$\mathbb{R}$	( )	`set`
$\surd$	(`term`)	`term`
$+$	(`term`, `term`)	`term`
$\cup$	(`set`, `set`)	`set`
$\rightarrow$	(`set`, `set`)	`set`
$\geq$	(`term`, `term`)	`stat`
$\wedge$	(`stat`, `stat`)	`stat`

So, for example:

– $\mathbb{R}$ has no parameters and is a set. So $in(\mathbb{R}) = (\ )$ and $out(\mathbb{R}) =$ `set`.

– $\surd$ is a constant with one parameter, a term, and delivers itself, again, a term. Hence, $in(\surd) =$ (`term`) and $out(\surd) =$ `term`.

– $\geq$ is a constant with two parameters, both being terms, delivering a statement.

– In the case of the preface above for book B, $\texttt{prefcons}(B) = \{\mathbb{R}, \surd, +, \cup, \rightarrow, \geq, \wedge\}$.

We assume that a preface for a book B, describing the weak (in- and out-)types for all the externally defined constants of B, is constructed by the book-writer himself, on the basis of his or her personal knowledge of mathematics. In the following sections, we implicitly suppose that each book B is extended with an appropriate preface $\texttt{pref}(B)$.

Remark 3.2.6 *When translating a given mathematical text into WTT, the translator has to construct the preface himself, by collecting all external constants occurring in that text and determining the appropriate weak in- and out-types. This is necessary for establishing the well-typedness of the translation of the given text.*

However, apart from the construction of this preface, it may be very useful to adorn the WTT-text with other information present in the original mathematics text, namely about the structure of the text or about its coherence. The latter information is lost in the WTT-translation, but is very welcome when one decides to translate the obtained WTT-book, in its turn, into Type Theory.

For instance, the original text possibly has interesting labels *attached to paragraphs or to subtexts, such as 'Theorem 5.2' or 'Proof'. Moreover, the original text may have many intermediate statements about interdependencies in the text, which are very useful when translating further into Type Theory. Think of* comments *such as 'Use Theorem 5.2', 'by the definition of c' or 'using formula 2.1'. (Recall that WTT does not take into account whether a text is logically or mathematically* valid*, whereas validity is an essential requirement for a possible formalization in Type Theory.)*

Therefore, it may be good practice to add, apart from the preface, two columns on either side of a WTT-book B, one with useful labels labeling lines in B, and another with comments going with specific lines. This makes it much easier for the translator of a WTT-book into Type Theory who does not have access to the original text.

See the example in section 4.4 for a practical situation.

3.3 Variables

The list `decvar` of *declared variables* of a context Γ is defined as follows:

Definition 3.3.1 *(1) If* $\Gamma = \emptyset$, *then* $\texttt{decvar}(\Gamma) = \emptyset$.

(2a) If $\Gamma = \Gamma', x : A$ *and* $x \notin \texttt{decvar}(\Gamma')$, *then* $\texttt{decvar}(\Gamma) = \texttt{decvar}(\Gamma'), x$.

(2b) Otherwise, if $\Gamma = \Gamma', S$, *then* $\texttt{decvar}(\Gamma) = \texttt{decvar}(\Gamma')$.

That is to say, $\texttt{decvar}(\Gamma)$ collects all subject variables of *declarations* in Γ, in the order in which they appear in Γ,[2] but $\texttt{decvar}(\Gamma)$ ignores the possible subject variables occurring in *assumptions* in Γ. (See also sections 2.8 and 2.9.)

Now the derivation rule for *variables* is[3]:

$$\frac{OK(B,\Gamma), \quad x \in \mathtt{V}^{\mathtt{t}/\sigma}, \quad x \in \texttt{decvar}(\Gamma)}{B;\Gamma \vdash x \texttt{ : term/set}} \quad (var)$$

Notation 3.3.2 *Here we combine two cases, distinguished by the slash '/'. In the first case,* $x \in \mathtt{V}^{\mathtt{t}}$, *the conclusion is:* $B;\Gamma \vdash x$ `: term`. *In the second case,* $x \in \mathtt{V}^{\sigma}$, *we get:* $B;\Gamma \vdash x$ `: set`. *We also use this abbreviation convention below.*

Hence, each declared variable in Γ has weak type `term` or `set`, relative to B and Γ.

3.4 Constants

We have the following derivation rules for constants. The first rule concerns internal constants, defined in the book, the second rule concerns external constants for which the weak in- and out-types are given in the preface of the book.

Constants defined internally in B are divided into *phrase definitions* for terms, sets, nouns and adjectives and *statement definitions*. See section 2.8.1 and 2.8.2.

The derivation rule for *internal* constants is:

$$\frac{\begin{array}{c} OK(B;\Gamma), \quad \Gamma' \triangleright D \ \in \ B, \\ \texttt{decvar}(\Gamma') = x_1, \ldots, x_n, \quad \texttt{defcons}(D) = c \in \mathtt{C}^{\mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}}, \\ B;\Gamma \vdash A_i \texttt{ : } \mathbf{W} \text{ if } B;\Gamma' \vdash x_i \texttt{ : } \mathbf{W} \ (i = 1, \ldots, n) \end{array}}{B;\Gamma \ \vdash \ c(A_1, \ldots, A_n) \texttt{ : term/set/noun/adj/stat}} \quad (int{-}cons)$$

Note that the list of declared variables of Γ', viz. $x_1, \ldots, x_n$, is the same as the variable list following the defined constant c in the definition line $\Gamma' \triangleright D$ (this follows from the rule $(int{-}def)$, see section 3.8). Otherwise said: $D \ \equiv \ c(x_1, \ldots, x_n) := \ldots$. Hence, the above derivation rule determines how such a $c(x_1, \ldots, x_n)$ can be instantiated, with result

[2] The *order* in the `decvar` list of a context Γ is reflected in the order of the variables in the variable list going with any constant defined with respect to Γ. See section 3.4.

[3] The abbreviation $OK(B,\Gamma)$ is defined in Notation 3.1.1.

$c(A_1, \ldots, A_n)$. The rule above expresses that an instantiation is only allowed if each variable x_i becomes replaced by a formula A_i of the same weak type $\mathbf{W}$ as x_i (see also example 3.2.3).

If c is an *external* constant of B, then it has weak in- and out-types as given in the preface of B. For such constants we have the following derivation rule:

$$\frac{OK(B,\Gamma), \quad c \text{ external to } B, \quad in(c) = (\kappa_1, \ldots, \kappa_n), \quad out(c) = \kappa, \quad B;\Gamma \ \vdash \ A_i : \kappa_i \quad (i = 1, \ldots, n)}{B;\Gamma \ \vdash \ c(A_1, \ldots, A_n) : \kappa} \quad (ext{-}cons)$$

A special kind of external constants is the pair $\uparrow$ and $\downarrow$ (see section 2.4.2). They also have weak in- and out-types, given in the following list:

constant name	weak type *in*	weak type *out*
$\uparrow$	(noun)	set
$\downarrow$	(set)	noun

We assume that the last mentioned list is always part of the preface of a book B. Hence, expressions with $\uparrow$ or $\downarrow$ can also be derived with the rule $(ext{-}cons)$ above.

Remark 3.4.1 *For a more efficient implementation of the derivation system described in this chapter, one may treat* internally *defined constant just as the* externally *defined ones. This can be done by listing the internally defined constants in a preface-like form, giving only their names and the calculated weak in-types and out-types. This is sufficient information for establishing weakly well-typedness for expressions containing constants. Otherwise said:* after *the line in which a constant c has been defined, the only information needed about c concerns its weak in- and out-types.*

In case one chooses the above treatment of internally defined constants, the $(int{-}cons)$ *rule can be dismissed and the* $(ext{-}cons)$ *rule can be renamed to* $(cons)$*, with the provision 'c external to B' removed from the premisse.* [4]

3.5 Binders

As explained already in sections 2.5 and 2.6, phrases can begin with a *binder*, as in $\iota_{\mathcal{Z}}(\mathcal{S})$.

In the previous section we discussed how to register weak in- and out-types for external constants in a preface.

Binders, as well, have fixed weak in- and out-types. For example, binder ι (see section 2.5.2) takes a statement and delivers a term or a set (the subscript $\mathcal{Z}$ is not important, in this respect). Below we give a list describing the weak types corresponding to all binding symbols used in section 2.5:

[4] This remark is due to Roel Körvers.

binder name	weak type *in*	weak type *out*
min	(term)	term
$\sum$	(term)	term
lim	(term)	term
λ	(term)	term
ι	(stat)	term/set/stat
Set	(stat)	set
$\bigcup$	(set)	set
Noun	(stat)	noun
Abst	(term/set/noun)	noun
Adj	(stat)	adj
$\forall$	(stat)	stat

Definition 3.5.1 *For binder b, we denote the in-type by in(b) and the out-type by out(b).*

For expressions constructed by means of a binder b we also have a derivation rule:

$$\frac{OK(B;\Gamma,Z),\quad b\in \mathtt{B},\quad in(b)=(\kappa_1),\quad out(b)=\kappa_2,\quad B;\Gamma,Z\ \vdash\ A:\kappa_1}{B;\Gamma\ \vdash\ b_Z(A):\kappa_2}\quad (bind)$$

Note the role of the subscript Z: since A may depend on the subject variable of the declaration Z, we require that $B;\Gamma,Z\vdash A:\kappa_1$, i.e., '$A$ is correct with respect to book B and context Γ *extended with* declaration Z'. (This shift of Z from the context to the subscript is also present in the formation rule and the abstraction rule of Type Theory, with which $\Pi_Z(A)$ and $\lambda_Z(A)$ are formed and typed.)

3.6 Phrases

Now we have derivation rules for all constructs mentioned in the abstract syntax of section 2.6: variables, constants with parameter lists and expressions with binders. We only miss the special combination $\alpha\nu$ of an adjective and a noun. The derivation rule concerning this linguistic construction is the following:

$$\frac{OK(B;\Gamma),\quad B;\Gamma\ \vdash\ n:\mathtt{noun},\quad B;\Gamma\ \vdash\ a:\mathtt{adj}}{B;\Gamma\ \vdash\ an:\mathtt{noun}}\quad (adj{-}noun)$$

Hence, our weak typing system allows an adjective to be combined with a noun.

3.7 Statements

The derivation rules given above also suffice for the constructs given in the abstract syntax for statements (see section 2.7). This includes constants $\mathtt{C}^{\mathcal{S}}$ for statements, as well as logical quantifiers covered by the linguistic construct $\mathtt{B}^{\mathcal{S}}_{\mathcal{Z}}(\mathcal{E})$.

For *typings*, there are two kinds of derivations, dependent on the 'level' of the statement: the first kind is for a statement of the form s : SET or S : STAT , saying that s is a set or

that S is a statement, the second is for statements of the form $t : s/n/a$, expressing that term t has type set s, noun n or adjective a.

Both kinds can be treated with the rule $(ext{-}cons)$ given above, provided that we take the following list in our standard preface:

constant name	weak types *in*	weak type *out*
: SET	(set)	stat
: STAT	(stat)	stat
:	(term, set/noun/adj)	stat

Note that for the first kind of typing, for example s : SET , we consider ': SET ' to be a *unary* constant, taking as weak in-type (set) and having weak out-type stat. This unary behaviour is necessary since SET is a constant not covered by our linguistic categorization.

For the second kind of statement we consider ':' to be a *binary* constant, as expected.

3.8 Definitions

For internal definitions, we have the following derivation rule:

$$\frac{\begin{array}{c} OK(B;\Gamma), \quad B;\Gamma \ \vdash \ t/s/n/a/S : \mathtt{term/set/noun/adj/stat}, \\ \mathtt{decvar}(\Gamma) = x_1, \ldots, x_n, \quad c \in \mathcal{C}^{\mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}}, \quad c \notin \mathtt{prefcons}(B) \cup \mathtt{defcons}(B) \end{array}}{B;\Gamma \ \vdash \ c(x_1, \ldots, x_n) := t/s/n/a/S : \mathtt{def}} \ (int\text{-}def)$$

Remark 3.8.1 *Sometimes the first part of the parameter list of a defined constant is omitted, since it can be reconstructed by listing the declared variables (see section 3.3) of the context. This, however, is a form of sugaring, and not according to the rules.*

3.9 Contexts

The rules for the weak type cont, for contexts, start with the rule for the empty context:

$$\frac{B : \mathtt{book}}{B \ \vdash \ \emptyset : \mathtt{cont}} \ (emp\text{-}cont)$$

Next, we have two rules for adding a *declaration* to the context:

$$\frac{OK(B;\Gamma),\ x \in \mathtt{V}^{\sigma/\mathcal{S}},\ x \notin \mathtt{decvar}(\Gamma)}{B \ \vdash \ \Gamma, x : \text{ SET / STAT } : \mathtt{cont}} \ (set/stat\text{-}decl)$$

$$\frac{OK(B;\Gamma), \quad B;\Gamma \ \vdash \ s/n : \mathtt{set/noun}, \quad x \in \mathtt{V}^{\mathtt{t}}, \quad x \notin \mathtt{decvar}(\Gamma)}{B \ \vdash \ \Gamma, x : s/n \ : \mathtt{cont}} \ (term\text{-}decl)$$

Finally, there is a derivation rule for the addition of an *assumption*. Note that the statement S below, as we said before, can be a typing $x : A$. The x, however, cannot be 'new' with respect to Γ, since $B;\Gamma \vdash S$: stat. (This is a consequence of Lemma 3.11.1, (1).) Hence, S cannot be a *declaration*.

$$\frac{OK(B;\Gamma), \quad B;\Gamma \ \vdash \ S : \mathtt{stat}}{B \ \vdash \ \Gamma, S : \mathtt{cont}} \ (assump)$$

3.10 Books

Books are lists of lines, containing either a definition or a statement in a context. As a start, we have a derivation rule stating that the empty book is derivable:

$$\frac{}{\vdash\ \emptyset : \texttt{book}} \quad (emp{-}book)$$

Finally, we only need a rule saying that every weakly well-typed line with respect to a book B, either containing a statement or a definition, may lead to a weakly well-typed extension of B:

$$\frac{OK(B;\Gamma),\quad B;\Gamma\ \vdash\ S/D : \texttt{stat/def}}{\vdash B\ \circ\ \Gamma \triangleright S/D : \texttt{book}} \quad (book{-}ext)$$

3.11 Properties of the derivation system

The following general properties hold for the derivation system as described above. (Proofs can be founds in [Kamareddine and Nederpelt 2001].)

Lemma 3.11.1 *(1) There are no free variables in a line and, hence, in a book. To be precise: if* $B;\Gamma \vdash S : \texttt{stat}$ *or* $B;\Gamma \vdash D : \texttt{def}$*, then all free variables in* S *or* D *are declared variables of* Γ*.*

(2) All constants occurring in a book are either internally defined or introduced in the preface. For all occurrences of internal constants which are not 'defining occurrences',[5] *there is a defining line* preceding *this occurrence in which this constant is the defined constant.*

This lemma is a consequence of lemma 3.11.3(2) and 3.11.4(2), see below.

