Is ZF a hack?

Comparing the complexity of some (formalist interpretations of) foundational systems for mathematics

Freek Wiedijk

Department of Computer Science, University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract. This paper presents Automath encodings (which also are valid in LF/λP) of various kinds of foundations of mathematics. Then it compares these encodings according to their size, to find out which foundation is the simplest.
The systems analyzed in this way are two kinds of set theory (ZFC and NF), two systems based on Church's higher order logic (Isabelle/Pure and HOL), three kinds of type theory (the calculus of constructions, Luo's extended calculus of constructions, and Martin-Löf predicative type theory) and one foundation based on category theory.
The conclusions of this paper are that the simplest system is type theory (the calculus of constructions) but that type theories that know about serious mathematics are not simple at all. Set theory is one of the simpler systems too. Higher order logic is the simplest if one looks at the number of concepts (twenty-five) needed to explain the system. On the other side of the scale, category theory is relatively complex, as is Martin-Löf's type theory.

(This paper is on the web, with the full Automath sources of the contexts described in it, at ⟨`http://www.cs.kun.nl/~freek/zfc-etc/`⟩.)

1 Introduction

1.1 Problem

Some time ago Bob Solovay drew my attention to the writings on proof checking by Raph Levien. In one of his postings on the forum called *Advogato*, Raph had written that 'ZF is a hack'.[1] This statement was the reason for this paper.

[1] Actually his statement was not that strong. He had written, in diary entry ⟨`http://www.advogato.org/person/raph/diary.html?start=265`⟩:

> *I was talking* [...] *with Bram, and he called ZF set theory a 'hack.' I more or less agree, but playing with it in the context of Metamath has led me to appreciate how powerful a hack it is. With a small number of relatively simple axioms, it gets you a rich set of infinities, but avoids the particular ones that*

Because I, too, do believe that ZF is a bit of a hack. The aim of this paper is to give some quantitative information on 'how much' of a hack I consider ZF to be.

The whole point of Cantor's set theory is (using type theoretic terminology) that Set and Set→bool should be the same thing. And in ZF they are not. The first are the sets and the second are the (possibly proper) classes. Of course every set is a class, so the two kinds of objects are related, but the relation between the two is not really simple. This distinction is the first reason we might want to call ZF a 'hack'.

In a presentation of ZF there are two levels that one has to introduce: first there is the level of first order predicate logic and second there is the level of the ZF axioms. On both levels one finds very similar concepts (for instance $\exists$ on the logic level corresponds to $\bigcup$ on the set theory level), and it seems strange that these levels cannot be identified. This is different in the foundations of the HOL theorem prover [5]. In that system, formulas in the logic are just functions that map to the set `bool`. So this distinction between levels is not necessary in the case of HOL, there the logical operations are just set theoretic functions like all the others. This distinction is the second reason why we might want to call ZF a 'hack'.

The majority of the ZF axioms state that the set theoretic universe is closed under certain operations. Once Henk Barendregt asked me whether I knew the number of ZF axioms. When I admitted I did not know the exact number offhand, he wrote the six ZF operations for me on a napkin (see Fig. 1; so the answer to his question was 'there are eight ZF axioms.') He then struck out the $F``x$ replacement operation saying that 'the F in this operation stands for Fränkel. If you remove it from the list, you remove the F from ZF and you get Zermelo set theory.'

Set properties	2		ext, found
Set existence	6	8	
$\mathrm{Set}(\emptyset)$			
$\mathrm{Set}(\{x, y\})$			
$\mathrm{Set}(\bigcup x)$			
~~$\mathrm{Set}(F``x)$~~			
$\mathrm{Set}(\mathcal{P}x)$			
$\mathrm{Set}(\omega)$			

Fig. 1. Henk's napkin

If one looks at ZF like this, it seems to be about 'six concepts'. But of course that presupposes that the whole machinery of first order predicate logic (which

bring the whole formal system crumbling down. You get integers, reals, tuples, sequences (finite and infinite), transfinites, and functions from set to set. [...]

is needed to phrase the ZF axioms) is already 'given'. The question that this paper wants to answer is 'how many concepts are needed to tell *the whole story* of ZF, without sweeping anything under the rug?' (and the answer turns out to be *thirty-one*, as is shown in Section 3.1.)

1.2 Approach

ZF distinguishes between a logic level and a set level, while HOL does not. Therefore, one would expect the HOL system to be the simpler of the two (of course it is also less powerful mathematically.) To investigate this, I wrote an Automath context both for ZFC and for HOL Light (of the HOL systems this system has the cleanest 'kernel'.) To my surprise, although the HOL context turned out to have less 'primitive notions' than the ZFC context, it was bigger. So ZF is not that much of a hack after all!

