Implementable Set Theory and Consistency of ZFC

Author: Han de Bruijn
Date: 2007 October

Abstract

With help of a bijection, which was basically discovered by Alexander Abian, a "simple model", or rather an *Implementation* of Set Theory in memory of common digital computers, has been conceived, in theory as well as in practice. With the implementation it can be proved that eight out of the nine axioms of ZFC are consistent, and that only the first four axioms are necessary for a constructive build of all sets. The ninth axiom of ZFC, Infinity, is *not* implementable.

Introduction

Instead of augmenting the already numerous attempts with one other attempt for founding the whole of Mathematics upon still another kind of Set Theory, quite a different strategy has been adopted in this article. Our purpose is to obtain a set theory which is just a theory of sets, that is: **not** suitable per se as a Foundation of Mathematics.
One thing that bothers us is: whether such a theory of sets *can be implemented* using existing - and future - computer hardware and software. In such a way that there are no bits and bytes left; our sets must be such that they are able to *consume all* memory of any computer (not yet) available.
The text in this article is accompanied with source code in Delphi Pascal. Most of the time, only the headers of a possible implementation are displayed in the text body. The rest is stored as an archive and can be downloaded here:

`http://hdebruijn.soo.dto.tudelft.nl/jaar2007/OPERATIE.ZIP`

It is emphasized that all (source) code are just rapid prototype programs for demonstration purposes. So please don't expect a plug and play package.
Our theory of sets will **not** be very different from common set theory, except that it seems to be more limited in its foundational scope. But yet it will reveal some hitherto undiscovered features, as might be clear from the *Abstract*.
In order to be sure not to miss anything, sections in the paper are named as chapters in: Paul R. Halmos, *Naive Set Theory*, D. van Nostrand Company, Inc., Princeton 1964. The book is followed as closely as possible. (But not too closely :-) The other key reference is: *on the consistency and independence of some set-theoretical axioms*, by Alexander Abian and Samuel LaMacchia:

`http://projecteuclid.org/DPubS/Repository/1.0/Disseminate`
`?view=body&id=pdf_1&handle=euclid.ndjfl/1093888220`

Last but not least, There is an ongoing, or finished, debate in *sci.math* about the article:

`http://groups.google.nl/group/sci.math/browse_frm/thread/d71aaa8e07e3af15`

Set implementations

It is possible to implement a **set** within the hardware and software of common digital computers in a number of ways. Only four possibilities are employed in this paper. They are:

- a set is an array
- a set is a bitmap
- a set is a number
- a set is a string

Let me clarify this with a few simple examples. Consider the set:

$$\{b, d, d, c, a\}$$

Such a set can conveniently be implemented as an array of objects. In my favorite programming language (Delphi Pascal) it reads:

```
type
  objects = array of TObject;
```

A set with the above objects as members might be created as follows:

```
procedure only_here;
var
  a,b,c,d : TObject;
  S : objects;
begin
{ Urelements }
  a := TObject.Create; b := TObject.Create;
  c := TObject.Create; d := TObject.Create;
{ Set }
  SetLength(S,5);
  S[0] := b; S[1] := d; S[2] := d; S[3] := c; S[4] := a;
end;
```

But, suppose that we consider instead the following set, containing numbers:

$$\{2, 7, 7, 5, 1\}$$

Then the same set can also be implemented by a series of bits, a *bitmap*, where a "bit up" | means that the corresponding element is a member of the set:

```
.. 0010100110
     | |  ||
.. 9876543210
```

The bits are numbered from the right to the left, starting with zero. Thus bit 0 is down i.e. 0 is not a member; bit 1 is up i.e. 1 is a member; bit 2 is up, i.e.