Remark 3.11.2 *The second part of this lemma implies that an internally defined constant has no other occurrence in its defining line.*

Note that one may define a notion of binding for constants, where the place of binding is either a defining occurrence or the occurrence in the preface. With this notion of binding, we can express the second part of this theorem as: There are no free constants in a book with appropriate preface.

Next, we reformulate a number of properties which are important in Type Theory (see [Barendregt 92]), for our WTT-case. We write $\Phi : \mathbf{W}$ for: formula Φ has weak type $\mathbf{W}$. As usual, $FV(\Phi)$ stands for the set of free variables in Φ.

Lemma 3.11.3 *(Free Variables lemma)*

(1) If $B \vdash \Gamma : \texttt{cont}$*, then the declared variables in* Γ *are distinct.*

(2) If $B;\Gamma \vdash \Phi : \mathbf{W}$ *then* $FV(\Phi) \subseteq \texttt{decvar}(\Gamma)$*.*

[5] A 'defining occurrence' of a constant c is an introduction of c in a definition line $\Gamma \triangleright c(x_1,\ldots,x_n) := \ldots$ occurring in the book B.

(3) If $B \vdash \Gamma$ **:** $\mathtt{cont}$ *where* $\Gamma \equiv \Gamma', x : A, \Gamma''$ *or* $\Gamma \equiv \Gamma', A, \Gamma''$*, then* $FV(A) \subseteq \mathtt{decvar}(\Gamma')$.

Lemma 3.11.4 *With the notion of binding for constants as explained in the previous remark, we can also add a Free Constants lemma (here FC means the set of 'free constants'):*

(1) If $\vdash B$ **:** $\mathtt{book}$*, then the defined constants in* B *are distinct.*

(2) If $B; \Gamma \vdash \Phi$ **:** $\mathbf{W}$ *then* $FC(\Phi) \subseteq \mathtt{prefcons}(B) \cup \mathtt{defcons}(B)$.

(3) If $B \vdash \Gamma$ **:** $\mathtt{cont}$ *where* $\Gamma \equiv \Gamma', x : A, \Gamma''$ *or* $\Gamma \equiv \Gamma', A, \Gamma'$*, then* $FC(A) \subseteq \mathtt{prefcons}(B) \cup \mathtt{defcons}(B)$.

Lemma 3.11.5 *(Uniqueness of Types lemma)*
If $B; \Gamma \vdash \Phi$ **:** $\mathbf{W_1}$ *and* $B; \Gamma \vdash \Phi$ **:** $\mathbf{W_2}$*, then* $\mathbf{W_1} \equiv \mathbf{W_2}$.

For the following lemma's, we need the notions of *subcontext* and *subbook*:

Definition 3.11.6 *(1)* $B' \subseteq B$ *(*B' *is a* subbook *of* B*) if* $\exists B'' : B \equiv B' \circ B''$.

(2) $\Gamma' \subseteq \Gamma$ *(*Γ' *is a* subcontext *of* Γ*) if* $\exists \Gamma'' : \Gamma \equiv \Gamma', \Gamma''$.

(3) Let B *be a book. A definition* $c(x_1, \ldots, x_n) := t/s/n/a/S$ *is* compatible *with* B *if* $c \notin \mathtt{prefcons}(B) \cup \mathtt{defcons}(B)$.

The following lemma is also called *Weakening lemma*:[6]

Lemma 3.11.7 *(Thinning lemma)*

(1) Let $\vdash B$ **:** $\mathtt{book}$ *and* $B' \subseteq B$*. Then:*

(1a) If $B'; \Gamma \vdash \Phi$ **:** $\mathbf{W}$ *where if* Φ *is a definition then it is compatible with* B*, then* $B; \Gamma \vdash \Phi$ **:** $\mathbf{W}$*, and*

(1b) If $B' \vdash \Gamma$ **:** $\mathtt{cont}$ *then* $B \vdash \Gamma$ **:** $\mathtt{cont}$.

(2) Let $B \vdash \Gamma$ **:** $\mathtt{cont}$*,* $\Gamma' \subseteq \Gamma$ *and* Φ *is not a definition. Then* $B; \Gamma' \vdash \Phi$ **:** $\mathbf{W}$ *implies* $B; \Gamma \vdash \Phi$ **:** $\mathbf{W}$.

The following lemmas concern typability of subbooks, subcontexts and subformula of a weakly well-typed book.

Lemma 3.11.8 *(Subbook and Subcontext property)*

(1) Let $\vdash B$ **:** $\mathtt{book}$ *and* $B' \subseteq B$*. Then* $\vdash B'$ **:** $\mathtt{book}$.

(2) Let $B \vdash \Gamma$ **:** $\mathtt{cont}$ *and* $\Gamma' \subseteq \Gamma$*. Then* $B \vdash \Gamma'$ **:** $\mathtt{cont}$.

[6] An extended form of this lemma can be found in [Kamareddine and Nederpelt 2001].

Lemma 3.11.9 *(Subformula property)*
Let $\vdash B$ **:** `book` *and* $B \equiv B' \circ \Gamma \triangleright \Psi \circ B''$. *Then:*

(1) $B' \vdash \Gamma$ **:** `cont`.

(2) $B'; \Gamma \vdash \Psi$ **:** `stat/def`.

(3) For every subformula Φ *of* Γ, $\exists \mathbf{W}, \exists \Gamma'$ *where* $\Gamma \subseteq \Gamma'$: $B'; \Gamma' \vdash \Phi$ **:** $\mathbf{W}$.

(4) For every subformula Φ *of* Ψ, $\exists \mathbf{W}, \exists \Gamma'$ *where* $\Gamma \subseteq \Gamma'$: $B'; \Gamma' \vdash \Phi$ **:** $\mathbf{W}$.

In WTT, there is only one sensible reduction relation, namely 'definition unfolding'. We use symbol $\xrightarrow{\delta}$ for this relation and call it *'δ-reduction'*. Its definition is as expected:

Definition 3.11.10 *Let* $\vdash B$ **:** `book` *and* $\Gamma \triangleright c(x_1, \ldots, x_n) := \Phi$ *a line in* B. *Then* $\xrightarrow{\delta}$ *is the* compatible *relation on subterms of* B *generated by*
$c(A_1, \ldots, A_n) \xrightarrow{\delta} \Phi[x_i := A_i]$,
provided that the occurrence of the latter constant c *is not the 'defining occurrence' in the defining line mentioned. As* δ*-reduction depends on the book in question, we write* $B \vdash c(A_1, \ldots, A_n) \xrightarrow{\delta} \Phi[x_i := A_i]$.
The relation $\xrightarrow{\delta}\!\!\!\!\rightarrow$ *is the symmetric, transitive closure of* $\xrightarrow{\delta}$.

Here $\Phi[x_i := A_i]$, as usual, stands for the simultaneous substitution of x_i by A_i in Φ. Now we have:

Theorem 3.11.11 *(Strong Normalisation; Uniqueness of Normal Form)*
Let $\vdash B$ **:** `book`. *For all subformulas* Ψ *occurring in* B, *relation* $\xrightarrow{\delta}$ *is strongly normalizing (i.e., definition unfolding inside a well-typed book is a well-founded procedure). Moreover, the normal form is unique.*

Corollary 3.11.12 *(Church Rosser)*
If $B \vdash \Phi \xrightarrow{\delta}\!\!\!\!\rightarrow \Phi_1$ *and* $B \vdash \Phi \xrightarrow{\delta}\!\!\!\!\rightarrow \Phi_2$, *then there exists* Ψ *such that* $B \vdash \Phi_1 \xrightarrow{\delta}\!\!\!\!\rightarrow \Psi$ *and* $B \vdash \Phi_2 \xrightarrow{\delta}\!\!\!\!\rightarrow \Psi$.

Theorem 3.11.13 *(Subject Reduction)*
If $\vdash B$ **:** `book`, $B; \Gamma \vdash \Phi$ **:** $\mathbf{W}$ *and* $B \vdash \Phi \xrightarrow{\delta} \Psi$, *then* $B; \Gamma \vdash \Psi$ **:** $\mathbf{W}$.

Other valid lemma's in WTT are:

Lemma 3.11.14 *(Substitution lemma)*
Let $B; \Gamma, x : A \vdash x$ **:** `term/set` *and* $B; \Gamma \vdash A'$ **:** `term/set`.

(1) If $B \vdash \Gamma, x : A, \Delta$ **:** `cont`, *then* $B \vdash \Gamma, \Delta[x := A']$ **:** `cont`.

(2) If $B; \Gamma, x : A, \Delta \vdash \Phi$ **:** $\mathbf{W}$ *and* Φ *is not a definition, then*
$B; \Gamma, \Delta[x := A'] \vdash \Phi[x := A']$ **:** $\mathbf{W}$.

Lemma 3.11.15 *(Condensing lemma)*

(1a) If $B \vdash \Gamma, x : A, \Delta$ **:** `cont` *and* $x \notin \Delta$*, then* $B \vdash \Gamma, \Delta$ **:** `cont`*.*

(1b) If $B; \Gamma \vdash S$ **:** `stat` *and* $B \vdash \Gamma, S, \Delta$ **:** `cont`*, then* $B \vdash \Gamma, \Delta$ **:** `cont`*.*

(2a) If $B; \Gamma, x : A, \Delta \vdash \Phi$ **:** $\mathbf{W}$*,* Φ *is not a definition and* $x \notin \Delta, \Phi$*, then* $B; \Gamma, \Delta \vdash \Phi$ **:** $\mathbf{W}$*.*

(2b) If $B; \Gamma \vdash S$ **:** `stat` *and* $B; \Gamma, S, \Delta \vdash \Phi$ **:** $\mathbf{W}$*, then* $B; \Gamma, \Delta \vdash \Phi$ **:** $\mathbf{W}$*.*

The derivation system for WTT also has the following nice property:

Theorem 3.11.16 *Derivations are syntax-driven.*

That is to say, in order to verify that a certain formula has a certain weak type, there is at most one derivation rule applicable. Hence it is easy and straightforward to check weak typing.

The syntax-driven behaviour of WTT can also be expressed in a *Generation lemma*, for which we refer to [Kamareddine and Nederpelt 2001].

Finally, we state the following corollary:

Corollary 3.11.17 *(Decidability of weak type checking and weak typability)*

(1) Weak type checking is decidable, i.e., there is a decision procedure for the question

$B; \Gamma \vdash \Phi$ **:** $\mathbf{W}$ *?.*

(2) Weak typability is computable, i.e., there is a procedure deciding whether an answer exists for

$B; \Gamma \vdash \Phi$ **:** *? and if so, delivering the answer.*

Chapter 4

Examples

In this section we give four examples of mathematical texts and their translation into WTT. We also discuss some syntactical properties of the WTT-texts, showing how they fit in our formal syntax of chapter 2. Moreover, we give an idea how the WTT-texts can be obtained with the derivation system of chapter 3.

4.1 Example 1

Our first example is a simple phrase, taken from a mathematical text:

[*1] 'the square root of the third power of a natural number'

We give two possible translations into WTT:

Translation 1.1: $\texttt{Noun}_{x:\mathbb{R}}\exists_{n:\mathbb{N}}(x = \sqrt{n^3})$
Translation 1.2: $\texttt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})$

Note that translation 1.1 is more informative in that it gives the final type of the noun (viz. $\mathbb{R}$), but that translation 1.2 is more compact.

In both cases it is easy to verify that the translation is a weakly well-typed noun. We check this for translation 1.2. We start with a preface incorporating all external constants of the phrase:

	constant name	weak types *in*	weak type *out*
(i)	^{3}	(`term`)	`term`
(ii)	$\surd$	(`term`)	`term`
(iii)	$\mathbb{N}$	( )	`set`
(iv)	`Abst`	(`term`)	`noun`

As an exercise, we give the categories (see chapter 2) of all subexpressions of the phrase in translation 1.2:

subexpression	category
n	$\mathtt{t}$
n^3	$\mathtt{t}$
$\sqrt{n^3}$	$\mathtt{t}$
n	$\mathtt{t}$
$\mathbb{N}$	σ
$n:\mathbb{N}$	$\mathcal{Z}$
$\mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})$	ν

We now give a derivation for the fact that $\mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})$ is a noun. For this purpose, we have to derive

$B;\Gamma \vdash \mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})\ \mathtt{:noun}$

for some B and Γ. In this case, it is clear that $B = \Gamma = \emptyset$. We assume that n belongs to the set $\mathtt{V^t}$ of term variables:

$(*)\ \ n \in \mathtt{V^t}$

The desired derivation is (numbers i, ii, iii and iv refer to the preface):

(1)		$\vdash$	$\emptyset\ \mathtt{:book}$	$(emp{-}book)$
(2)	$\emptyset$	$\vdash$	$\emptyset\ \mathtt{:cont}$	$(emp{-}cont, 1)$
(3)	$\emptyset;\emptyset$	$\vdash$	$\mathbb{N}\ \mathtt{:set}$	$(ext{-}cons, 1, 2, iii)$
(4)	$\emptyset$	$\vdash$	$n:\mathbb{N}\ \mathtt{:cont}$	$(term{-}decl, 1, 2, 3, *)$
(5)	$\emptyset; n:\mathbb{N}$	$\vdash$	$n\ \mathtt{:term}$	$(var, 1, 4, *)$
(6)	$\emptyset; n:\mathbb{N}$	$\vdash$	$n^3\ \mathtt{:term}$	$(ext{-}cons, 1, 4, i, 5)$
(7)	$\emptyset; n:\mathbb{N}$	$\vdash$	$\sqrt{n^3}\ \mathtt{:term}$	$(ext{-}cons, 1, 4, ii, 6)$
(8)	$\emptyset;\emptyset$	$\vdash$	$\mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})\ \mathtt{:noun}$	$(bind, 1, 4, iv, 7)$

This derivation is given in the format of *forward reasoning*: we start with smaller subexpressions and build larger ones. Of course, the derivation can also be developed in the format of *backward reasoning* or, otherwise said, in the *goal-directed* manner. In that case we start with the ultimate *goal* (line (8)) and investigate how this goal can be reached. The only applicable rule to get line (8) is (*bind*). This gives new goals, generated by the main symbol (`Abst`) and (*iv*). These goals are the judgements written in lines (1), (4) and (7), and so forth. Note that our derivation system is *syntax-driven* in the sense that for each goal, *only one* rule is applicable.