Then I wanted to know how these two contexts compare to type theory. I did not expect to be able to represent the rules for the inductive types of Coq in Automath (there does not even seem to be a publication in which they are stated precisely.) Instead I wrote a context for the MLW^{ext} system, after a paper by Peter Aczel [1]. This context turned out to be much bigger than both the ZFC and the HOL context, in all respects.

Then I decided to extend my collection of Automath contexts to even more foundational systems, to turn it into a 'comparative review of foundations of mathematics.' The result is this paper.

To summarize the approach followed: for each foundations of mathematics I constructed a context in a logical framework. This context was intentionally kept as close as possible to some given presentation from the literature (in all cases I used a presentation of the system 'in a few pages'.) Then I computed some statistics of these contexts to compare the 'complexity' of the different systems.

One might expect that there is a lot of variation possible in the encoding of a foundational system as an Automath context. However, it was my experience that it does not feel like one has that many choices if one follows a given presentation. It feels like there is only one simple way to do it. So if it is claimed here that a system 'needs 25 primitive notions', I would be surprised if a variation on the encoding would differ more than only one or two from that.

For convenience I used Automath as a logical framework, but the contexts also are valid in the logical framework LF/λP. It is well-known that Automath can be used as a logical framework: N.G. de Bruijn called it the 'Automath restaurant'. He compares intuitionistic mathematics to kosher food: just like in a non-kosher restaurant one might be able to order kosher food, one can 'order' constructive mathematics in Automath by working in an appropriate context. This paper is like the menu of the Automath restaurant.

In the contexts from this paper we use higher order abstract syntax (HOAS). This means that bindings in the object logic are encoded by bindings in the framework. For this reason, we do not have to deal with variable names or de Bruijn indices. More importantly it means that we never will have to model the contexts of the object logic: the Automath contexts will take care of that for

us. This is the main respect in which the Automath rendering of our source texts deviates from the originals. Often there is quite some talk in these texts about contexts Γ, which is ignored completely in the Automath version. Another deviation caused by our use of HOAS is that *instantiation* often does not need a primitive notion, as the Automath binding will take care of that as well.

My personal philosophy of mathematics is that of 'full-blooded' Platonism (when I do not think too hard about it), or that of fictionalism (when I do). Of course as Mark Balaguer explains in [2] both are basically the same thing, and fictionalism is just a fancy name for formalism.

Another aspect of my philosophy of mathematics: I do not like constructive mathematics very much (apparently my taste runs towards Platonist fiction.) According to Bas Spitters this is strange. He claims that a formalist should not mind which formal system he is using. And in some sense he is right! When I look at the contexts from this paper, the Platonist one of ZFC does not seem very different to me, emotionally, from the constructivist one of $\mathrm{MLW}^{\mathrm{ext}}$. (Of course some constructivists probably will think that encoding the rules of constructive mathematics as a formalist system misses the point entirely in the first place.)

1.3 Related Work

Contexts for logical systems are the subject of the field of logical frameworks. See [11] for an overview of this field. Popular systems for formalizing logics in a logical framework are Twelf [12] and Isabelle [10].

1.4 Contribution

The contexts that we present here are nothing special, as far as modeling logics in a logical framework go. The main differences with other work are:

- the use of the Automath system as the logical framework;
- the attempt to model the contexts as closely as possible to a given informal presentation of the system;
- the comparison in a *quantitative* way, by collecting statistics about the contexts.

1.5 Outline

This paper is organized as follows. We start in the next section by presenting a small Automath context – for first order predicate logic – in full detail. The other contexts will not be given explicitly, but will just be summarized. For the ZFC and HOL contexts there will be a full list of the primitive notions in the context. For the other contexts there will be just a table of counts and a list of Automath types.

The contexts are grouped according to paradigm. The set theory contexts are in Section 3. The contexts based on higher order logic are in Section 4. The type theory contexts are in Section 5. The category theory context is in Section 6.