2 is a member; bit 3 is down i.e. 3 is not a member; bit 4 is down i.e. 4 is not a member; bit 5 is up i.e. 5 is a member; bit 6 is down i.e. 6 is not a member; bit 7 is up i.e. 7 is a member. The calling sequence for encoding an array of integers as a bitmap could be defined by:

```
function makeSet(S : integers) : bitmap;
```

But one and the same bitmap can also interpreted as a *number*:

```
.. 0010100110 (binary) = 166 (decimal)
```

At last, the set $O = \{b, d, d, c, a\}$ as well as the set $N = \{2, 7, 7, 5, 1\}$ can be implemented (more or less trivially) as strings. In Delphi Pascal this would read:

```
O := '{b,d,d,c,a}';
N := '{2,7,7,5,1}';
```

It will be assumed in the sequel that the size of all set implementations can be as large as needed, which is the same as saying that we have a virtually unlimited amount of memory at our disposal. Thus all possible limitations of current (as well as future) digital computer hardware are assumed not to be relevant in any respect. Let's start reading in Halmos' book now.

Axiom of Extensionality

The most important undefined notion in set theory is *being a member* of a set. If a is a member of A then we shall write, according to Halmos:

$$a \in A$$

Fortunately, this notion can be mapped upon our TObject arrays in a well defined manner. Calling sequence defined by:

```
function is_in(e : TObject; X : Objects) : boolean;
```

It can also be mapped upon our bitmapped arrays in a well defined manner. Calling sequence defined by:

```
function is_in(e : integer; X : bitmap) : boolean;
```

Programming such calling sequences for a strings implementation is a bit less trivial, because it would require a parsing mechanism. Therefore any strings implementation will be postponed until we bounce against limitations of the others.
If A and B are sets and each member of A is also a member of B then we say that A is a *subset* of B. And, according to Halmos, this shall be written as:

$$A \subseteq B \quad \text{or} \quad B \supseteq A$$

This can be implemented in software quite easily. Calling sequences defined by:

```
function SubSet(A,B : Objects) : boolean;
function SubSet(A,B : bitmap) : boolean;
```

The Axiom of Extensionality states that two sets are *equal* if and only if they have the same elements. If A is equal to B then we write:

$$A = B$$

And the fact that A and B are not equal is expressed by writing:

$$A \neq B$$

Equality of sets $A = B$ can be conveniently defined as: A is a subset of B and B is a subset of A. Formally:

$$(A = B) \qquad \equiv \qquad ((A \subseteq B) \quad \text{and} \quad (A \supseteq B))$$

This too can be easily implemented in software. Calling sequences defined by:

```
function Equal(A,B : Objects) : boolean;
function Equal(A,B : bitmap) : boolean;
```

We conclude that the Axiom of Extensionality is piece of an implementable set theory. And two implementations have been provided already.
A warning is in place, though. Extensionality might be not as simple as it looks like, at first sight. Especially with complicated sets, where the elements themselves in turn are sets of sets (towers of sets), the look and feel of a set can become quite messy. One way to get rid of that messiness, at least in part, is to adopt the following more detailed rules:
- if x and y are members of a set, so are y and x (x and y can be swapped)
- if x and x are members of a set S, then only x is a member of S (squeeze x)
Example: swap c and d, squeeze d and d, swap b and a, swap b and d. In:

$$\{b, d, c, d, a\} = \{b, d, d, c, a\} = \{b, d, c, a\} = \{a, d, c, b\} = \{a, b, c, d\}$$

In practice, this means that duplicate elements in a set can be removed (*terse* sets). And that a set can be sorted "in alphabetic order".
Herewith, chapter 1 of Halmos' book is covered a great deal. The reader is encouraged to look up *there* any details of common naive set theory which might be missing in our treatment. We are not going to repeat all of the well known stuff that is common to traditional *and* implementable set theory.
The chapter 'Specification Axiom' will be skipped altogether for the moment being. The reason is that logic and predicates have not been done yet as an implementable theory, at least not by this author.
It is noted, though, that if theorems are true in an implementable form of mathematics, then they are also true in a *constructive* mathematical sense. If the reverse is also true, then this could provide some important clues.