We vary a bit on this example and look at the following *statement*:

[*2] '8 is the square root of the third power of a natural number'

For the translation, the easiest thing is to use our previous example, obtaining the statements:

Translation 2.1: $8 : \mathtt{Noun}_{x:\mathbb{R}}\exists_{n:\mathbb{N}}(x = \sqrt{n^3})$
Translation 2.2: $8 : \mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3})$

But in this case, there is a shorter and more elegant translation possible, viz. the 'logical' statement:

Translation 2.3: $\exists_{n:\mathbb{N}}(\sqrt{n^3} = 8)$

Remark 4.1.1 *The last mentioned example shows, that the 'ordinary' mathematical formulas (so without our extension with nouns and adjectives) are often good enough for translations of a mathematical text into a formal form. In the case above, translation* 2.3 *is quite satisfactory, albeit that translations* 2.1 *and* 2.2 *are, in a sense, 'closer' to the original text.*

However, when definitions enter the stage, then the extension with nouns and adjectives, and hence with `Noun`*,* `Abst`*,* `Adj`*,* $\uparrow$ *and* $\downarrow$*, is more appropriate. See sections 4.2, 4.3 and 4.4 for examples.*

Now look at the statement:

[*3] 'The square root of the third power of a natural number is non-negative'

Again, we can use translations 1.1 or 1.2:

Translation 3.1: $\forall_{y:\mathbb{R}}(y : \mathtt{Noun}_{x:\mathbb{R}} \exists_{n:\mathbb{N}}(x = \sqrt{n^3}) \Rightarrow y \geq 0)$
Translation 3.2: $\forall_{y:\mathbb{R}}(y : \mathtt{Abst}_{n:\mathbb{N}}(\sqrt{n^3}) \Rightarrow y \geq 0)$

But here, again, a considerably simpler and more 'traditional' translation is possible:

Translation 3.3: $\forall_{n:\mathbb{N}}(\sqrt{n^3} \geq 0)$

Remark 4.1.2 *The above examples show, that the translation of a mathematical text into WTT is not 'compositional'. Look at the indefinite article 'a' as it occurs in the phrases* [*1] *or* [*2]*: '... the third power of a natural number'. Its translation is* `Noun`*,* `Abst` *or* $\exists$*, see 1.1, 1.2, 2.1, 2.2 and 2.3. However, when embedding it in statement* [*3]*, then none of these roles can be maintained: in all three cases 3.1, 3.2 and 3.3, the translation of article 'a' is changed into* $\forall$*.*

This non-compositionality is present in several places in the translation process. It has to be said that the indefinite article 'a' is an especially versatile word. It may have roles `Noun`*,* `Abst`*,* $\exists$ *or* $\forall$*, as shown above, but still other roles are possible, in particular: the introduction of a variable in the context of a definition. For the latter, see section 4.3. The 'versatility' of the definite article 'the' is hardly less.*

4.2 Example 2

Our second example concerns a definition and its application. Consider the following mathematical text:

> DEFINITION A *Fermat-sum* is a natural number which is the sum of two squares of natural numbers.
>
> LEMMA The product of a square and a Fermat-sum is again a Fermat-sum.

A translation into WTT could be:

a Fermat-sum $:= \texttt{Noun}_{n\in\mathbb{N}}\exists_{k\in\mathbb{N}}\exists_{l\in\mathbb{N}}(n = k^2 + l^2)$
$\forall_{u:\ a\ square}\forall_{v:\ a\ Fermat\text{-}sum}(uv :\ a\ Fermat\text{-}sum)$

This small WTT-book B consists of two lines, one being a definition and the other a statement. Both lines have an empty context. So the abstract format of B is:
$\emptyset \triangleright D \ \circ\ \emptyset \triangleright S$.

Note how the defined constant '*a Fermat-sum*', a noun, is used in the statement following the definition. The noun '*a square*' is not defined in B, hence the text assumes that the definition of this noun has been given beforehand: it is an external constant to B. Hence, it has to be incorporated in the preface when applying the derivation rules in order to establish $\vdash B : \texttt{book}$.

4.3 Example 3

Our third example is from mathematical analysis. It contains the definitions of 'difference quotient' and of 'differentiable' and a statement using the latter definition:

> 'Definition. Let $h \neq 0$, let f be a function from A to $\mathbb{R}$, $a \in A$ and $a + h \in A$. Then $\frac{f(a+h)-f(a)}{h}$ is the *difference quotient* of f in a with difference h. We call f *differentiable* at $x = a$ if $\lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists.
> The function $\sqrt{|x|}$ is not differentiable at 0.'

Both definitions require a context. We use the 'flag notation' to build the context in the translation of this text:

(1) $\boxed{\rightsquigarrow A \subseteq \mathbb{R}}$
(2) $\quad\boxed{\rightsquigarrow f : A \to \mathbb{R}}$
(3) $\quad\quad\boxed{\rightsquigarrow a \in A}$
(4) $\quad\quad\quad\boxed{\rightsquigarrow h \in \mathbb{R}}$
(5) $\quad\quad\quad\quad\boxed{h \neq 0}$
(6) $\quad\quad\quad\quad\quad\boxed{a + h \in A}$
(7) $\quad\quad\quad\quad\quad\quad$ *the difference quotient of* $f := \frac{f(a+h)-f(a)}{h}$
(8) $\quad\quad\quad$ *f is differentiable at a* $:= \lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists
(9) $\neg(\lambda_{x:\mathbb{R}}(\sqrt{|x|})$ is differentiable at 0)

We discuss the details of this flag-structured WTT-book below. First, we explain the flag notation and its advantages over the notation in WTT-format as described in chapter 2.

The flag notation is a shorthand for dealing with contexts: since (parts of) contexts are frequently repeated in succeeding lines, it saves space to allow multiple use of context entries

(declarations and assumptions). A second advantage of the flag notation is that the structure of a WTT-book is more easily visible.

Remark 4.3.1 *We emphasize explicitly that the flag notation is no more than 'sugaring', flags do neither exist in the WTT-syntax, nor in the derivation system for WTT. The flag notation provides for a certain 'view' on WTT-texts which can be helpful for a human reader, e.g. when examining or inspecting a formalized mathematical text in an electronic library.*

The above flag-text is the 'sugared' version of the book given below. For convenience we first abbreviate:

$\Gamma_1 = A \subseteq \mathbb{R}, f : A \to \mathbb{R}, a : A, h : \mathbb{R}, h \neq 0, a + h \in A$
$\Gamma_2 = A \subseteq \mathbb{R}, f : A \to \mathbb{R}, a : A$

The book matching the above flag-text consists of the $\circ$-concatenation of the following three lines:

$\Gamma_1 \quad \triangleright \quad$ *the difference quotient of* $f := \frac{f(a+h)-f(a)}{h}$
$\Gamma_2 \quad \triangleright \quad f$ *is differentiable at* $a := \lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists
$\emptyset \quad \triangleright \quad \neg(\lambda_{x:\mathbb{R}}(\sqrt{|x|})$ is differentiable at $0)$

Notice how the context administration works in the flag notation. For example, the part

$A \subseteq \mathbb{R}, a : A, f : A \to \mathbb{R}$

needs not to be repeated for the definition in line (8), since it is still 'open' (the three flagpoles of lines (1) to (3) are still present in line (8)). In example 4 (see section 4.4) we show again how convenient the flag notation is.

We now discuss the details of the above WTT-book.

- Each context element is separately placed in a 'flag'. The attached 'flagpole' registers how long the context element is supposed to be present. Nesting of the flags fixes the order of the context elements.
- Context elements can be either declarations or assumptions. In order to visualize the difference, we start declarations with symbol '$\rightsquigarrow$'. (The symbol '$\rightsquigarrow$' 'points at' the declared variable.)
- Note that the indefinite article 'a' occurring in the noun 'a function' in the original text, is translated here into the flag in line (2). We pointed at this possibility in Remark 4.1.2.
- In the WTT-book there are only *bound* variables. Notice for example that all free variables in line (8) are bound in the context lines (1) to (3).
- In translating the original mathematical text into (flag-style) WTT-format, we added the declaration of A, which was not explicitly stated in the mathematical text.
- Note that the statement $a \in A$ is translated as a declaration, whereas $a + h \in A$ is an assumption. (Both statements are treated similarly in the original mathematical text.)

- The context elements containing h are rather arbitrarily arranged in the original mathematical text. We concentrate these elements in lines (4) to (6). This enables us to make a smooth use of the context administration: for the definition in line (8) we just skipped – i.e. cut the flagpoles of – the context elements (4) to (6). As a matter of fact, h is a bound variable in $\lim_{h\to 0}\frac{f(a+h)-f(a)}{h}$ and should not be *declared* in the context. Moreover, the assumptions $h \neq 0$ and $a + h \in A$ are not desired for the definition in line (8). In general, it is of importance to find a 'minimal' context for a definition or statement.

- In line (7) we have a *phrase definition* of the form $\mathtt{C}^{\mathtt{t}}(\vec{\mathtt{V}}) \;:=\; \mathtt{t}$. (See section 2.8.1.) The parameter list of the constant 'the difference quotient' is (A, f, a, h), but only f is accounted for in the WTT-book. (A more precise formulation would be to write: 'the difference quotient of f in a with difference h (where f has domain A)'.)

- The definition in line (8) is a *statement definition*, corresponding to the abstract format $\mathtt{C}^{\mathcal{S}}(\vec{\mathtt{V}}) \;:=\; \mathcal{S}$. (See section 2.8.2.) Of the parameter list (A, f, a) of the constant 'is differentiable at', we only find f and a in the book. The definition can also be given as a phrase definition (namely as the definition of the *adjective* 'differentiable'), in the format $\mathtt{C}^{\alpha}(\vec{\mathtt{V}}) := \alpha$. (See section 2.8.1.) This case is elaborated below.

- The limit-binder $\lim_{h\to p}$ can be considered to be a (non-binding!) constant of three variables, $\lim(p, X, g)$, defined beforehand in a context consisting of the three declarations $p \in \mathbb{R}, X \subseteq \mathbb{R}, g : X \to \mathbb{R}$. That is, we consider $\lim_{h\to p} g(h)$ to be an alternative notation for $\lim(p, X, g)$, with X the domain of function g. In line (8) we have the *instantiation*

  $\lim(0, X, \lambda_{h\in X}\frac{f(a+h)-f(a)}{h})$,

  with $X = \{x \in \mathbb{R} | a + x \in A \wedge x \neq 0\}$.

- The *existence* of the limit as required in line (8) should also have been defined beforehand, in a piece of mathematical theory stating:

  (1) The definition of the *limit-property* of a function $g : X \to \mathbb{R}$ with respect to a point p on the real x-axis. This property is expressed in an existential statement (probably with ε's and δ's), for example in the statement definition:

  g has the limit-property in p :=
  $\exists_{l\in\mathbb{R}}\forall_{\varepsilon\in\mathbb{R}^+}\exists_{\delta\in\mathbb{R}^+}\forall_{x\in X\setminus\{p\}}(|x - p| < \delta \Rightarrow |g(x) - l| < \varepsilon)$[1]

  (2) The theorem that *if* the limit-property holds, then the existing l is unique.

  (3) The definition of $\lim(p, X, g)$ as being this unique l, again under the assumption that the limit-property holds.

  Now '$\lim(p, X, g)$ exists' is equivalent to '$g : X \to \mathbb{R}$ has the limit-property in p'.

[1] This is a statement definition of the form $\mathtt{C}^{\mathcal{S}}(\vec{\mathtt{V}}) := \mathcal{S}$. It needs a context with p, X and g as declared variables.

- In line (9), parameters A, f and a of the statement 'f is differentiable at a' are instantiated with $\mathbb{R}$, $\lambda_{x:\mathbb{R}}(\sqrt{|x|})$ and 0, respectively. The resulting statement needs no context. The same holds for its negation with the logical constant $\neg$.

We give an alternative translation into (flag-styled) WTT where 'differentiable (at)' is defined as an adjective. Since $f : A \to \mathbb{R}$ becomes a subscript declaration in this translation, it should be left out of the context for statement (8′). (Lines (1) to (7) are the same as before, in line (9′) we employ the typing symbol ':' instead of the verb 'is'.)

(1) $\boxed{\rightsquigarrow A \subseteq \mathbb{R}}$

(2) $\quad\boxed{\rightsquigarrow f : A \to \mathbb{R}}$

(3) $\quad\quad\boxed{\rightsquigarrow a \in A}$

(4) $\quad\quad\quad\boxed{\rightsquigarrow h \in \mathbb{R}}$

(5) $\quad\quad\quad\quad\boxed{h \neq 0}$

(6) $\quad\quad\quad\quad\quad\boxed{a + h \in A}$

(7) $\quad\quad\quad\quad\quad\quad$ *the difference quotient of* f $:=$ $\frac{f(a+h)-f(a)}{h}$

(8′) $\quad\quad\quad$ *differentiable at* a $:=$ $\mathtt{Adj}_{f:A\to\mathbb{R}}(\lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists)

(9′) $\neg(\lambda_{x:\mathbb{R}}(\sqrt{|x|}) :$ differentiable at 0)

Derivations leading to either of the two WTT-books given above, need many small steps, but can be constructed straightforwardly. We omit the derivations themselves. We only give a number of remarks regarding these derivations:

- In line (1) of both WTT-books we type A by declaring it to be a *subset* of $\mathbb{R}$. This is not according to the rules. However, we may consider the 'declaration' $A \subseteq \mathbb{R}$ to be shorthand for the proper declaration $A :$ SET followed by the *assumption* $A \subseteq \mathbb{R}$. Another option is to rewrite $A \subseteq \mathbb{R}$ as $A : \wp(\mathbb{R})$.
- Equality, addition, subtraction and division should also obtain weak types in the preface. Their common weak in-type is $(\mathtt{t}, \mathtt{t})$, their weak out-type is $\mathtt{t}$. The weak in-type of $\sqrt{}$ is $(\mathtt{t})$, the out-type is $\mathtt{t}$. Logical operator $\neg$ has weak in-type $(\mathcal{S})$ and weak out-type $\mathcal{S}$.
- *Function application* as in $f(a)$ and $f(a+h)$ can be treated as a binary external constant `appl` with weak in-type $(\mathtt{t}, \mathtt{t})$ and out-type $\mathtt{t}$.
- In lines (9) and (9′) we use an internally defined notion ('differentiable'). In both translations, either as a constant for a statement or as a constant for an adjective, we can apply rule $(int{-}cons)$ for establishing well-typedness.

4.4 Example 4

Finally we give an example from elementary algebra. We consider the following theorem together with its proof:

THEOREM. Let G be a set with a binary operation $\cdot$ and left unit element e. Let H be a set with binary operation $*$ and assume that φ is a homomorphism of G onto H. Then H has a left unit element as well.