2 FOL: first order predicate logic

Here is an Automath context for first order logic:

---

the types

```
* prop : TYPE := PRIM
* [a:prop] proof : TYPE := PRIM
* term : TYPE := PRIM
```

first order formulas

```
* false : prop := PRIM
a * [b:prop] imp : prop := PRIM
* [p:[z,term]prop] for : prop := PRIM
* [x:term][y:term] eq : prop := PRIM
a * not : prop := imp(a,false)
```

natural deduction

```
b * [_1:[_,proof(a)]proof(b)] imp_intro : proof(imp(a,b)) := PRIM
b * [_1:proof(imp(a,b))][_2:proof(a)] imp_elim : proof(b) := PRIM
p * [_1:[z,term]proof(<z>p)] for_intro : proof(for(p)) := PRIM
p * [_1:proof(for(p))][z:term] for_elim : proof(<z>p) := PRIM
a * [_1:proof(not(not(a)))] classical : proof(a) := PRIM
x * eq_intro : proof(eq(x,x)) := PRIM
y * [q:[z,term]prop][_1:proof(eq(x,y))][_2:proof(<x>q)]
  eq_elim : proof(<y>q) := PRIM
```

(This context is part of the contexts for the foundations built on first order predicate logic. It is the start of the ZFC and NF set theories in Section 3, and – with minor modifications to get many-sorted logic – of McLarty's axiomatic treatment of category theory in Section 6.)

We will summarize Automath contexts like this one in two ways. First, we will present the counts of the primitive notions (those are the Automath notions of which the body of the definition is `PRIM`) in a table:

the types	3
first order formulas	4
natural deduction	7
total	14

Second, we will focus on the Automath `TYPE`s in the context (because they are the most interesting part of it.) We will show them in the following style:

```
       prop
[prop] proof
       term
```

This means that `prop` is a type without any arguments (it represents the first order formulas), that `proof` is a type that takes one argument of type `prop` (it represents the provability of its argument: the type is inhabited if and only if the formula is provable) and that `term` is also a type without arguments (it represents the first order terms).

3 Set theories

3.1 ZFC: Zermelo-Fränkel set theory with the Axiom of Choice

The ZFC context is based on Henk's napkin that was shown in Section 1.1. It contains thirty-two primitive notions (thirty-one for ZF plus one for AC):

first order logic

1.		propositions
2.		proofs
3.		sets
4.	$\bot$	false
5.	$\to$	implication
6.	$\forall$	universal quantifier
7.	$=$	equality
8.	$\in$	element predicate
9.	$\to_I$	implication introduction
10.	$\to_E$	implication elimination (modus ponens)
11.	$\forall_I$	universal quantifier introduction
12.	$\forall_E$	universal quantifier elimination
13.		classical logic (excluded middle)
14.		reflexitivity of $=$
15.		substitution property of $=$

definition by cases

16.		definition by cases
17.		definition by cases axiom

set theory

18.	$\emptyset$	empty set
19.	$\{x, y\}$	pair set
20.	$\bigcup x$	union
21.	$F\text{“}x$	replacement operation
22.	$\mathcal{P}x$	power set
23.	ω	infinity

the axioms
24. *extensionality*
25. *foundation*
26. *empty set axiom*
27. *pair set axiom*
28. *union axiom*
29. *replacement*
30. *power set axiom*
31. *axiom of infinity*

choice
32. AC *axiom of choice*

So these are the concepts that one has to explain to introduce ZFC. One might object to the duplicate appearance of the notions corresponding to Henk's six operations (items 18–23 and items 26–31). But I think it is fair to distinguish between the empty set (labeled '$\emptyset$') and the empty set axiom (labeled 'empty set axiom'). It is analogous to distinguishing between the logical operations as formula constructors, and their natural deduction rules as proof constructors.

The definition by cases construction is needed to be able to derive the comprehension axiom from the replacement axiom. One can shift things around a bit, for instance one can have the comprehension axiom as primitive notion and then derive the definition by cases from that, but that does not make much of a difference in the complexity of the context. The solution that is presented here seems reasonable to us.

To summarize this ZFC context we have the following counts:

first order logic	15
definition by cases	2
set theory	6
the axioms	8
choice	1
total	32

and the following types:

```
        prop
[prop]  proof
        set
```

These types are items 1–3 from the list. Of course they are just the types for the encoding of first order logic from the previous section. The reason that in the previous section there were only 14 primitive notions for first order logic while here there are 15, is that we here have the $\in$ predicate as well.