Axiom of empty set

We have arrived in Halmos' book at chapter III. However, because we are lacking a 'Specification Axiom' (a la Halmos chapter II), it is necessary to start with at least one set which comes out of the blue sky. Fortunately though, the Axiom of *empty set* is one of the axioms of Zermelo Fraenkel set theory. The empty set is a set without elements. The extensionality axiom says that all such sets are equal. Thus there is only *one* empty set. The usual symbol for the empty set is:

$$\emptyset$$

But because it's is a set with no elements (between its curly braces) another useful notation for this "empty box" is:

$$\{\}$$

The only property of the empty set to be mentioned in this section is the fact that it is a subset of any set A:

$$\emptyset \subseteq A$$

How is the empty set implemented in *our* set theory? As an empty array:

```
SetLength(empty,0);
```

And indeed, this is true in theory as well as in practice: the empty array is a subset of any array.
Another implementation of the empty set in our set theory is a bitmap of zeroes; the bitmapped implementation is any with **no** bits up:

```
.. 0000000000
```

However, exactly the same bit sequence can be interpreted as a number. What number does it represent then? Answer: the number zero. Therefore we conclude that the empty set is equivalent with zero. Or rather that it *is* zero. Not quite perhaps. Let's stay cautious for the moment being and write the zero with an overline, to indicate that it is a set, not (yet) a number:

$$\{\} = \overline{0}$$

A little philosophy may be in place here. It is noted that the content of the empty set is emptiness itself. But the empty set is equivalent with zero. Thus zero is not emptiness itself. It *contains* emptiness. Emptiness itself is represented in programming languages by something even more nothing than zero, commonly called NIL or NULL. End of philosophical note.

Pairing and Union

So far so good about the empty set. Lets proceed with the Axiom of pairing: *for each two arbitrary sets, there is a set that contains them both as a member.* Now we are going to do some *real* set theoretical stuff. With the empty set alone,

we can form the following set, with help of pairing. But the extensionality axiom says that it is equal to a set with only the empty set as an element. And the resulting set is not empty !

$$\{\emptyset, \emptyset\} = \{\emptyset\} = \{\{\}\} \neq \{\}$$

Maybe it is possible to implement this as array sets, with help of a decent computer language. But as far as I know, it's cumbersome with Delphi Pascal. Empty boxes inside empty boxes. (It's only until recently that I am beginning to feel a bit comfortable with the mere idea.)
With the axiom of pairing, we can make *singletons*, i.e. sets with only one *element* in them. But, of course, we can make pairs as well, once a few singletons are created. For example:

$$\{\{\}\} \quad \text{and} \quad \{\} \quad \Longrightarrow \quad \{\,\{\{\}\}\,,\,\{\}\,\}$$

Another philosophical note is in place, when we are saying that we "make with an axiom" and denote this as an implication $A \Longrightarrow B$. In common mathematics, the implication $\Longrightarrow$ just means what is defined by a truth table in propositional logic. But there is another form of mathematics, called *constructivism*. Within constructivist mathematics, an implication has a more "operational" meaning, like: given A, we can *construct* B from A. So if we say "make with an axiom", then it is expressed herewith that we adhere to the constructivist meaning of an implication. End of philosophical note.
The axiom of pairing allows us to create singletons and pairs. But it doesn't allow us to create sets with more than two elements. In other to accommodate with the latter possibility, there is another axiom in common set theory, which is called the Axiom of Union:
For any set x there is a set y whose elements are precisely the elements of the elements of x.
Sounds more complicated than it actually is. A simple example. Union says that, if there is a set $\{\{1,2\},\{3\}\}$, then there is also another set, namely $\{1,2,3\}$. It is seen that the former set is a pair and the latter set contains more than just two elements. We can proceed in this way, forming the pair and the union $\{\{1,2,3\},\{4\}\} \Longrightarrow \{1,2,3,4\}$, and so on and so forth. It is noted once more that we will not dwell upon the well known set theoretic properties of *union* and *intersection*. For the obvious reason that it can all be found in many books about common set theory (e.g. Halmos' book).
So, for the moment being, we end up with only four out of the nine axioms of Zermelo Fraenkel (and axiom of Choice, eventually: ZFC) set theory:

1. axiom of Extensionality
2. axiom of empty set
3. axiom of Pairing
4. axiom of Union

Still another approach might be feasible: via construction rules. As published earlier in 'sci.math' by this author and (to be) polished later by Chas Brown.