PROOF. Take $e' = \varphi(e)$. Let $h \in H$. There is $g \in G$ such that $\varphi(g) = h$. Then

$e' * h = \varphi(e) * \varphi(g) = \varphi(e \cdot g) = \varphi(g) = h,$

hence e' is left unit element of H. □

A translation into WTT is:

(1) $\boxed{\rightsquigarrow G : \text{ SET}}$
(2) $\quad\boxed{\rightsquigarrow \cdot : (G \times G) \to G}$
(3) $\quad\quad\boxed{\rightsquigarrow e : G}$
(4) $\quad\quad\quad\boxed{e : \text{ a left unit element of } G}$
(5) $\quad\quad\quad\quad\boxed{\rightsquigarrow H : \text{ SET}}$
(6) $\quad\quad\quad\quad\quad\boxed{\rightsquigarrow * : (H \times H) \to H}$
(7) $\quad\quad\quad\quad\quad\quad\boxed{\rightsquigarrow \varphi : G \to H}$
(8) $\quad\quad\quad\quad\quad\quad\quad\boxed{\varphi : \text{ a surjective homomorphism}}$
(9) $\quad\quad\quad\quad\quad\quad\quad e' := \varphi(e)$
(10) $\quad\quad\quad\quad\quad\quad\quad\boxed{\rightsquigarrow h : H}$
(11) $\quad\quad\quad\quad\quad\quad\quad\quad \exists_{g \in G}(\varphi(g) = h)$
(12) $\quad\quad\quad\quad\quad\quad\quad\quad\boxed{\rightsquigarrow g : G}$
(13) $\quad\quad\quad\quad\quad\quad\quad\quad\quad\boxed{\varphi(g) = h}$
(14) $\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad e' * h = \varphi(e) * \varphi(g) = \varphi(e \cdot g) = \varphi(g) = h$
(15) $\quad\quad\quad\quad\quad\quad\quad\quad e' * h = h$
(16) $\quad\quad\quad\quad\quad\quad\quad e' : \text{ a left unit element of } H$
(17) $\quad\quad\quad\quad\quad\quad\quad H \text{ has a left unit element}$

This book is a good example of the gains obtained by using the flag notation. In fact, the book consists of five lines with contexts overlapping largely. Schematically denoted, the lines are:

$(1) - (8)$	$\triangleright$	(9)
$(1) - (8), (10)$	$\triangleright$	(11)
$(1) - (8), (10), (12), (13)$	$\triangleright$	(14)
$(1) - (8), (10)$	$\triangleright$	(15)
$(1) - (8)$	$\triangleright$	(16)
$(1) - (8)$	$\triangleright$	(17)

The first of these lines is a definition, the second to fifth are intermediate results (statements), being part of the proof, and the last line expresses the theorem (a statement, as well). Note that we made the choice to translate the proof first, and putting the theorem at the end. (This is not necessary for the translation into WTT, but it makes a further translation into Type Theory easier.)

We note the following about our translation:

- Lines (1) to (3) could be concentrated in the single declaration:

  $$(1') \qquad \boxed{\rightsquigarrow (G, \cdot, e) : \text{ a groupoid with left unit element}}$$

  and something similar for lines (5) and (6).

  In fact, the use of WTT (or Type Theory!) in practice often 'asks for' such a kind of abbreviations for dependent parts of a context (also called *telescopes*, see [Zucker 77]).

- Again we see that not all parameters are accounted for in the WTT-book. For example, one is inclined to make lines (4) and (8) more specific in the following manner:

  (4) $e :$ a left unit element of G *with respect to* $\cdot$,

  (8) $\varphi :$ a homomorphism of G onto H *with respect to* $\cdot$ *and* $*$, *respectively.*

- Note that line (9) is a definition! This is not immediately visible in the original text: 'Take $e' = \varphi(e)$', which suggests no more than an equality. But the fact that e' is 'new' here immediately leads to the interpretation as a definition.

- We left out the parameter list for the newly defined constant e' in line (9). As we said before, this list can be reconstructed since it is equal to the list of the declared variables occurring in the context of the definition. Hence, the 'official' format of line (9) would be:

  $e'(G, \cdot, e, H, *, \varphi) := \varphi(e)$.

  Moreover, in the subsequent *uses* of the constant e' in lines (14) to (16), there should be parameter lists as well behind each occurrence of constant e'. In this case, however, the instantiations for the variables in the variable lists of e' in lines (14) to (16) are *exactly the same* as in the definition itself (so variable G is instantiated with G, variable $\cdot$ with $\cdot$, etc.). This shows once more that it can be very economical to allow a shorter notation for parameter lists such that reconstructable or unchanged heads of parameter lists may be omitted. This is only sugar and can always be undone. (In Automath, [De Bruijn 70], this is a syntactic feature.)

- Another thing is, that the context for the definition $e' := \varphi(e)$ in line (9) of the example is larger than necessary. In fact, we could do with context $G :$ SET $, e : G, H :$ SET $, \varphi : G \to H$ and corresponding parameter list (G, e, H, φ). In our example, however, we keep close to the original CML-text, in which the local definition $e' = \varphi(e)$ is made in the full context of the theorem.

- Lines (12) and (13) are a direct consequence of our wish to *avoid free variables* in a WTT-book. Note that g is a *bound* variable in line (11), but without line (12) it would be a *free variable* in line (14)!

  It is, in fact, a free variable in the original mathematical text. This is due to the habit in mathematics to extend the scope of an existentially bound variable outside the formula in which it is introduced:

  'There is $g \in G$ such that $\varphi(g) = h$. Then $\ldots \varphi(e) * \varphi(g) = \varphi(e \cdot g) = \varphi(g) = \ldots$.'

  The occurrences of g after 'Then...' are free! What is actually happening in such cases is that the *existence* of such a g automatically induces the (silent) introduction of such a g (for convenience *called* g again) outside the scope of the $\exists$-binder.[2] Repair of this sloppy habit is straightforward, by using the corresponding logical rule of $\exists$-elimination:

  $$\frac{\exists_{x \in U}(P(x)), \quad \forall_{x \in U}(P(x) \Rightarrow r)}{r}$$

  This rule is the background for lines (12) and (13), since lines (12) to (14) prove that

  $$\forall_{g \in G}(\varphi(g) = h \Rightarrow e' * h = h),$$

  which, together with (11) and $\exists$-elimination justifies the conclusion (15)!

  Hence, the apparent 'detour' via lines (12) to (14) is necessary in order to mend a flaw occurring frequently in mathematical texts. So this complication is not the fault of our translation. On the contrary, the translation shows where mathematical language has its weakness.

- Line (14) is a chain of statements: there are as many statements in this line as there are $=$-signs, and maybe even one more: the 'implicit conclusion' $e' * h = h$, which is repeated in (15) (but in a smaller context).

- The translation is as near as possible to the original text (but for the mending of the sloppiness with the existential quantifier, see above). This implies that intermediate results which one would expect in the formal version, are nevertheless left out. For example, one could expect the following line preceding line (16), in the same context (1) – (8):

  $\forall_{h \in H}(e' * h = h)$.

  (This is the logical consequence of lines (10) and (15), due to the $\forall$-introduction rule.)

- It is important to note (again) that we *omit all justifications* in WTT-books. Our reason for doing this is, that we want to have a rather simple syntax for WTT, which is neither concerned with 'meta-arguments' about the logical or mathematical correctness, nor with the interdependence of statements induced by these correctness arguments. Hence, we find no words like 'Since', 'Hence' or 'Because' in WTT.

[2] In a more general sense, this is a case of what is called *non-monotonicity*. This is a discourse phenomenon which appears frequently in natural language and which was one of the reasons for the development of DRT ('Discourse Representation Theory', see [Kamp and Reyle 93] and [Eijck and Kamp 96]).

The drawback of this is, that the argumentation structure of a book can become unclear. This is mended when we make the next step: translating WTT into Type Theory. See the following chapter.

However, as we mentioned in Remark 3.2.6, there is another solution: we may choose to add to the WTT-book above, two columns on either side, one with labels and one with comments.

In the *label* column, to the left of the sample WTT-text, we could write

- At line (9): 'Proof start' and 'Definition',
- At line (16): 'Proof end',
- At line (17): 'Theorem'.

In the *comments* column, to the right of the text, we could write for example

- At line (11): 'From (8)' or 'Because of (7), (8) and (10)',
- At line (15): '$\exists$-elimination on (11) and (12) to (14)',
- At line (16): 'From (10) to (15), $\forall$-introduction and definition of *left unit element*'.

Again, we do not give a derivation for the WTT-book above. There are no special problems in making such a derivation, the system of derivation rules shows the way to do this.

Chapter 5

From WTT to Type Theory

When a mathematical text has been properly expressed in Weak Type Theory, it fits in a well-defined grammar, as given in chapter 2. Moreover, there is a derivation system for books in WTT, as we showed in chapter 3. However, the typing requirements imposed by the grammar or the derivation system are rather weak.

If we are interested in the mathematical rigour of a given text, it is advisable to move over to a stronger theory, for example (a form of) Type Theory itself. For Type Theory has the powerful property that a succeeded type-check of a certain proposition entails 'correctness' in the logico/mathematical sense.

In this chapter we discuss the possibilities of the adaptation of a WTT-book such that it fits in Type Theory. It turns out that the first step is choosing an appropriate Type Theory (section 5.1). The next step is the adaptation itself. Since WTT is, indeed, an 'attenuated' form of Type Theory, it is rather straightforward to indicate the aspects which have to be changed or added in order to attain (full) Type Theory. The actual work that has to be done to translate a WTT-book into Type Theory can, however, be considerable, since Type Theory requires all the details, in a very precise manner. Another thing is that WTT has no so-called 'proof-objects', i.e. terms embodying a formal justification of a proposition. These proof-objects have to be added and this is not an easy job. It requires the skills of a person having experience with Type Theory and having also a firm background in mathematics.

5.1 Choosing an appropriate Type Theory

Type Theories come in various versions, from simply typed λ-calculus via Pure Type Systems to extended systems as underlying existing theorem provers such as Coq and LEGO.[1] In order to be able to successfully convert a WTT-book into Type Theory, we need at least the power of $\lambda P\underline{\omega}$, as shown in the Automath project ([De Bruijn 70]). A version of λC (the Calculus of Constructions) would even be more appropriate, since the parameter mechanism accompanying constants is not fully covered by $\lambda P\underline{\omega}$ (see [Franssen 2000]).

[1] For a classification, see [Barendregt 92]. We only consider here the so-called *explicitly* typed versions (or *type theories in Church-style*), in which each variable is explicitly typed. Moreover, we do not consider Martin-Löf Type Theory, which is based on a different principle.

For a discussion of some problems arising when directly translating mathematical content into Type Theory, see [Geuvers et al. 2001]. Also [Oostdijk 2001] is interesting in this respect.

In order to faithfully represent mathematical texts, some sort of constant administration is indispensable. Defined constants are core entities of mathematical texts and make the development of mathematical theories feasible.

Therefore, constants must be first class citizens in a Type Theory used for formalizing mathematics, and hence also for a Type Theory which is intended for the completion of a WTT-book. This has consequences for the choice which we make for the Type Theory. Below, we discuss some possibilities. Thereafter we compare 'books' with single 'judgements'. Finally we list a number of points to be considered for choosing a Type Theory suited for WTT.

5.1.1 Extending Type Theory with constants

There have been several attempts to enrich Type Theory with definitions (and hence with defined constants and parameters). See e.g. [Bloo et al. 2002], [Kamareddine et al. 2001], [Severi and Poll 94] and [Laan 97]. An important rule in this respect expresses the typing of an *instantiation* $c(A_1, \ldots, A_n)$ of a constant $c(x_1, \ldots, x_n)$. Such a typing rule has the form:

if c is defined in a certain context as a of type T:

$x_1 : T_1, \ldots, x_n : T_n \triangleright c(x_1, \ldots, x_n) := a : T$,

then:

$$\frac{A_j : T_j[x_i := A_i]_{i=1}^{j-1} \text{ for } j = 1, \ldots, n}{c(A_1, \ldots, A_n) : T[x_i := A_i]_{i=1}^{n}}$$

Note that each A_j must have the 'proper' type, namely the same type T_j as x_j has, but *with the appropriate substitutions*, viz. of A_i for the x_i $(i = 1, \ldots, j-1)$ occurring in T_j. The reason is that, in a context $x_1 : T_1, \ldots, x_n : T_n$, a type T_j of x_j may *depend on* the x_i with $1 \leq i < j$. Thus, the instantiation replacing x_i with A_i has also consequences for the type T_j.

Another general rule necessary for the dealing with constant, is a 'constant reduction operator' (see also Definition 3.11.10), in the above case leading to a reduction of the form:

$c(A_1, \ldots, A_n) \xrightarrow{\delta} a[x_i := A_i]_{i=1}^{n}$.

This reduction enables the 'unfolding' of a definition.

Many existing type systems with definitions incorporate all relevant constants *inside* the judgement-under-consideration. Constants 'global' to a statement $M : N$ in the type-theoretical judgement[2] $\Gamma \vdash M : N$ may be introduced in a special kind of 'declaration', embedded in the context Γ:

$\Gamma = \ldots, c(x_1, \ldots, x_n) = a : T, \ldots,$

and 'local' constants may be introduced at the head of a subterm of either M or N as

$d(y_1, \ldots, y_m) = b : T' \ \underline{in}\ t$,

where t is the subterm in which the constant d occurs (with instantiated parameters).

[2] We recall that a judgement $\Gamma \vdash M : N$ in Type Theory can be interpreted as: it is derivable that expression M has type N in context Γ.

In this manner, type systems with definitions maintain the same format as type systems without. Note that in Type Theory, the notion 'judgement' is the largest entity and by the incorporation of constants in a judgement as described above, this central status of the 'justification' is not disturbed. An advantage of this procedure is, of course, that the rich and well-elaborated field of Type Theory can be fruitfully exploited.

5.1.2 Books versus judgements

A derivation in Type Theory is generally guided by the wish to 'justify' a certain judgement (which will appear as the final conclusion of the derivation). A judgement in Type Theory usually has the form $\Gamma \vdash M : N$, whereas in WTT the 'inhabitant' M of N is often omitted (in particular when M codes a proof).

In mathematics, however, the ultimate construction is not the line, but the *discourse* (or 'book'), which is a *sequence* of (interconnected) lines, expressing theorems, lemma's, definitions and other constructs. In this view, the *discourse level* is dominant over the *judgement level*.

In Type Theory there has long been a tradition to dismiss theorems, definitions and the like, in order to keep Type Theory simple and transparent. This is a theoretically immaculate point of view, since definitions can be undone ('unfolded') by replacing a defined constant with its 'content'. Something similar holds for theorems and the like. But in a practical setting we need these auxiliary notions, which are all covered by the notion '(defined) constant'. (An intermediate solution is, as we observed above, to *incorporate definitions in judgements*, either in the context ('global definitions') or in the statements themselves ('local definitions').)

We, however, prefer not to take the judgement, but the *book* as the central notion of the Type Theory in which we translate mathematics. Since a book can be defined in Type Theory as a *sequence* of judgements, this point of view widens the angle of observation. Above we already gave good arguments to consider this possibility. We elaborate a bit on this subject:

- The Automath project, a pioneering form of Type Theory, uses the *book* as its central entity. Such an Automath book has a great resemblance with what we call a (WTT-)book. [3]

  Automath books consists of 'lines' (comparable with our WTT-lines) which all come in one standard format[4]:

  $\Gamma \triangleright c(x_1, \ldots, x_n) := M : N,$

  expressing that, in context Γ, constant c – with parameters $x_1, \ldots, x_n$ – is a name for M of type N. It follows that constants are 'first class citizens' in Automath. Such a constant c can denote an 'object' M (which is the most common use of constants outside Type Theory), but it can also be the name of a *proof* M of the *proposition* N. In the latter case, the propositions-as-types character of Type Theory is applied.