3.2 NF: Quine's set theory of New Foundations

The NF context is based on an e-mail explanation of that system by Randall Holmes. It has the following counts:

first order logic	15
stratification levels	3
stratified formulas	14
new foundations	5
total	37

and the following types:

```
                          prop
                 [prop]   proof
                          set
                          nat
                          prop'
                  [nat]   set'
           [prop][prop']  same_prop
 [n:nat][set][set'(n)]    same_set
                 [prop]   axiom_scheme
```

The first three types are those of first order logic, like before. The type `nat` holds the natural numbers, for the stratification levels. The types `prop'` and `set'` are for the representation of *stratified* first order formulas. These stratified formulas are just syntactic objects, there is no deduction system for them in the context. An example of a constructor of `prop'` is the stratified version of the universal quantifier:

```
* [n:nat][p':[z',set'(n)]prop'] for' : prop' := PRIM
```

The types `same_prop` and `same_set` together represent the judgment that a formula corresponds to some stratified counterpart. For instance the constructor of `same_prop` for the universal quantifier is:

```
n * [p:[z,set]prop][p':[z',set'(n)]prop']
  [_1:[z,set][z',set'(n)][_:same_set(n,z,z')]same_prop(<z>p,<z'>p')]
  same_for : same_prop(for(p),for'(n,p')) := PRIM
```

(Randall Holmes made the observation that one can avoid the `same_set` predicate by having for each `n` a function from `set` to `set'(n)`. Taking this alternative approach would hardly change the complexity of the context.)

Finally the `axiom_scheme` type represents the judgment that a formula is an instance of the NF comprehension axiom. This is needed because the `same_prop` judgment only works for *closed* formulas. But to get the full power of the system, NF comprehension also needs to be there for formulas that have free variables. So the `axiom_scheme` judgment is used to put sufficiently many universal quantifiers around the axiom first to turn it into a closed formula.

4 Systems based on higher order logic

4.1 Isabelle/Pure

The first half of the context for the Isabelle/Pure logic is based on slides by Stefan Berghofer [4]. Later I discovered Section 5.2 of [10]. The equality rules have been modeled after the deduction rules from that section.

The Isabelle/Pure context has the following counts:

meta logic	8
proof terms	5
equality	10
total	23

and the following types:

	`type`
`[type]`	`term`
`[term(prop)]`	`proof`

These types are the equivalents for higher order logic of the types for first order logic from Section 2. But note that the `prop` and `term` types have been unified by having a `prop` object of type `type`.

4.2 HOL Light

The HOL Light system [5] has a very elegant logical kernel. This makes it very obvious what should be in a context that corresponds to the system.

In fact, for each primitive notion in the HOL Light context we can give an ML expression that corresponds to it. In the following list we give these expressions together with their ML types:

`type.ml`

1. `hol_type` *type*
2. `‘:bool‘` `hol_type`
3. `‘:A->B‘` `hol_type`

`term.ml`

4. `term` *type*
5. `Comb` `term` × `term` → `term`
6. `Abs` `term` × `term` → `term`
7. `‘(=)‘` `term`

`thm.ml`

8. `thm` *type*
9. `REFL` `term` → `thm`
10. `TRANS` `thm` → `thm` → `thm`

11.	`MK_COMB`	`thm × thm → thm`
12.	`ABS`	`term → thm → thm`
13.	`BETA`	`term → thm`
14.	`EQ_MP`	`thm → thm → thm`
15.	`DEDUCT_ANTISYM_RULE`	`thm → thm → thm`
16.	*new basic type definition*	`hol_type`
17.	*abs*	`term`
18.	*rep*	`term`
19.	$\vdash abs(rep(a)) = a$	`thm`
20.	$\vdash P(r) = (rep(abs(r)) = r)$	`thm`
`num.ml`		
21.	`‘:ind‘`	`hol_type`
22.	`INFINITY_AX`	`thm`
`class.ml`		
23.	`ETA_AX`	`thm`
24.	`‘(@)‘`	`term`
25.	`SELECT_AX`	`thm`

The items are headed with the names of the HOL Light source files in which they are implemented. Note that the ten basic HOL Light inference rules (see Section 5.3 of [5]) are only 7 of the 25 primitive notions (items 9–15).

To summarize the HOL Light context we have the following counts:

`type.ml`	3
`term.ml`	4
`thm.ml`	13
`bool.ml`	0
`num.ml`	2
`class.ml`	3
total	25

and the following types:

	`type`
`[type]`	`term`
`[term(bool)]`	`thm`

(Apart from the names – `prop` is called `bool` here, and `proof` is called `thm` – these are exactly the same types as the ones in the Isabelle/Pure context.)