Hereditary Sets

Convenient references can be found on the Internet:

`http://en.wikipedia.org/wiki/Hereditary_Set`
`http://en.wikipedia.org/wiki/Hereditarily_finite_set`

The end result of the preceding section is a *construction* method for the so called *hereditarily finite sets*:

- The empty set is a hereditarily finite set.
- If $a_1, ..., a_k$ are hereditarily finite sets, then so is $\{a_1, ..., a_k\}$.

Note that adopting this as an axiom schema will not replace the axiom of union, which is still needed for other situations (in standard set theory, at least). The axiom of pairing may be replaced by the above generalization, though.
Let's see what the most simple hereditary sets look like. We found a notation for the *empty set*, that distinguishes it a bit from zero:

$$\{\} = \overline{0}$$

With the empty set alone, we can make - employing pairing and extensionality - the singleton that contains it. We give it a name and associate it with the next natural number:

$$\{\ \{\}\ \} = \{\overline{0}\} = \overline{1}$$

With the sets numbered as $\overline{0}$ and $\overline{1}$, we can proceed and create the following sets - employing pairing and extensionality:

$$\overline{2} = \{\overline{1}\} = \{\{\overline{0}\}\} = \{\{\{\}\}\}$$

$$\overline{3} = \{\overline{0}, \overline{1}\} = \{\{\}, \{\{\}\}\}$$

Next we do:

$$\begin{aligned} \overline{4} &= \{\overline{2}\} = \{\{\{\{\}\}\}\} \\ \overline{5} &= \{\overline{0}, \overline{2}\} = \{\{\}, \{\{\{\}\}\}\} \\ \overline{6} &= \{\overline{1}, \overline{2}\} = \{\{\{\}\}, \{\{\{\}\}\}\} \end{aligned}$$

But here we are stuck, because the axiom of pairing forbids us to do something like this:

$$\overline{7} = \{\overline{0}, \overline{1}, \overline{2}\}$$

So we need the axiom of union for the first time:

$$\{\overline{3}, \{\overline{2}\}\} = \{\{\overline{0}, \overline{1}\}, \{\overline{2}\}\} \quad \Longrightarrow \quad \{\overline{0}, \overline{1}, \overline{2}\} = \overline{7} = \{\{\}\{\{\}\{\{\}\{\{\}\}\}\}\}$$

Where it is noted that it would become more and more difficult to decide if a set has "equal members" eventually. Especially if we leave out the commas, which may be essential for readability, but not for definiability. It is thus argued, once

more, that the axiom of extensionality might be not so trivial as it looks like. Things we have found so far can also be written as follows, while skipping the first four results:

$$\begin{aligned}\overline{4} &= \{\overline{2}\}\\ \overline{5} &= \overline{1} \cup \overline{4}\\ \overline{6} &= \overline{2} \cup \overline{4}\\ \overline{7} &= \overline{3} \cup \overline{4}\end{aligned}$$

Where it is noted that actually elements of a Pair (e.g. $\{\overline{1}, \overline{4}\}$) are taken for making a Union with $\cup$. Continuing this story, we find:

$$\begin{aligned}\overline{8} &= \{\overline{3}\}\\ \overline{9} &= \overline{1} \cup \overline{8}\\ \overline{10} &= \overline{2} \cup \overline{8}\\ \overline{11} &= \overline{3} \cup \overline{8}\\ \overline{12} &= \overline{4} \cup \overline{8}\\ \overline{13} &= \overline{5} \cup \overline{8}\\ \overline{14} &= \overline{6} \cup \overline{8}\\ \overline{15} &= \overline{7} \cup \overline{8}\end{aligned}$$

General pattern:

$$\overline{2^n} = \{\overline{n}\}$$

$$\overline{2^n + k} = \overline{k} \cup \overline{2^n} \qquad \text{where} \quad 1 \leq k \leq 2^n - 1$$