[3] Automath, however, differs considerably from WTT in that it is a *complete* form of type theory, allowing no gaps in the reasoning. That is to say: all logico-mathematical justifications are incorporated and its syntax is involved to such an extent, that syntactical correctness entails mathematical correctness.

[4] We present Automath here in a format suited to Type Theory.

In the Automath project it was shown that many sizeable corpora of mathematical texts, taken from a great variety of areas in mathematics, can be translated into the above described 'book'-format. A computer program enables the final check of the translated texts on syntactical correctness. This entails 'logico-mathematical correctness'. Hence, the book-format has proved to be an adequate format for the formalized representation of mathematical texts.

- A mathematical text has 'naturally' a line-for-line character when put on paper. This is another argument to prefer 'books' over single judgements or lines. Not only reasonings are built step by step ('line by line'), but also theories, including the necessary definitions, theorems and the like. Such lines are interdependent in the sense that all kinds of notions introduced in previous lines may be used in later ones. This concerns defined notions which may be used again (this is one of the most important motivations for making a definition in the first place!), but also theorems and lemma's which may be used over and over again. This is standard procedure in mathematics. The use of a constant or a theorem usually takes place in a different context, so instantiations of variable lists are more rule than exception.

  In order to give a good theoretical reflection of the interdependencies of the lines ('judgements') in a book, it appears a good thing to *preserve* the separation between lines and books. It is of course feasible to condense a 'book' into one 'line', by compressing all necessary constants in the context (or in the heads of subterms) of the last line of the book. The question remains whether the gains of this approach are higher than the losses.

- In any practical application of Type Theory, also outside of mathematics, the need for constants is felt immediately. This is not only for conceptual reasons – people like to think in terms of entities and it is most convenient to give them a name in order to have a good reference – but also for reasons of feasibility: terms in Type Theory soon become forbiddingly large, the complexity of 'pure' typetheoretical terms grows enormously. When one admits to give constants the rights of first-class citizens, then a natural next step is to allow definitions as separate entities, as well. A definition is a kind of *judgement* in its own right. Therefore it deserves its own place, as a separate judgement between other judgements, and not hidden *inside* a judgement. This pleads again for taking the *book* as the core notion for a Type Theory intended for mathematics.

5.1.3 Other desiderata

Apart from the points we mentioned in the two sections above about adding constants and the advantages of books over lines or judgements, there are the usual, well-known decisions which have to be taken before making the choice of an appropriate Type Theory. These concern notions for which Type Theory is not immediately well-equipped. We mention a number of them:

- **Equality.** There are two different kinds of equality which have to be taken care of (see [Barendregt and Geuvers 2001]):

  - *Definitional equality.* This is the 'equality' implied by the reduction relations in the type system, e.g. β (the λ-calculus reduction) and δ (the unfolding operation for definitions). Both can be built in (equality is then decided by means of a derivation inside the type system) or external (equality is decided outside the type system, it is a 'side condition').
  - *Book equality.* This is a user-defined equality, formulated inside the Type Systems. It can be embedded in the type-theoretical frame: the equality of M and N, both of type A, is justified by finding an *inhabitant* p of the type $M =_A N$ (and proving that $p : M =_A N$).

- **Equivalences.** A related question is how to deal with equivalence relations and equivalence classes. The pair of a set V and an equivalence relation $\sim$ on V, is sometimes called a *setoid* in Type Theory and treated as a special syntactic notion.

- **Extensionality.** Equality of functions can be of two kinds: intensional ('equality of functions is equality of algorithms') and extensional ('equality of functions is equality of graphs'). In the latter case, there is a rule 'extensionality' which allows a derivation of $f =_{A\to B} g$ if $\forall_{x\in A}(fx =_B gx)$. See again [Barendregt and Geuvers 2001].

- **Equational reasoning.** In mathematics, many *calculations* are part of a theory, especially in proofs. An example is:

  $ax^2 + bx + c = a(x^2 + \frac{b}{a}x) + c = a(x + \frac{b}{2a})^2 + c - \frac{b^2}{4a}$,

  of the form

  $expr_1 = expr_2 = expr_3$.

  This is a condensed notation for:

  $expr_1 = expr_2 \wedge expr_2 = expr_3$,

  with possibly also includes the implicit conclusion that

  $expr_1 = expr_3$.

  Equational reasoning is required in many other instances, for example in using the rule

  $x + 0 = x$

  to establish that

  $(x + y + 0) \times z = (x + y) \times z$.

  Reasonings like this one should be expressible in the chosen Type Theory, preferably without too many technical complications.

  An alternative could be to call upon an external Computer Algebra system. Such a system must then be fully reliable, otherwise there must yet come a translation of the 'computer algebra calculation' into a series of judgements in Type Theory.

- **Reasoning with a partial order.** Similar remarks can be made regarding partial orders, sometimes in connection with equational reasoning, e.g.:

  given that $t < q$ and $q = p$,

  infer that $t < p$.

- **Reflexivity, symmetry, transitivity.** A related problem is how the Type Theory deals with these three basic properties of relations. In practice, there is often an enormous amount of bothersome 'administration' involved in proofs depending on these notions. It would be nice to use a Type Theory which has a gentle 'sugaring' to deal with this problem.

- **Commutativity, associativity.** Similar problems are connected with operations having properties like these.

- **Basic constructs.** For basic constructs in mathematics, e.g. the set of reals, it is always the question whether we should actually *construct* them in the Type System, or introduce them *axiomatically*.

- **Structures.** Dependent structures like 'group', 'ring' or 'metric space' are tuples of sets and operations, together with some specific properties. In Type Theory (and in WTT) these become contexts. (For an example, see section 4.4.) Since a mathematician likes to consider these structures as *single* objects (G is a group, S is a metric space), there is a natural wish to make *abbreviations for contexts*. It is not yet quite clear how this can be done in a transparent, easily applicable way.

- **Partial functions.** Although *total* functions are fundamental in Type Theory, due to its lambda-calculus base, it is not immediately clear how *partial* functions can be treated in Type Theory. For a possible approach, see [Kuper 93].

- **Sets, subsets.** *Sets* are easily representable in Type Theory, viz. as inhabitants of the sort $*$ (or $*_s$), but it is not immediately clear how *subsets* can be treated. A possibility is to introduce a subset as the pair of a carrier set and a predicate on this set.

  Note that there is an intrinsic difference between $x \in V$ and $x : V$. In the first case, x is an element of V, but it can also be an element of many other sets. In particular, if $V \subseteq W$, then $x \in W$, as well. In the second case, however, x has type V *and only type* V (up to conversion). As a consequence, if $V \subseteq W$, then $x : W$ *only if* V *is definitionally equal to* W.

  Hence, the set-relation, and accordingly the subset-relation, have to be treated differently from the typing-relation. A possibility is to *define* sets and the subset-relation in Type Theory, together with the rules concerning elementhood. Another possibility is to introduce *subtyping* in one way or another, with as a consequence the loss of Uniqueness of Types. Several approaches have been investigated in the literature (see e.g. [Plotkin et al. 94]).

- **Coercion.** The notion of coercion is related to subtyping. Coercion is a frequently used device in mathematics. For example, a function $f : \mathbb{Z} \to \mathbb{Z}$ can be applied to a *natural* number n. Moreover, the result is an integer, but at the same time a *real* number. These so-called *coercions* (namely coercing n to be integer, as well, and fn to be real) can be incorporated in Type Theory, in different manners. For example, the subset-mechanism with its properties can be used for this purpose.[5]

  Coercions are also very convenient for using setoids. For example, let S be the setoid $\langle V, \sim \rangle$ of set V and equivalence relation $\sim$ on V. Then a coercion of a setoid S to its carrier set V has the effect that '$t \in S$', which is syntactically incorrect, is transformed into the correct '$t \in V$'. This is in accordance with existing habits in mathematics: one says 'x is an element of the group G', meaning that x is an element of the *carrier set* of the *structure* G.

- **Induction, recursion.** Both notions should have an explicit counterpart in the Type Theory. Again, one possibility is to internally define these notions in a Type Theory. Another one is to use *inductive types*. See [Paulin–Mohring 93] or [Coq 96].

- **Cartesian products or Σ-types.** Again, here is a choice to be made: define pairs and Cartesian products in the Type Theory, or use Σ-types.

- **Choice of the logic.** Type Theory usually is a logical *framework*, meaning that the logic can be chosen (constructive logic, classical logic, ...). This choice has to be made. In general, classical logic will be the obvious choice.

- **Proof irrelevance.** This is a well-known problem in Type Theory. For example, the natural logarithm ln is a partial function on $\mathbb{R}$. Hence, ln has *two* arguments: a real number r, *and a proof that this* r *is positive*. Proof irrelevance entails that the numeric outcome of applying ln to r does not depend on the actual proof p of $r > 0$.

5.2 Comparing WTT with Type Theory

Books in Weak Type Theory are *richer* than their translation into Type Theory on the one hand, and *poorer* on the other hand. Below we give a list of some of the relevant differences and we suggest what must be added or eliminated to a WTT-text (or to Type Theory) in order to enable a feasible and trustworthy translation 'WTT → Type Theory'.

- *Richer*
  - First of all, WTT has *constants*, contrary to many Type Theories. Above, in section 5.1.1, we suggested to extend Type Theory with constants.
  - WTT has more linguistic notions than Type Theory. For example, it has *nouns* and *adjectives*. Below (section 5.3.1) we discuss how these notions can be tailored to Type Theory.

[5] Note that in WTT there is no need for coercions, due to the weakness of the weak types: for example, the weak types of natural numbers, integers and reals are all the same, viz. `term`.

- WTT allows many *binders* in its grammar, whereas Type Theory has only a few (the λ, the Π and possibly the Σ.) We come back to this in section 5.3.2.
- WTT makes a clear distinction between *propositions* ('relational or logical statements') and *sets*. This difference is not present in the simplest forms of Type Theory, but it can easily be added. The most common way to do this is to split sort $*$ into the sorts $*_s$ (for sets) and $*_p$ (for propositions). So $S : *_s$ expresses that S is a set and $P : *_p$ says that P is a proposition. Then $a : S$ expresses that a is element of S. Via the propositions-as-types isomorphism we can now interpret $u : P$ as the fact that term u 'proves' proposition P.
- *Definitions* are first class citizens in WTT. In the previous section we discussed the importance of definitions (in relation with constants) and we proposed to adapt Type Theory in order to allow definitions. However, WTT has different kinds of definitions, one for each of five linguistic categories (see section 2.8). These have to be streamlined a bit, in order to guarantee a smooth transition to Type Theory. We discuss this in section 5.3.3.
- We also pleaded in the previous section for a Type Theory with the notion *book* rather than *judgement* as the basic entity.

- *Poorer*
  - A WTT-book misses the *argumentation structure* of a mathematical exposition, which is also essential for Type Theory (but not explicitly expressed there either). This has to be repaired in the transition from WTT to Type Theory. We come back to this in section 5.3.4.
  - Related to this is the fact that Type Theory heavily depends on so-called *proof objects*, i.e. inhabitants of propositions which embody a formal version of the proof of the proposition. However, in WTT there are no proof objects at all. More than that, the notion 'inhabitant of a proposition' is fully absent in WTT. This has to be mended as well in the translation.
  - In particular, all *assumptions* in a WTT-book must obtain a variable as proof object. That is, if S is an assumption occurring in a context Γ, then it has to be replaced by the *declaration* $x : S$ for some fresh variable x. This has consequences for the parameter lists of defined constants, which now have to be extended with the new variables, and also for each place where a definition is used. (The length of the parameter list of a defined constant is now equal to the 'length' of the context of the definition.)
  - A book usually contains only the *main steps* in a reasoning. That is, many intermediate steps are skipped and left to the understanding reader. In translating WTT into Type Theory, however, all these intermediate steps have to be added in some way or another. There are several strategic possibilities, for example: first extend the WTT-text to a more complete WTT-text and then translate, or translate directly and doing the extension/completion while translating.

- The linguistic requirements of WTT are much too weak for establishing the *correctness* of a book. Hence, in the transition to Type Theory we have to add many arguments in order to let the translation agree with the strict rules of Type Theory. (See also section 5.3.4.) For example, parameter lists of constants must obey rather complicated conditions (see the previous section). Hence, each 'use' of a definition brings along a set of *proof obligations*, which all have to be elaborated.
- The lastmentioned observation holds in particular for *function application*. Note that functions and application are fundamental notions in Type Theory (which is no surprise, since it is based on lambda calculus), in WTT they are much less prominent. As a consequence, application of a function to an argument is only linguistically restricted (both the function and the argument must be *terms*). In the transition to Type Theory, it must be checked whether the argument 'inhabits' the domain of the function. This, again, is a proof obligation.
- There are many other proof obligations when translating a WTT-text into Type Theory. This is not only the case when verifying the existence of a limit or the existence of a unique element ($\iota_{x \in V} \ldots$), but also in many, many cases where a mathematics writer jumps over details or counts on the understanding reader, whereas Type Theory forbids all unchecked transitions[6].
- Filling in all the necessary steps omitted in a WTT-book, can have the effect that the number of lines 'explodes'. However, work in the Automath-project showed that the 'loss factor' is rather constant, 'something like 10 or 20' according to de Bruijn (see [Nederpelt, Geuvers, de Vrijer 94], p. 160).

Note that some of the notions which make WTT 'richer', in comparison with Type Theory, ask for *an extension of Type Theory* (e.g. with constants and definitions). On the other hand, there are also notions in the class 'richer' which can better be *eliminated from WTT-texts*, before translating into Type Theory (e.g. nouns, adjectives, different formats for definitions). The lastmentioned notions are introduced in WTT in order to do the 'working mathematician' a favor, since WTT thus stays close to the linguistic habits of humans-doing-mathematics. For Type Theory, however, these notions are superfluous.

For those aspects in which WTT is 'poorer' than Type Theory, there is however only one solution: extend a WTT-book in a suitable manner in order to transform it into a Type Theory representation.

5.3 Translating a WTT-book into Type Theory

Below we give more details about some of the adaptations needed to translate a WTT-book into Type Theory.

5.3.1 Nouns and adjectives

In WTT, nouns and sets play comparable roles. It is easy to switch between the two categories, as section 2.4.2 shows: for each noun n there is a corresponding set $n\uparrow$, and for each set S

[6] But for the basic relation of β-reduction.

there is a corresponding noun $S\downarrow$.

We assume that, as is now always the case, Type Theory is not extended with nouns and adjectives (it could be a challenge, however, to invent such a Type Theory!).