5 Type theories

5.1 CC/λC: the Calculus of Constructions in the form of a Proper Type System

The context for the calculus of constructions is based on a tutorial paper about proper type systems by Henk Barendregt [3]. It has the following counts:

terms	4
specifications	3
judgments	2
equality	7
proper type system	5
λC sorts	4
λC axioms	1
λC rules	4
total	30

and the following types:

```
                          pterm
                [pterm]   sort
          [pterm][pterm]  axiom
    [pterm][pterm][pterm] rule
          [pterm][pterm]  in
          [pterm][pterm]  eq
```

The type `pterm` is the type of pseudo-terms. The next three types encode the information about the proper type system: the type `sort(`s`)` is inhabited when s is a sort of the PTS, the type `axiom(`c`,`s`)` is inhabited when $c : s$ is an axiom of the PTS, and the type `rule(`s_1`,`s_2`,`s_3`)` is inhabited when (s_1, s_2, s_3) is a rule of the PTS. The type `in(`A`,`B`)` encodes the judgment $A : B$. Finally the type `eq(`A`,`B`)` represents $A =_\beta B$.

It might seem strange that the λC PTS has two sorts ($*$ and $\Box$), while there are four primitive notions about the λC sorts in the list. These four notions are:

```
* star : pterm := PRIM
* box : pterm := PRIM
* sort_star : sort(star) := PRIM
* sort_box : sort(box) := PRIM
```

5.2 ECC: Luo's Extended Calculus of Constructions

The context for the extended calculus of constructions ECC is based on the original paper about the system [6]. It has the following counts:

universe levels	3
terms	10
conversion	12
type cumulativity	8
inference rules	13
total	46

and the following types:

```
                    omega
                    pterm
[pterm] [pterm]     eq
[pterm] [pterm]     sub
[pterm] [pterm]     in
```

The type `omega` holds the natural numbers used to encode the universe levels. The type `pterm` is for the ECC terms. The types `eq(`A`,`B`)`, `sub(`A`,`B`)` and `in(`A`,`B`)` represent respectively $A \simeq B$, $A \preceq B$ and $A : B$ (see p. 2 of [6] for the meaning of those symbols.)

5.3 MLW$^{\text{ext}}$: extensional Martin-Löf type theory with W-types but no type universes

Originally we tried to formalize Martin-Löf type theory from the standard reference about the subject [9], but there the rules are scattered throughout the book. Then we found a very nice paper by Peter Aczel [1], that gives the rules of the system compactly in a few pages (pp. 7–9) and even has a nice three letter acronym for the system, so that type theory's MLW can share this kind of identification with set theory's ZFC.

The context for MLW$^{\text{ext}}$ has the following counts:

terms	1
judgments	4
equality rules	8
congruence rules	2
the empty type	4
the unit type	7
the Booleans	13
product types	9
sum types	11
W-types	8
extensionality	10
total	77

and the following types:

```
                          pterm
                [pterm]   type
        [pterm] [pterm]   eq_type
        [pterm] [pterm]   in
[pterm] [pterm] [pterm]   eq_in
```

The first type is for pseudoterms. The latter four types are the four basic judgments of Martin-Löf type theory (see Section 4.1 of [9]):

$$A \; set$$
$$A = B$$
$$a \in A$$
$$a = b \in A$$

In [1] these judgments are written (see the description of *pseudobody* on p. 3) as:

$$M \ \mathsf{type}$$
$$M_1 = M_2$$
$$M_0 : M$$
$$M_1 = M_2 : M$$

Our Automath specifications of type theory all use a type for *pseudoterms*. This means that it is possible to write Automath terms that are well-typed in the Automath sense (they have type `pterm`) but that do not correspond to terms that are well-typed in the type theory. The judgments are then used to encode what it means for a pseudoterm to be well-typed in the type theory itself.

An alternative would be to put the type theory types in the Automath types themselves, similar to the way that the types are encoded in the higher order logic contexts. This would lead to the following types:

```
                                  type
                          [type] term
                  [type][type] eq_type
[A:type][term(A)][term(A)] eq
```

However, this is less faithful to the usual descriptions of type theory. If one uses such an encoding, then *conversion* needs an explicit Automath function in the terms. Thorsten Altenkirch explained to me that addition of such a function to a system necessitates the addition of several extra rules. Alternatively one might replace the equality of type theory (where both sides have the same type) by *John-Major equality* [7]. This would lead to the the following set of types:

```
                                  type
                          [type] term
                  [type][type] eq_type
[A:type][B:type][term(A)][term(B)] eq
```

Note that replacing the judgment $a = b \in A$ by the judgment $a \in A = b \in B$ would be a significant departure from traditional Martin-Löf type theory.