This is a recursive pattern. There are precisely enough numbers k already defined with every jump from 2^n to the next power of 2.
Preliminary **Definition.** The *Binary successor* $(\overline{x})^+$ of a set $\overline{x}$ is defined by the above pattern. Well, guess that the binary successor of $\overline{166}$ will be $\overline{167}$ ...
We will shortly reveal what the full secret behind this Binary successor is. For the moment being, it's important to note that it's quite different from the common successor function in mainstream mathematics, which is defined as: $(\overline{x})^+ = \overline{x} \cup \{\overline{x}\}$.
The mapping of sets upon numbers, denoted by an overline as in $\overline{x}$, has not originally be invented by me (Han de Bruijn), but by a mathematician named *Alexander Abian*. One of my key references is the paper '*on the consistency and independence of some set-theoretical axioms*'. The notation is in there all over the place. But, according to more standard mathematics, the overline should be written as a mapping. And a bijection, say the 'Abian' function A, which maps sets oneto integers, should be defined in the first place. After doing so, we could proceed with, for example:

$$A(\ \{\{\}\{\{\}\{\{\}\}\}\}\ ) = 7 \quad \text{and} \quad A^{-1}(7) = \{\{\}\{\{\}\{\{\}\}\}\}$$

Chas Brown has proved in 'sci.math' that the *A*bian mapping is indeed a true bijection:

http://groups.google.nl/group/sci.math/msg/59ea756bdbccf44c

But instead of all that jazz with bijections and overlines, we will follow an even more simple minded approach and simply *equate* sets with numbers. This is motivated by the fact that sets and numbers share exactly the same bitmaps, when implemented in a computer's memory. As will be revealed now.

Bitmapped Sets

In the subsection about the empty set, we have shown that the empty set is equivalent with an integer number, namely zero (without the overlines here):

$$\{\} = 0$$

We have the following rule of formation for hereditarily finite sets:
- If $a_1, ..., a_k$ are hereditarily finite sets, then so is $\{a_1, ..., a_k\}$.
In particular, starting with the empty set:

$$\{\} = 0 \quad \Longrightarrow \quad \{\{\}\} = \{0\}$$

But the set containing only a zero can be bitmapped upon a sequence of bits where only the bit at position zero is up:

```
.. 0000000001
```

And the numerical (integer) equivalent of this is the number 1 :

$$\{\} = 0 \quad \Longrightarrow \quad \{\{\}\} = \{0\} = 1$$

We can proceed in this way:

$$\{\,\{\{\}\}\,\} = \{1\} = ?$$

Bitmap:

```
.. 0000000010
```

Hence:

$$\{\,\{\{\}\}\,\} = \{1\} = 2$$

Next example:

$$\{\,\{\{\}\}, \{\}\,\} = \{1, 0\} = 3$$

Bitmap:

```
.. 0000000011
```

The general pattern may be formulated as follows:
- $\{\} = 0$ is a set (the empty set = zero)
- if $A_1, ..., A_k$ are sets, then $\{A_1, ..., A_k\} = 2^{A_1} \cup ... \cup 2^{A_k}$ is a set

Note. A slightly less accurate notation is employed in the paper by Alexander Abian, where the "bitwise or" $\cup$ is denoted by a plus $+$. An even better notation would be: $(1 \leftarrow A_1) \cup ... \cup (1 \leftarrow A_k)$, where $\leftarrow$ denotes a shift to the left ('**shl**' in Delphi Pascal terms).
Every set can formed in this way, through a finite sequence of applications of the above rules. Examples were:

$$\{\{\}\} = 2^{\{\}} = 2^0 = 1$$
$$\{\,\{\{\}\}\,\} = 2^{\{\{\}\}} = 2^1 = 2$$
$$\{\,\{\{\}\}, \{\}\,\} = 2^{\{\{\}\}} \cup 2^{\{\}} = 2 \cup 1 = 3$$

This procedure is programmed as a recursive routine in Delphi Pascal. "Towers" (powers of two) get a numeric meaning as soon as the empty set is encountered:

```
function natural(hereditarily : string) : integer;
```

It is noted that such a routine should always be preceded by a check on correct syntax, something that has not been done in the present rapid prototyping. (As a consequence, sets can be processed now which are not really well formed sets)
A somewhat more elaborate example has been published repeatedly in 'sci.math':

$$\{\{\{\}\{\{\}\}\{\}\{\{\{\}\}\}\{\}\{\{\}\}\{\{\}\}\}\} = 166$$

Where it commas and blanks are left out for unreadability (:-)
We can easily see now what the secret is behind the *Binary successor*, as mentioned in the preceding section. *If any set is mapped upon the binary number x, then the binary successor of that set is the one which is mapped upon the next binary number $x + 1$.* Hence the name: binary successor. This is all due to the fact that sets and numbers share exactly the same bitmaps, when implemented in a common computer's memory.