Therefore, in the transition to Type Theory, we replace each noun by a set. E.g., we translate the noun 'a prime' into 'the set of all primes', say 'Pr', and 'an edge of d' in 'the set of all edges of d', say '$Edge{-}set(d)$'. The noun 'a set' gives no problems, in this respect: its translation is SET , 'the class of all sets'.[7]

Adjectives of WTT can be rephrased as *predicates*. For example, the adjective 'Abelian', defined for groups, can be translated into the predicate type Ab, with e.g. the following definition:

$Ab = \lambda_{\langle G,\cdot,e,^{-1}\rangle\,:\,\text{Structure}-\text{of}-\text{groups}}(\forall_{x,y:G}(x\cdot y = y\cdot x))$.[8]

(The type of Ab is $\Pi_{G:\text{Structure}-\text{of}-\text{groups}}(*_p)$ – a so-called Π-type – with $*_p$ the type of all propositions. The Π-type mentioned is *the type of all predicates over the class of groups*.)

5.3.2 Binders

We look at a number of binders in WTT and their translation into Type Theory form:

- The λ-binder stays as it is, since λ is also a primitive of Type Theory.
- Many common binders in mathematics can be reformulated as *constants*. We already gave an example in section 4.3, where the $\lim_{h\to p}$–binder was transformed into the constant lim.
- The ι-binder can also be rewritten as a constant, acting on a set (or a class of sets). Examples:

  $\iota_{n\in\mathbb{N}}(2<n<\pi) \mapsto iota(\{n:\mathbb{N}|2<n<\pi\})$,

  $\iota_{U\in\,\text{SET}}\ (3\in U\wedge|U|=1)\mapsto iota(\{U:\ \text{SET}\ |\,3\in U\wedge|U|=1\})$.
- Another example concerns the $\bigcup$-binder:

  $\bigcup_{n\in\mathbb{N}}(V_n) = union\{W:\ \text{SET}\ |\,\exists_{n\in\mathbb{N}}(W=V_n)\}$.
- In its turn, the set-binder $\{\ldots|\ldots\}$ can be omitted by using either the notion of *subset* or that of *predicate*. In the first case we make an appeal to a subset-notion being incorporated in the chosen Type Theory. In the second case we identify a predicate on A with the subset of all elements of A for which the predicate holds. (See also section 5.1.3.)
- The `Noun`-binder should be replaced by the set-binder (and transformed as discussed in section 5.3.1), the `Adj`-binder finds a counterpart in the predicate-binder, as expected.

[7] We use the word 'class' here, because 'the set of all sets' reminds too much of Russell's paradox. This paradox, however, is not applicable in Type Theory, due to the strict hierarchy of types.

[8] Here we consider a group to be a structure (a 4-tuple). See section 5.1.3 for this notion of 'structure'.

- The `Abst`-binder should be removed, as explained in section 2.5.4, Notes. That is, first `Abst` must be translated into a combination of `Noun` and $\exists$ (see the examples in that section), followed by a translation of `Noun` into the set-binder, as discussed above.

- The $\forall$-binder finds it natural counterpart in the Π-binder of Type Theory, the $\exists$-binder can either be represented as $\neg\forall\neg$, or be transformed into the Σ-binder of Type Theory.

5.3.3 Definitions

In WTT we have five different formats for definitional lines (see sections 2.8.1 and 2.8.2).[9] In Type Theory, however, we need only one:

$\Gamma \triangleright c(x_1, \ldots, x_n) := M : N,$

expressing that, in context Γ, constant c is defined as M of type N. The parameter list $(x_1, \ldots, x_n)$ of c can be omitted, as we explained before, since it is equal to the list of the subject variables of Γ and can therefore be reconstructed.

It will be obvious how all five definitions of section 2.8 fit in the above format. As explained in section 5.3.1, nouns are transformed into sets and adjectives into predicates. We give two examples:

a divisor of $n := \mathtt{Noun}_{k\in\mathbb{N}}(\exists_{l\in\mathbb{N}}(k \cdot l = n)) \mapsto$
$n : \mathbb{N} \triangleright$ Divisor$-$set$(n) := \{k : \mathbb{N} \mid \exists_{l:\mathbb{N}}(k \cdot l = n)\} : *_s,$

positive $:= \mathtt{Adj}_{x\in\mathbb{R}}(x > 0) \mapsto$
Positivity $:= \lambda_{x:\mathbb{R}}(x > 0) : \mathbb{R} \to *_p.$

5.3.4 Adding arguments

It is very usual to give 'clues' in mathematical texts about the coherence of that text. For example, one may find metastatements like:

Applying Theorem 5.6, ...
Since f is continuous, ...
Hence, ...
Following Definition 12, ...
Combining these results, we obtain ...

In WTT-books as we presented them in chapters 2 and 3, however, we omitted all indications in the original mathematical text about the *structure* of that text. Hence, in the translation of WTT into Type Theory, many of these motivations must be 'reconstructed'. In Remark 3.2.6 we suggested to save as many information as possible in two columns with 'labels' and 'comments'. This can be very helpful in the translation, since the human making the translation (with or without computer assistance) can use these clues as guidelines. As a by-effect, it can make the WTT-text more understandable for a reader who has not the original text at his disposal.

[9] We recall that this are the definitions of the form $\mathtt{C}^{\mathtt{t}/\sigma/\nu/\alpha/S}(\vec{\mathtt{V}}) := \mathtt{t}/\sigma/\nu/\alpha/S$.

5.3.5 The preface

In Type Theory, there is no such a thing as a preface. The constants occurring in a WTT-book which are external to this book and accounted for in the preface, can possibly be translated as *primitive* constants in Type Theory, that is as constants of a certain type with empty content. So, instead of the line

$\Gamma \triangleright a := M : N$,

we have now

$\Gamma \triangleright a :=$ `primitive` $: N$.

This will sometimes do. However, the unfolding of the definition of a has now become impossible. Hence, in many cases all relevant facts about a have to be given explicitly in Type Theory in order to make the translation 'faithful'.

5.4 Example of a translation from WTT into Type Theory

In order to show how a translation from WTT into Type Theory could work out, we revisit Example 4 (section 4.4). We only translate part of the text given in that section. This is enough to demonstrate that things soon become rather complicated, but we hope to transfer the feeling that such a translation is nevertheless feasible.

We assume that a form of Type Theory is chosen which allows books (i.e. sequences of judgements or 'lines') and definitions. We do not specify exactly which Type Theory is the basis of our translation below, but use only a kind of Type Theory *format*. This suffices, in our opinion, to give the reader some insight in the process of translation from WTT into Type Theory. We give six example Type Theory lines, with comments.

(1) We start with a definition of 'surjectivity' in Type Theory:
$A : *_s, B : *_s, f : A \to B \ \triangleright\ \mathrm{surj}(A, B, f) := \Pi_{b:B} \exists_{a:A} \mathrm{IS}\ B\ (fa)\ b$
Comments:
– This definition cannot be found in the original text, it is however necessary for establishing the correctness of the formalized proof. Note that for the derivation system of WTT, it is sufficient to include the adjective 'surjective' in the preface, with weak in- and out-types only. But for Type Theory, the full definition is required, because in each use of the defined notion, there are many details which have to be checked.
– As before, we borrow the symbol $\triangleright$ from WTT, where Type Theory judgements usually put the symbol $\vdash$.
– The symbol Π from Type Theory can be read as $\forall$ in the above case.
– The symbol $\exists$ is not a basic symbol of Type Theory. It has to be defined with the appropriate axioms or rules, or constructively as $\neg\forall\neg$.
– The symbol IS stands for equality, IS B stands for equality on B. Equality and its properties (reflexivity, symmetry, transitivity, Leibniz[10]) have to be formalized beforehand (see section 5.1.3).

[10]Instead of 'Leibniz' one also says 'replacement of equals by equals'. The Leibniz rule is: If $p = q$ then $\ldots p \ldots = \ldots q \ldots$, so the replacement of subexpression p by its equal q in a formula Φ, results in a formula equal to Φ.

(2) Next, we add the definition of 'homomorphism' to the Type Theory book:

$A : *_s, m_1 : A \to (A \to A), B : *_s, m_2 : B \to (B \to B), f : A \to B \triangleright$

homomorphism $(A, m_1, B, m_2, f) := \Pi_{a:A}\Pi_{b:B}$ IS $B(f(m_1ab))(m_2(fa)(fb))$

Comments:

– The binary operations m_1 and m_2 are given here in their *Curry-ed* versions: they take arguments one by one, subsequently, and not as a pair. Another option is to use Cartesian products (see also section 5.1.3).

(3) Next, we give a Type Theory version of the first line in the WTT-book of Example 4, the definition '$e' := \varphi(e)$' (line (9) in the flag-structured version):

$G : *_s, H : *_s, \varphi : G \to H, e : G \ \triangleright$ image $(G, H, \varphi, e) := \varphi e$

Comments:

– We embedded the definition here in the smallest possible context. (In the WTT-example of section 4.4 the context was larger than necessary; cf. what we said about line (9) in that section.)

– It turns out that the translation of this definition into Type Theory makes things more complicated than desired! Instead of the short formula φe, we introduce here the 'abbreviation' (!) image (G, H, φ, e). Of course, an easy way out is to omit this definition altogether and to use φe for every occurrence of e' in the WTT-text. This would be a sensible intervention of the translator.

Here, however, we want to keep close to the WTT-text in order to show what happens. Hence, we keep this definition of 'image'.

The problems mentioned arise of course from the fact that in the original WTT-text, the author had a *didactic* purpose with the definition $e' := \varphi(e)$: the use of the primed symbol e' *suggests* the reader already that the φ-image of e will be the wanted left unit element of H (as it indeed turns out to be). In Type Theory, however, the emphasis is on correctness, not on didacticism. Therefore, a faithful translation into Type Theory can sometimes have unexpected consequences.

(4) The following line gives a Type Theory translation of the second line of the WTT-book, given in line (11) of the flag-version.

$G : *_s, H : *_s, \varphi : G \to H, p :$ surj $(G, H, \varphi), h : H \ \triangleright \ ph : \exists_{g:G}$ IS $H\ (\varphi g)\ h$

Comments:

– Again, we need a smaller context here than in the WTT-text. Moreover, we split the assumption 'φ : a surjective homomorphism' into two parts: 'φ : a surjection' and 'φ : homomorphism', from which we here only need the first one.

– Note that an assumption in the context is treated on a par with a declaration in Type Theory: the assumption itself is that 'surj (G, H, φ)' holds, which is expressed as the *declaration* '$p :$ surj (G, H, φ)'. The term p is a variable '*inhabiting*' surj (G, H, φ), with intended meaning: let p be a *proof* of the assumption.

– In the same manner, the term ph is an inhabitant of $\exists_{g:G}$ IS $H\ (\varphi g)\ h$, hence with the intention of being a *proof* of that proposition. The fact that the relation $ph : \exists_{g:G}$ IS $H\ (\varphi g)\ h$ does indeed hold in the given context, is a consequence of the rules of Type Theory, which we do not discuss here.

(5) The first equality in the third line of the WTT-book of section 4.4 ($e' * h = \varphi(e) * \varphi(g)$, see also line (14)) becomes in Type Theory:

$G : *_s, e : G, H : *_s, m_2 : H \to (H \to H), \varphi : G \to H, h : H, g : G, u : \text{IS } H\ (\varphi g)\ h \triangleright$
$\quad \ldots : \text{IS } H\ (m_2(\text{image } (G, H, \varphi, e))h)\ (m_2(\varphi e)(\varphi g))$

Comments:

– Again, the context is shorter than in the WTT-version.

– We do not give the proof object, since this involves Leibniz ($\varphi g = h \Rightarrow m_2(\varphi e)\varphi g = m_2(\varphi e)h$) and the formalization of Leibniz is rather complicated.

– Also, the definitional equality of image (G, H, φ, e) and φe plays a role. It depends on the kind of Type Theory chosen, whether such a definitional equality must be determined with a separate proof object, or that it is a consequence of an internal reduction ('definitional reduction'), comparable to β-reduction, for which no explicit justification is needed.

(6) We conclude with the second equality in the third line of the WTT-book, $\varphi(e) * \varphi(g) = \varphi(e \cdot g)$. This becomes:

$G : *_s, m_1 : G \to (G \to G), e : G, H : *_s, m_2 : H \to (H \to H), \varphi : G \to H,$
$\quad q : \text{homomorphism } (G, m_1, H, m_2, \varphi), h : H, g : G, u : \text{IS } H\ (\varphi g)\ h \triangleright$
$\quad\quad \text{symm } (H, \varphi(m_1 eg), (m_2(\varphi e)(\varphi g)), qeg) : \text{IS } H\ (m_2(\varphi e)(\varphi g))\ \varphi(m_1 eg)$

Comments:

– Here we need the second part only of the assumption that φ is a surjective homomorphism: $q : \text{homomorphism } (G, m_1, H, m_2, \varphi)$.

– The proof object qeg proves that IS $H\ \varphi(m_1 eg)\ (m_2(\varphi e)(\varphi g))$. Here we need a proof for its symmetric variant, so with interchanged second and third argument. Therefore we use the axiomatic property *symmetry* for equality, with definition:

$A : *_s, m : A, n : A, v : \text{IS } A\ m\ n \triangleright \text{ symm } (A, m, n, v) := \texttt{primitive} : \text{IS } A\ n\ m$

We stop here with the translation of Example 4, expecting that we have given the reader some idea about what Type Theory is and how a WTT-text can be translated into Type Theory. We also tried to give an impression about how to solve some of the problems mentioned in earlier sections of this chapter.

Chapter 6

Final remarks

In this chapter, we compare our work with that of others and we discuss future work on the subject.

6.1 Comparison with other work

In the past, formal languages for mathematics ('mathematical vernaculars', 'specification languages for mathematics', cf. Section 1.4) have often been developed in relation to *theorem checkers* and *theorem provers*. Below we compare our WTT with a number of mathematical vernaculars as have been developed in the past three decades. We subsequently discuss: Automath-related systems, mathematical vernaculars behind theorem provers and stand-alone systems.

6.1.1 Automath-related mathematical vernaculars

- A direct source of inspiration for WTT was the mathematical language WOT ('Wiskundige OmgangsTaal', i.e. mathematical everyday language), [De Bruijn 79], which has been devised by the international pioneer in Theorem Proving, N.G. de Bruijn (Eindhoven University, the Netherlands). The description of WOT was general and only partially formalized. The language WTT is based on ideas employed in WOT, but it is independent and worked out in details. As we have shown in the previous chapters, WTT has a syntax and a derivation systems obeying all reasonable requirements and is therefore fully formalized, in contrast with WOT.

- De Bruijn's languages SEMIPAL and PAL [De Bruijn 70] are intended for the representation of the 'administrative structure' of mathematical texts. In both languages one can account for contexts, parameter lists and variables (in PAL there are also types). However, these languages are by nature insufficient to represent mathematical contents and they miss therefore the expressivity necessary to comply with the safety criterion (see Section 1.2).