6 Category theory

6.1 McLarty's axiomatization of a well-pointed topos with natural numbers and choice

The context for category theory as a foundations of mathematics is based on an e-mail message [8] by Colin McLarty to the FOM mailing list. It has the following counts:

many-sorted first order logic	15
category theory	8
terminal object and cartesian products	9
non-trivial Boolean topos	10
natural numbers	1
well-pointed topos with choice	2
total	45

and the following types:

```
       prop
[prop] proof
       sort
[sort] elt
```

These types are a variation on the types of Section 2, for *many-sorted* first order logic. Instead of `term` the rules use `elt(s)`, with `s` a `sort` of the system. The two sorts of category theory then are introduced by the rules:

```
* object_sort : sort := PRIM
* arrow_sort : sort := PRIM
* object : TYPE := elt(object_sort)
* arrow : TYPE := elt(arrow_sort)
```

Apart from the rules for many-sorted first order logic, this context is just a long list of axioms.

7 Conclusion

7.1 Discussion

The conclusions of this paper are that:

All these foundational systems have roughly the same complexity.

and:

The number of primitive concepts in these systems is bigger than one would expect.

To support the first claim we have put the statistics of the contexts for these systems in three bar diagrams (Fig. 2–4). The conclusions from that are the following. Set theory is not as simple as one might expect (it needs 32 concepts!), but it still is one of the simplest foundations. Therefore the answer to the question in the title of this paper seems to be: 'ZF might be a hack, but we do not have anything better.' Type theory – that is, the calculus of constructions written as a PTS – is still simpler, but if one wants to extend it to a serious system in which one really can do all mathematics, it becomes much more complex. Finally, HOL is the simplest system if one only looks at the number of concepts needed.

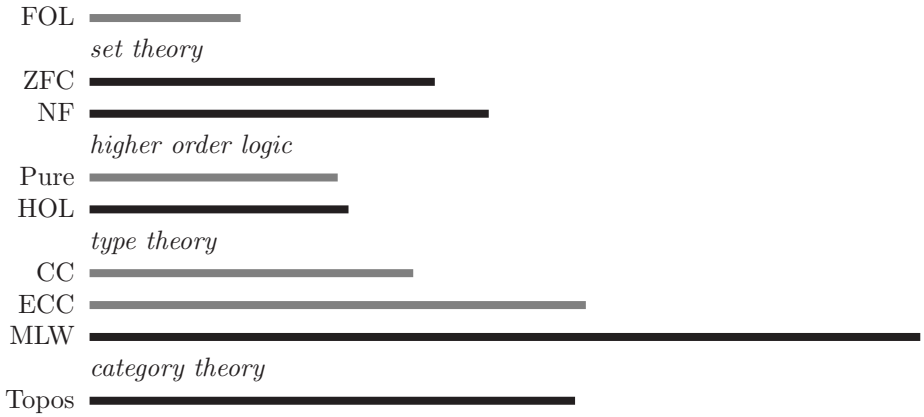

Fig. 2. Comparing systems according to the number of primitive notions

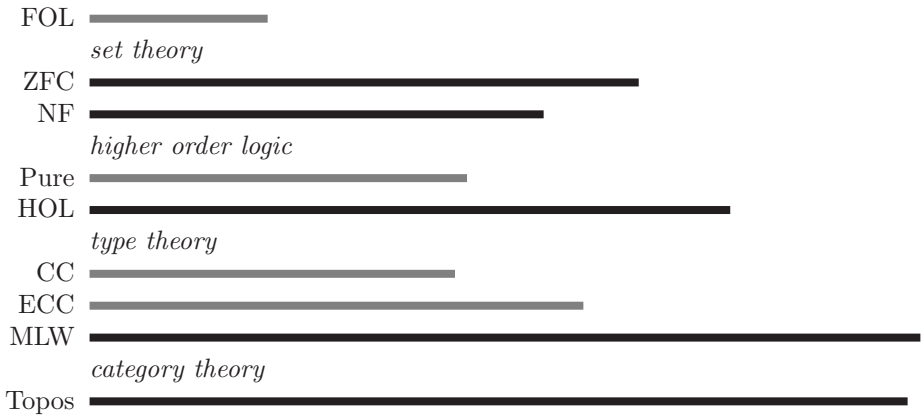

Fig. 3. Comparing systems according to the (compressed) size of the specification

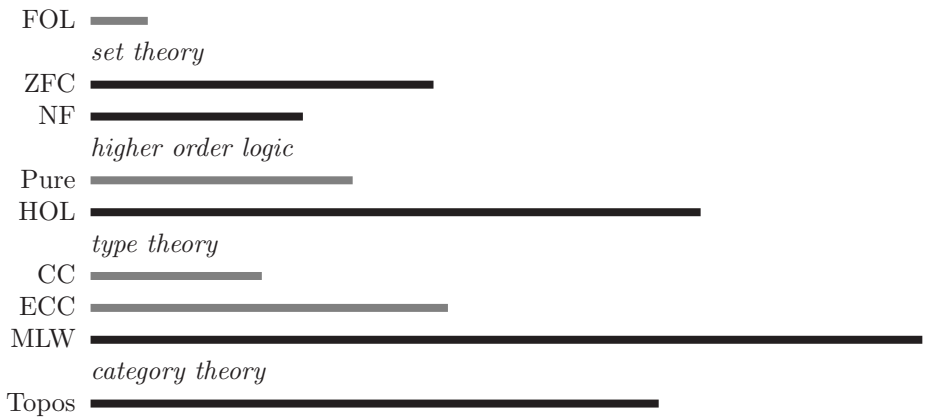

Fig. 4. Comparing systems according to the size of the λ-term

Of course one should note that we are not comparing exactly matching systems here: some of these systems have classical logic and the axiom of choice, while others do not. And one might argue that Isabelle/Pure and the (extended) calculus of constructions are too poor to encode mathematics in a reasonable way by itself,[2] for which reason their bars have been grayed in the figures. We summarize all this in the following table (the '$\circ$' in the row of NF means that NF disproves the axiom of choice):

	classical	choice	'all math'
FOL	•		
ZFC	•	•	•
NF	•	$\circ$	•
Pure			
HOL	•	•	•
CC			
ECC			
MLW			•
Topos	•	•	•

Adding classical logic or choice to a system generally does not make a big difference, though. Generally such an addition involves only one extra primitive notion, and only a few lines of Automath text.

Some remarks about the sizes of these contexts. The NF context is small (if one disregards the gray bars it is the smallest in two out of three categories),

[2] The calculus of constructions cannot prove all equalities that one would like to have. This is obvious from the existence of the proof irrelevance model, which shows that $0 \neq 1$ is not provable.

so it seems an attractive foundation. Unfortunately its consistency strength is unknown.[3]

The Isabelle/Pure and HOL Light logics are very similar, but the HOL Light context is much larger. This is because it is a full context for mathematics (while Isabelle/Pure is not) including infinity and choice: to introduce these two concepts some logic needs to be developed, which is a sizable part of the specification. Also, in the HOL Light context really the whole story is present, including the type definition mechanism. The Isabelle/Pure context represents only an abstracted version of the Isabelle system.

The second conclusion of this paper is that these contexts are more complex than one would expect. The smallest serious one (HOL Light) has 25 primitive notions. Naively one would hope to be able to build all mathematics from something like about 10 notions. A list of 25 items sounds like a long list! (The 'standard model' of elementary particle physics – the foundations of physics – has a list of 18 unexplained dimensionless numbers. This always struck me as a large number. But foundations of mathematics is even worse!)

A reader comments: After Thorsten Altenkirch read this paper his reaction was:

> *I can't help making the obvious comment that simplicity can't be measured by size. Actually, I often find that the contrary is the case that simpler systems are slightly longer than more complicated ones.*
> *E.g. if you look at some of these obfuscated C-programs which are very short but doing something very subtle (people sometimes put them into their sigs), would you call them 'simple'? Are those not precisely what people call a 'hack'?*
> *I mean I think it is a good idea to look for something which short* and *simple. And obviously length can be more easily measured than simplicity. It is just that you say we measure the size to find out which one is simplest.*

For the record: I agree with Thorsten that simple and short are not the same thing. (I had a remark about this, but somehow it did not survive my editing the paper.) So I now put Thorsten's comment in, to address the point.

Please note that I did not try to make the contexts as small as possible by 'obfuscating' them. In each case I tried to keep as close to the original 'informal' description of the system as I could. For instance, it is easy to reduce the number of ZFC axioms by putting them all in one big conjunction. I certainly did not do this.

[3] The consistency strengths of NFU ('NF with urelements'), and of NFU with choice and infinity added, *are* known. Both are weaker than ZFC. Bob Solovay told me that the first has a consistency strength strictly between $I\Delta_0+\mathrm{Exp}$ and $I\Delta_0+\mathrm{SuperExp}$, while the second has the same consistency strength as HOL Light.