- Another invention of de Bruijn was the *mathematical vernacular* MV[1] [De Bruijn 87].

[1] In fact, de Bruijn was the one to coin the expression 'mathematical vernacular' which is now generally

In MV, however, many choices of a logical and mathematical character have already been made, which WTT still postpones. Moreover, MV incorporates certain correctness requirements, there is for example a hierarchy of types corresponding with sets and subsets. Therefore, MV is suited for a partial formalization of mathematical content, but it is already 'on its way' to a full formalization. Hence, in our opinion, WTT is 'closer by' a given informal mathematical content than MV, so a WTT-like language obeys more to the safety criterion.

Apart from this, MV has a description which differs considerably from that of WTT, and MV has a syntax which is less compact and not as naturally confined as the syntax of WTT. On the other hand, it is worth while to investigate whether MV (or a variant thereof) is suited as a next stage of intermediate language in the direction of a full formalization.

- The 'mathematical language Automath' [De Bruijn 70] requires mathematical content to be completely formalized. It enables immediate 'theorem checking'. By its amount of details, it is far too complex to obey the safety criterion. However, it would be interesting to compare WTT with Automath, especially as regards a 'translation' in either of the directions. See also Section 6.2.

6.1.2 Mathematical vernaculars behind theorem provers

Presently, there is a great variety of theorem checkers and theorem provers. We mention Automath [De Bruijn 70], Mizar [Rudnicki 92], NuPrl [Constable et al. 86], Coq [Coq 96], Lego [Luo and Pollack 92], Alf [Altenkirch et al.], ΩMEGA [Benzmüller, Kohlhase et al. 97], PVS [Owre et al. 92] and HOL [Gordon and Melham 93]. When implemented, many of these systems provide help for the users, e.g. by offering a friendly user interface or by enabling the use of *tactics*. These tactics are special instructions for the computer, in order to simplify or develop the actual *proof goal* during the construction or check of a proof. Some of the mentioned systems are based on a fully formalized language for mathematics, such as Coq. In those systems, tactics are help routines for the user, but the final formalization is in the underlying language. This is not the case with other systems like PVS, where a completed proof is not a text with mathematical content, but only a listing of the tactics used.

A number of the theorem provers mentioned offer a mathematical vernacular in the WTT-sense, i.e. an incomplete, textual representation of a mathematical content which has resemblance with an informal exposition. Below we discuss some of these.

- The theorem proving system Coq (the 'Calculus of Constructions', see [Coq 96]) incorporates in its documentation a kind of intermediate specification language, called *Gallina* ([Gallina]). This 'vernacular' is a formally defined language meant for the development of mathematical theories and to prove specifications of programs. The intention is that it is usable as a 'language of commands' for Coq, helping the Coq-user to stay closer to 'normal' intuition when proving a mathematical proposition. It is, however, not meant as 'a first step in formalization' as WTT is, it has a rather specific

used for all kinds of 'languages for mathematics'.

form which does only distantly reflect a mathematical discourse in CML and it is also not very adequate for the purposes exposed in this report.

- The mathematical vernacular proposed in [Dowek 90a] relies on the Calculus of Constructions (Coq). It provides instructive 'labels' (like `Axiom`, `Definition`, `Hypothesis`, `Statement`, `Proof`). Moreover, a number of natural deduction rules are replaced with more intuitive alternatives (e.g., $\exists$-elimination). The aims are clearly different from ours: labels as the ones above are out of scope in our formalization and so are *comments* about the (e.g. logical) structure, as is discussed in Remark 3.2.6. Moreover, since WTT is not concerned with validity, the proof structure itself (e.g., natural deduction) has no formal counterpart in our system.

  Consequently, a Proof Synthesis Algorithm as in [Dowek 90b] has no direct application in WTT.

  However, when WTT is translated into more complete formal languages (see Section 6.2), the ideas of Dowek will most probably be fruitful and inspiring.

- The aim of the project ΩMEGA (see for example [Benzmüller, Kohlhase et al. 97] or [Siekmann and Hess 2001]) is to develop a software environment for the support of a scala of theorem provers. It is geared towards an expert user, and not a typical mathematician. The built-in version of the 'mathematical vernacular' is meant to give the user on request a mathematics-like computer 'view' of an already checked proof. It has the same drawbacks as the mathematical vernacular of Coq.

- The basic language of Mizar [Rudnicki 92] is near to the safety criterion and it has proven to be suited for expressing large corpora of mathematical content. Its syntax is, however, rather complicated and it requires therefore much of an ordinary user to become acquainted with it.

- As far as we know, none of the other theorem provers provides an independent language for describing mathematical content in such a manner that the safety criterion is sufficiently accounted for. The mathematical vernaculars as presented in those systems are moreover (to our knowledge) not ready for immediate use, or even difficult accessible for a mathematical user. (For an extensive overview of 'systems implementing mathematics in the computer', see [Wiedijk].)

6.1.3 Other mathematical vernaculars

We finally mention a number of attempts in the direction of a 'mathematical vernacular' with no direct connection to a theorem prover.

- In 'Towards a Formal Mathematical Vernacular' [Groote 92] *proof constellations* are defined, with tactics as mappings from one constellation to another. The presented ideas are meant for a manner of formalized proof construction which is closer than usual to 'normal' mathematical behaviour. In some sense this is indeed the case, albeit that the language presented still has a strong affinity with full-fledged type theory and that it does not come very close to the original mathematics text.

- The project 'Implementing a Mathematical Vernacular' [Luo and Callaghan 99] is still in development. It is based on constructive Type Theory and aims at a much richer structure than WTT does. For example, it must contain formal proofs and proof terms, which are absent in WTT. A formal report on the syntax has not been published. See also [Callaghan and Luo 98] about the Durham Mathematical Vernacular Project.

  The same group of researchers is presently working on a new theorem prover (called 'Plastic') and an associated type-theoretic framework approach to interactive theorem proving (see [Callaghan and Luo 2001] and [Callaghan et al. 2001]). Again, the objectives are on the theorem proving side, not on the purely linguistic side of the spectrum (as WTT is).

- In the *Theorema project* ([Buchberger 97]) existing computer algebra systems are extended with facilities for mathematical proving. The provers are designed to imitate the proof style humans employ in their proving attempts. The proofs can be produced in human-readable style. However, this is done by *post-processing* a formal proof in natural language. This deviates from our approach, for which no formal proof needs to be present. The natural language part of Theorema has little structure and restricts itself to comments on the logical steps employed, which is a part that does not appear in WTT. On the other hand, the linguistic facilities of WTT are absent in Theorema's natural language.

  Moreover, the text style of Theorema (insofar as we could see) is not based on a formal grammar for the textual language.

However, the ideas exposed in the projects above are worth while to be studied when enriching WTT in the direction of a full formalization.

6.2 Future work

We list a number of items concerning possible future work.

- First of all, the syntax and derivation rules of WTT must be *tested* on a corpus of mathematical texts which has considerable size and comes from various areas in mathematics. Former tests, though varied and carried out by many generations of students (cf. Chapter 1), are not extensive enough to allow definite conclusions about WTT in its present shape. The forthcoming tests should make appeal to a potential *users group*, ranging from students to all sorts of mathematicians, both in the theoretical and the applied field, and working in either teaching or research. This may lead to an adapted form of WTT.

- In this testing stage, it should be considered whether certain forms of *sugaring* can safely be added to WTT. This sugaring is probably desirable in order to make WTT an acceptable language tool for the users, also in the experimental phase. For example, the possibility of infix notation makes the text considerably better writable and readable for humans. This holds to a lesser degree for other sugaring devices, such as the possibility

of omitting empty parameter lists. Other examples are given earlier in this text; as said before, in this report we deviate several times from the official syntax as given in Chapter 2, only for reasons of readability (the law-abiding reader was supposed to undo these acts of sugaring 'on the fly', whenever they occurred).

Another possible sugaring which may be advantageous is the use of 'flags' as employed in Chapter 4. In the examples of that chapter, the benefits of the flag notation were explained.

For enabling a consistent use of certain forms of sugaring, the syntax of WTT must be extended. It may be preferable to do this in a kind of 'shell', built on top of WTT, in order to maintain the safety criterion for WTT itself. The development of such a sugaring-shell is user-driven and will probably ask for rather arbitrary decisions, without much coherence. In this respect, the sugaring-shell differs considerably from the underlying WTT syntax, which has – the author believes – a tighter structure and a more mathematically-driven motivation.

- Next, it is desirable to *implement* WTT. A parser should be able to parse WTT-expressions and to check whether such expressions obey to the grammar of Chapter 2. If there is a syntax shell containing the sugaring possibilities, this should be considered, as well.

  Moreover, a (grammatical) type checker must be implemented which establishes the weakly well-typedness of contexts, books etc. according to the derivation rules of Chapter 3. User-friendliness of such a type checker is of importance, in order to make the tool acceptable for mathematicians who wish to write (or translate) their text in(to) WTT. This may ask for things as 'flags', but also for other aids such as pop-up windows, appropriate input-output handling, interactive communication, possible storage and retrieval of texts and contexts, et cetera.

- Another important feature is whether the WTT-syntax can be extended in such a manner that a form of *recursion* is possible, both for recursive definitions and for recursive functions. This may make WTT more powerful, just like the more recent forms of Coq with recursion are better usable than the original version without it (compare [Coq 96] and [Coquand and Huet 88]).

- Moreover, it is interesting to investigate how WTT can be *enriched* in the direction of either (some form of) Type Theory, or another acceptable complete mathematical language, such as that of Zermelo-Fraenkel Set Theory. One step in that direction is to incorporate labels, proofs and proof methods. This encapsulates the *label* and *comments* columns mentioned in Remark 3.2.6.

- It is conceivable that in this stage of the research, *a chain of intermediate languages* between WTT and – for example – full fledged Type Theory is the best manner to bridge the gap. In this case, translation protocols should be devised for each link of the chain. For different transitions in this translation process, different specialists may be the preferred executives: mathematicians, computer scientists, type theorists.

It may be worth while to compare this research with work on (hierarchies of) specification languages as has been done in computer science. Also, work on open terms in proofs (cf. [Jojgov 2001]) may be helpful.

- In this process, it should also be investigated to what extent *computer assistance* can be invoked in either the full transition process or in one of the translation stages, from WTT to a completely formalized version of a mathematical text.

- Finally, it can be investigated how the results obtained can be made profitable for the community of mathematicians, both in developing and in using mathematics, in several degrees of precision. For example, easy access to WTT technology can be useful for computer help in writing mathematics. On the other side of the spectrum, we have the complete formalization of a certain text, which is suitable for a complete check on correctness and subsequently for storage of correct mathematical knowledge in a – publicly accessible – data base. We mentioned still more possibilities in the introducing Chapter 1.

Acknowledgements

The author thanks the following persons for their valuable comments on the subject and on the text in an earlier stage: Roel Bloo, Tijn Borghuis, Georgi Jojgov, Fairouz Kamareddine, Roel Körvers, Mark Scheffer and Freek Wiedijk. The idea for adding a derivation system to the syntax of WTT is due to Henk Barendregt, the author thanks him for this suggestion. Fairouz Kamareddine repaired several errors in the theorems and lemmas of section 3.11. She also delivered the full proofs of all theorems and lemma's in that section (these are worked out in [Kamareddine and Nederpelt 2001]), for which the author is very grateful.

Bibliography

[Altenkirch et al.] T. Altenkirch et al., *A user's guide to ALF*, Department of Computing Science, University of Göteborg / Chalmers University of Technology.

See `http://www.cs.chalmers.se/Cs/Research/Logic/alf/guide.html`.

[Barendregt 92] H.P. Barendregt, *Lambda calculi with types*, in Handbook of Logic in Computer Science, S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, eds., vol. 2, Oxford University Press, 1992, ch. 2, pp. 117–309.

[Barendregt and Geuvers 2001] Henk Barendregt and Herman Geuvers, *Proof-assistants using Dependent Type Systems*, in: Handbook of Automated Reasoning, Alan Robinson and Andrei Voronkov, eds, 2001, Elsevier Science Publishers, pp. 1149–1238.

[Benzmüller, Kohlhase et al. 97] Christoph Benzmüller, Michael Kohlhase et al., *ΩMEGA: Towards a Mathematical Assistent*, in: Proceedings of Conference on Automated Deduction (CADE-14), W. McCune, ed., vol. 1249 of Lecture Notes in Artificial Intelligence, Townsville, Australia, 1997, Springer, pp. 252–255.

[Bloo et al. 2002] Roel Bloo, Fairouz Kamareddine, Twan Laan and Rob Nederpelt: *Parameters in Pure Type Systems*, in: Proceedings of Latin02, Cancun, Mexico, Lecture Notes in Computer Science 2286, Springer Verlag, 2002, pp. 371–385.

[De Bruijn 70] N.G. De Bruijn, *The mathematical language AUTOMATH, its usage and some extensions*, in: Symposium on Automatic Demonstration, IRIA, Versailles (1968), Lecture Notes in Mathematics 125, Berlin, Springer Verlag, 1970, pp. 29–61. Also in [Nederpelt, Geuvers, de Vrijer 94], pp. 73–100.

[De Bruijn 79] N.G. de Bruijn, *Grammatica van WOT*, Euclides 55, 1979/1980, pp. 66–72.

[De Bruijn 87] N.G. de Bruijn, *The Mathematical Vernacular, a language for mathematics with typed sets*, in Proceedings of the Workshop on Programming Languages, Marstrand, Sweden, 1987. Also in [Nederpelt, Geuvers, de Vrijer 94], pp. 865–936.

[Buchberger 97] Bruno Buchberger et al., *A survey of the Theorema project*, in: Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation), Maui, Hawai (1997), ACM Press, 1997, pp. 384-391.

See also `http://www.theorema.org/Overview.html`.

[Callaghan and Luo 98] P.C. Callaghan and Z. Luo, *Mathematical vernacular in type theory based proof assistants*, UITP'98, 1998.

[Callaghan and Luo 2001] P.C. Callaghan and Z. Luo, *An implementation of LF with coercive subtyping and universes*, Journal of Automated Reasoning, 27 (1), 2001, pp. 3–27.

[Callaghan et al. 2001] P.C. Callaghan, Z. Luo and J. Pang, *Object languages in a type-theoretic meta-framework*, Workshop of Proof Transformation and Presentation and Proof Complexities, Siena, Italy, 2001.

[Constable et al. 86] R.L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice Hall, 1986.

[Coquand and Huet 88] T. Coquand and G. Huet, *The Calculus of Constructions*, Information and Computation 76, pp. 95–120.

[Coq 96] Coq, *The Coq Proof Assistant Reference Manual (version 6.1)*, INRIA, Rocquencourt and CNRS–ENS, Lyon, 1996.

[Dowek 90a] Gilles Dowek, *Naming and scoping in a mathematical vernacular*, Rapports de Recherche No. 1283, INRIA (Institut National de Recherche en Informatique et en Automatique, Rocquencourt, 1990.