7.2 Future work

There are several things that can be done with this:

- One can try to prove these contexts *adequate*. It is easy to show that the various systems can be represented in their contexts as presented here, but one also has to show that it is not possible to derive something in the context – maybe by using the strength of the logical framework – which is not derivable in the system itself. (Whether this is the case also might depend on the framework itself: potentially LF/λP might be able to derive less in the same context than the Automath framework $\Delta\Lambda$.)
  While interesting, the adequacy of the contexts is not relevant for this paper. The goal here is to estimate the complexity of the foundational system. Even if a context is *not* adequate, it probably gives a good impression of the complexity of the corresponding system.
- One can build 'De Bruijn criterion' style checkers based on these contexts. Automath then would be an independent checker for formal mathematics formalized in one of these systems.
  For instance one could have HOL Light generate a stream of correct Automath definitions when doing its proofs. This is closely related to the first kind of HOL to Coq translation described in [13].
- More foundational systems might be represented in the style of this paper. Possible candidates are von Neumann-Bernays-Gödel and Morse-Kelley set theory, Church's original version of higher order logic, and Russell's theory of ramified types.
- The fact that these contexts need so many primitive notions is philosophically unsatisfactory. A system should be looked for that is equivalent to ZFC, but is less complex than the systems presented here. Such a system then would be less of a hack than ZF.

Acknowledgments. Thanks to Thorsten Altenkirch for advice on MLW. Thanks to Herman Geuvers for explaining to me why the calculus of constructions by itself is not enough for all of mathematics. Thanks to Dan Synek and Bas Spitters for putting up with my ravings about my collection of contexts. Thanks to Randall Holmes for advice on NF.

References

1. Peter Aczel. On Relating Type Theories and Set Theories. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs, Proceedings of Types '98*, volume 1657 of *LNCS*. Springer-Verlag, 1999. ⟨`http://www.cs.man.ac.uk/~petera/ts-st.ps.gz`⟩.
2. Mark Balaguer. *Platonism and Anti-Platonism in Mathematics*. Oxford University Press, 1998.
3. Henk Barendregt. Problems in type theory. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, Proceedings of the NATO Advanced Study Institute on Computational Logic, held in Marktoberdorf, 1997*, volume 165 of *Series F:*

Computer and Systems Sciences. Springer-Verlag, 1999. ⟨ftp://ftp.cs.kun.nl/pub/CompMath.Found/marktoberdorf.ps.Z⟩.
4. Stefan Berghofer. New features of the Isabelle theorem prover – proof terms and code generation, 2000. ⟨http://www4.in.tum.de/~berghofe/papers/TYPES2000_slides.ps.gz⟩.
5. John Harrison. *The HOL Light manual (1.1)*, 2000. ⟨http://www.cl.cam.ac.uk/users/jrh/hol-light/manual-1.1.ps.gz⟩.
6. Zhaohui Luo. ECC, an extended calculus of constructions. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA*, pages 386–395, Los Alamitos, CA, 1989. IEEE Computer Society Press. ⟨http://www.dur.ac.uk/~dcs0zl/LICS89.ps.gz⟩.
7. Conor McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, University of Edinburgh, 1999. ⟨http://www.dur.ac.uk/c.t.mcbride/thesis.ps.gz⟩.
8. Colin McLarty. Challenge axioms, final draft. Message ⟨199802061421.JAA28643@po.cwru.edu⟩ as sent to the FOM mailing list, ⟨http://www.cs.nyu.edu/pipermail/fom/1998-February/001181.html⟩, 1998.
9. Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory, An Introduction*. Oxford University Press, 1990. ⟨http://www.cs.chalmers.se/Cs/Research/Logic/book/book.ps⟩.
10. L.C. Paulson. *The Isabelle Reference Manual*, 2003. ⟨http://isabelle.in.tum.de/doc/ref.pdf⟩.
11. Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 17, pages 1063–1147. Elsevier Science Publishers, 1999. ⟨http://www-2.cs.cmu.edu/~twelf/notes/handbook00.ps⟩.
12. Frank Pfenning and Carsten Schuermann. *Twelf User's Guide, Version 1.4*, 2002. ⟨http://www-2.cs.cmu.edu/~twelf/guide-1-4/twelf.ps⟩.
13. F. Wiedijk. Encoding the HOL Light logic in Coq. ⟨http://www.cs.kun.nl/~freek/notes/holl2coq.ps.gz⟩, 2002.