[Dowek 90b] Gilles Dowek, *A Proof Synthesis Algorithm for a Mathematical Vernacular*, in: Proceedings of the First Workshop on Logical Frameworks, Antibes, France, G. Huet and G. Plotkin, eds., 1990.

[Eijck and Kamp 96] Jan van Eijck and Hans Kamp, *Representing Discourse in Context*, in: Handbook of Logic and Language, J. van Benthem and A. ter Meulen, eds., Amsterdam, Elsevier, 1996, pp. 179–237.

[Franssen 2000] Michael Franssen, *Cocktail: A Tool for Deriving Correct Programs*, PhD thesis, Eindhoven University of Technology, 2000.

[Gallina] See `http://coq.inria.fr/doc/node.0.0.html`.

[Geuvers et al. 2001] J.H. Geuvers et al., *The 'Fundamental Theorem of Algebra' Project*, FTA web page.

See `http://www.cs.kun.nl/gi/projects/fta/index.html`.

[Gordon and Melham 93] M.J.C. Gordon and T.F. Melham, *Introduction to HOL: A theorem proving environment for higher order logic*, Cambridge University Press, 1993.

See also
`http://www.cl.cam.ac.uk/Research/HVG/HOL/HOL.html#history`.

[Gries 81] David Gries, *The Science of Programming*, Springer, 1981.

[Groote 92] J. F. Groote, *Towards a formal mathematical vernacular*, Technical Report 84, Department of Philosophy, University of Utrecht, December 1992.

[Jojgov 2001] G.I. Jojgov, *Systems for Open Terms: An Overview*, CS-Report 01-03, Eindhoven University of Technology, Department of Mathematics and Computer Science, 2001.

[Kamareddine et al. 2001] F. Kamareddine, L. Laan and R.P. Nederpelt, *Refining the Barendregt cube using parameters*, Fifth International Symposium on Functional and Logic Programming, FLOPS 2001, Lecture Notes in Computer Science 2024, pp. 375–389, 2001.

[Kamareddine and Nederpelt 2001] Fairouz Kamareddine and Rob Nederpelt, *A derivation system for a formal language of mathematics*, submitted for publication.

See `http://www.win.tue.nl/~wsinrpn/publ1.htm` for a postscript-version.

[Kamp and Reyle 93] Hans Kamp and Uwe Reyle, *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, Dordrecht, The Netherlands, Kluwer Academic Publishers, 1993.

[Kuper 93] Jan Kuper, *An axiomatic theory for partial functions*, Information and Computation, 107(1), 1993, pp. 104–150.

[Laan 97] Twan Laan, *The Evolution of Type Theory in Logic and Mathematics*, PhD thesis, Eindhoven University of Technology, 1997.

[Luo and Callaghan 99] Zhaohui Luo and Paul Callaghan, *Mathematical Vernacular and Conceptual Well-formedness in Mathematical Language*, in: Proceedings of the 2nd International Conference on Logical Aspects of Computational Linguistics (LACL '97), Nancy, Lecture Notes in Computer Science 1582, 1999.

See also `http://www.dur.ac.uk/CARG/mv.html`

[Luo and Pollack 92] Z. Luo and R. Pollack, *LEGO Proof Development System: User's Manual*, Department of Computer Science, University of Edinburgh, 1992.

[Nederpelt, Geuvers, de Vrijer 94] R.P. Nederpelt, J.H. Geuvers and R.C. de Vrijer, eds., *Selected Papers on Automath*, vol. 133, Studies in Logic and the Foundations of Mathematics, Elsevier, 1994.

[Nederpelt and Kamareddine 200X] Rob Nederpelt and Fairouz Kamareddine, *An abstract syntax for a formal language of mathematics*, Proceedings Fourth International Tbilisi Symposium on Language, Logic and Computation, 2001, Borjomi, Georgia. To appear.

[Oostdijk 2001] M. Oostdijk, *Generation and presentation of formal mathematical documents*, PhD thesis, Eindhoven University of Technology, 2001.

[Owre et al. 92] S. Owre, J.M. Rushby and N. Shankar, *PVS: A prototype verification system*, in Lecture Notes in Artificial Intelligence, vol. 607, D. Kapur, ed., Springer Verlag, 1992, pp. 748–752.

[Paulin–Mohring 93] C. Paulin–Mohring, *Inductive definitions in the system Coq*, in Lecture Notes in Computer Science, Marc Bezem and Jan Friso Groote, eds., vol. 664, Springer Verlag, 1993, pp. 328–345.

[Plotkin et al. 94] G.D. Plotkin, M. Abadi, L. Cardelli, *Subtyping and Parametricity*, Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science, Paris, France, July 1994, IEEE Computer Science Press, pp. 310–319.

[Rudnicki 92] P. Rudnicki, *An overview of the MIZAR project.*, in Proceedings of the 1992 Workshop on Types for Proofs and Programs, Bå̊stad, B. Nordström, K. Petterson and G. Plotkin, eds., 1992, pp. 311–332.

See also: `http://www.mizar.org/`

[Siekmann and Hess 2001] Jörg Siekmann, Stephan Hess et al, $\mathcal{LOUI}$*: Lovely* Ω*MEGA User Interface*, Formal Aspects of Computing 3, 2001, pp. 1–17.

[Severi and Poll 94] P. Severi and E. Poll, *Pure type systems with definitions*, in: Proceedings of LFCS'94, editors A. Nerode and Yu. V. Matiyasevich, Lecture Notes in Computer Science 813, pp. 316–328, 1994.

[Wiedijk] `http://www.cs.kun.nl/ freek/digimath/bycategory.html`

[Zucker 77] J. Zucker, *Formalization of classical mathematics in Automath*, in: Colloque Internationale de Logique, Clermont-Ferrand, France, 1975 (Paris, CNRS), pp. 135–145. Also in [Nederpelt, Geuvers, de Vrijer 94], pp. 127–140.

Appendix A

List of abstract syntax rules

Metasymbols for the standard categories (p. 10)

level	category	symbol	representative
atomic level	*variables*	$\mathtt{V}$	x
	constants	$\mathtt{C}$	c
	binders	$\mathtt{B}$	b
phrase level	*terms*	$\mathtt{t}$	t
	sets	σ	s
	nouns	ν	n
	adjectives	α	a
sentence level	*statements*	$\mathcal{S}$	S
	definitions	$\mathcal{D}$	D
discourse level	*contexts*	$\mathbf{\Gamma}$	Γ
	lines	$\mathbf{l}$	l
	books	$\mathbf{B}$	B

- **variables** (p. 11)

  $\mathtt{V} \;=\; \mathtt{V}^{\mathtt{t}} \mid \mathtt{V}^{\sigma} \mid \mathtt{V}^{\mathcal{S}}$

- **constants** (p. 12)

  $\mathtt{C} \;=\; \mathtt{C}^{\mathtt{t}} \mid \mathtt{C}^{\sigma} \mid \mathtt{C}^{\nu} \mid \mathtt{C}^{\alpha} \mid \mathtt{C}^{\mathcal{S}}$

- **binders** (p. 14)

  $\mathtt{B} \;=\; \mathtt{B}^{\mathtt{t}} \mid \mathtt{B}^{\sigma} \mid \mathtt{B}^{\nu} \mid \mathtt{B}^{\alpha} \mid \mathtt{B}^{\mathcal{S}}$

- **terms** (p. 17)

  $\mathtt{t} \;=\; \mathtt{C}^{\mathtt{t}}(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^{\mathtt{t}}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\mathtt{t}}$

- **sets** (p. 17)

  $\sigma \;=\; \mathtt{C}^{\sigma}(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^{\sigma}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\sigma}$

- **nouns** (p. 17)

  $\nu \;=\; \mathtt{C}^{\nu}(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^{\nu}_{\mathcal{Z}}(\mathcal{E}) \mid \alpha\nu$

- **adjectives** (p. 17)

  $\alpha \;=\; \mathtt{C}^{\alpha}(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^{\alpha}_{\mathcal{Z}}(\mathcal{E})$

- **statements** (p. 17)

  $\mathcal{S} \;=\; \mathtt{C}^{\mathcal{S}}(\overrightarrow{\mathcal{P}}) \mid \mathtt{B}^{\mathcal{S}}_{\mathcal{Z}}(\mathcal{E}) \mid \mathtt{V}^{\mathcal{S}}$

- **definitions** (p. 18, 20)

  $\mathcal{D} \;=\; \mathtt{C}^{\mathtt{t}}(\overrightarrow{\mathtt{V}}) := \mathtt{t} \mid \mathtt{C}^{\sigma}(\overrightarrow{\mathtt{V}}) := \sigma \mid \mathtt{C}^{\nu}(\overrightarrow{\mathtt{V}}) := \nu \mid \mathtt{C}^{\alpha}(\overrightarrow{\mathtt{V}}) := \alpha \mid \mathtt{C}^{\mathcal{S}}(\overrightarrow{\mathtt{V}}) := \mathcal{S}$

- **contexts** (p. 20)

  $\mathbf{\Gamma} \;=\; \emptyset \mid \mathbf{\Gamma}, \mathcal{Z} \mid \mathbf{\Gamma}, \mathcal{S}$

- **lines** (p. 21)

  $\mathbf{l} \;=\; \mathbf{\Gamma} \triangleright \mathcal{S} \mid \mathbf{\Gamma} \triangleright \mathcal{D}$

- **books** (p. 21)

  $\mathbf{B} \;=\; \emptyset \mid \mathbf{B} \circ \mathbf{l}$

Other metasymbols (p. 11)

category	symbol	representative
expressions	$\mathcal{E}$	E
parameters	$\mathcal{P}$	P
typings	$\mathcal{T}$	T
declarations	$\mathcal{Z}$	Z

- **expressions** (p. 11)

  $\mathcal{E} \;=\; \mathtt{t} \mid \sigma \mid \nu \mid \mathcal{S}$

- **parameters** (p. 12)

  $\mathcal{P} \;=\; \mathtt{t} \mid \sigma \mid \mathcal{S}$

- **typings** (p. 17)

  $\mathcal{T} \;=\; \sigma : \textsc{set} \mid \mathcal{S} : \textsc{stat} \mid \mathtt{t} : \sigma \mid \mathtt{t} : \nu \mid \mathtt{t} : \alpha$

- **declarations** (p. 18)

  $\mathcal{Z} \;=\; \mathtt{V}^{\sigma} : \textsc{set} \mid \mathtt{V}^{\mathcal{S}} : \textsc{stat} \mid \mathtt{V}^{\mathtt{t}} : \sigma \mid \mathtt{V}^{\mathtt{t}} : \nu$

Appendix B

List of derivation rules

$OK(B;\Gamma) \quad \equiv \quad \vdash B : \texttt{book},\ B \vdash \Gamma : \texttt{cont}$ (p. 24)

- **variables** (p. 27)

$$\frac{OK(B,\Gamma),\quad x \in \mathtt{V}^{\mathtt{t}/\sigma},\quad x \in \texttt{decvar}(\Gamma)}{B;\Gamma \vdash x : \texttt{term/set}} \quad (var)$$

- **constants** (p. 27, 28)

$$\frac{\begin{array}{c} OK(B;\Gamma),\quad \Gamma' \triangleright D \ \in\ B, \\ \texttt{decvar}(\Gamma') = x_1,\ldots,x_n,\quad \texttt{defcons}(D) = c \in \mathtt{C}^{\mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}}, \\ B;\Gamma \vdash A_i : \mathbf{W} \text{ if } B;\Gamma' \vdash x_i : \mathbf{W}\ (i = 1,\ldots,n) \end{array}}{B;\Gamma\ \vdash\ c(A_1,\ldots,A_n) : \texttt{term/set/noun/adj/stat}} \quad (int{-}cons)$$

$$\frac{\begin{array}{c} OK(B,\Gamma),\quad c \text{ external to } B,\quad in(c) = (\kappa_1,\ldots,\kappa_n),\quad out(c) = \kappa, \\ B;\Gamma\ \vdash\ A_i : \kappa_i\quad (i = 1,\ldots,n) \end{array}}{B;\Gamma\ \vdash\ c(A_1,\ldots,A_n) : \kappa} \quad (ext{-}cons)$$

- **binders** (p. 29)

$$\frac{OK(B;\Gamma,Z),\quad b \in \mathtt{B},\quad in(b) = (\kappa_1),\quad out(b) = \kappa_2,\quad B;\Gamma,Z\ \vdash\ A : \kappa_1}{B;\Gamma\ \vdash\ b_Z(A) : \kappa_2} \quad (bind)$$

- **phrases** (p. 29)

$$\frac{OK(B;\Gamma),\quad B;\Gamma\ \vdash\ n : \texttt{noun},\quad B;\Gamma\ \vdash\ a : \texttt{adj}}{B;\Gamma\ \vdash\ an : \texttt{noun}} \quad (adj{-}noun)$$

- **definitions** (p. 30)

$$\frac{\begin{array}{c} OK(B;\Gamma),\quad B;\Gamma\ \vdash\ t/s/n/a/S : \texttt{term/set/noun/adj/stat}, \\ \texttt{decvar}(\Gamma) = x_1,\ldots,x_n,\quad c \in \mathcal{C}^{\mathtt{t}/\sigma/\nu/\alpha/\mathcal{S}},\quad c \notin \texttt{prefcons}(B) \cup \texttt{defcons}(B) \end{array}}{B;\Gamma\ \vdash\ c(x_1,\ldots,x_n) := t/s/n/a/S : \texttt{def}} \quad (int{-}def)$$

- **contexts** (p. 30)

$$\frac{B \texttt{ : book}}{B \ \vdash \ \emptyset \texttt{ : cont}} \quad (emp{-}cont)$$

$$\frac{OK(B;\Gamma),\ x \in \mathtt{V}^{\sigma/\mathcal{S}},\ x \notin \texttt{decvar}(\Gamma)}{B \ \vdash \ \Gamma, x : \ \text{SET} \ / \ \text{STAT} \ \texttt{ : cont}} \quad (set/stat{-}decl)$$

$$\frac{OK(B;\Gamma), \quad B;\Gamma \ \vdash \ s/n \texttt{ : set/noun}, \quad x \in \mathtt{V}^{\mathtt{t}}, \quad x \notin \texttt{decvar}(\Gamma)}{B \ \vdash \ \Gamma, x : s/n \ \texttt{ : cont}} \quad (term{-}decl)$$

$$\frac{OK(B;\Gamma), \quad B;\Gamma \ \vdash \ S \texttt{ : stat}}{B \ \vdash \ \Gamma, S \texttt{ : cont}} \quad (assump)$$

- **books** (p. 31)

$$\frac{}{\vdash \ \emptyset \texttt{ : book}} \quad (emp{-}book)$$

$$\frac{OK(B;\Gamma), \quad B;\Gamma \ \vdash \ S/D \texttt{ : stat/def}}{\vdash B \ \circ \ \Gamma \triangleright S/D \texttt{ : book}} \quad (book{-}ext)$$