Set mapped Numbers

The inverse procedure will be explained at hand of the 166 number. It's an elementary exercise to find the binary representation of it:

$$166 = 128 + 32 + 4 + 2 = 2^7 + 2^5 + 2^2 + 2^1 =$$
$$2^{(2^2+2^1+2^0)} + 2^{(2^2+2^0)} + 2^{2^1} + 2^{2^0} = 2^{(2^{2^{2^0}}+2^{2^0}+2^0)} + 2^{(2^{2^{2^0}}+2^0)} + 2^{2^{2^0}} + 2^{2^0}$$

Here the $(+)$ signs could as well have been replaced by $(\cup)$ signs, because no two elements are the same in this representation (of course not). So here comes our bijection upon the hereditarily finite set:

$$0 = \{\} \implies 1 = \{0\} = \{\{\}\}$$
$$\implies 2 = \{1\} = \{\{\{\}\}\}$$
$$5 = \{2, 0\} = \{\,\{\{\{\}\}\}\,, \{\}\,\}$$
$$7 = \{2, 1, 0\} = \{\,\{\{\{\}\}\}\,, \{\{\}\}\,, \{\}\,\}$$
$$\implies 166 = \{7, 5, 2, 1\} =$$
$$\{\,\{\,\{\{\{\}\}\}\,, \{\{\}\}\,, \{\}\,\}\,, \{\,\{\{\{\}\}\}\,, \{\}\,\}\,, \{\{\{\}\}\}\,, \{\{\}\}\,\}$$

The implementation in Delphi Pascal is again a recursive routine:

```
function Elements(G : integer) : string;
```

Calling sequence as in:

```
  test(Elements(166));
```

Giving as output the (meanwhile well known) result:

```
{{{}}{{{}}}{{}{{{}}}}{{}{{}}{{{}}}}} = 166
```

If we define the "tower" (power of two) as:

```
function T(N : integer) : integer;
begin
  T := 1 shl N;
end;
```

Then we can arrange the outcome of an 166 experiment as follows:

```
{ { { }}  { { { }}}  { { }  { { { }}}}  { { }  { { }}  { { { }}}} = 166
(T(T(0))+T(T(T(0)))+T(T(0)+T(T(T(0))))+T(T(0)+T(T(0))+T(T(T(0))))) = 166
```

Leading to the following **conjecture**: the structure of the parentheses with nesting and adding towers is exactly the same as the structure of the curly brackets in the corresponding hereditarily finite set representation.
Mathematical note: our definition of a "tower" function is in agreement with others, as can be Google'd up with "tower function" and "Maple". We find:

```
base^^0=1
base^^(height+1)=base^(base^^height)
```

From which the rest follows. In our case 'base' = 2 and 'height' = integer ≥ 0. The mapping of numbers onto sets is also useful from another point of view. I don't claim it will ever become a method with beats others from a viewpoint of effiency, but representing numbers as sets in array implementations certainly is an alternative way of doing calculations with *very large numbers*. A rapid prototyping package, previously called ZERMELO.ZIP, is published on the web, for demonstration purposes only:

```
http://hdebruijn.soo.dto.tudelft.nl/jaar2007/OPERATIE.ZIP
```

It all works by means of the *standard array implementation* of sets, as will be defined in the next section. Calling sequence of (main) addition and multiplication routines defined by:

```
function plus(a,b : integers) : integers; { Add }
function maal(a,b : integers) : integers; { Multiply }
```

A little example in footnote style:

```
53! = 427488328406002556429801375338939964969034378836681372467200000000000
```

Standard array sets

We have seen that there are several ways to implement a theory of sets with hardware and software of (not yet) available digital computers. But only four possibilities out of many shall be employed in this paper. They are:

- a set is an array
- a set is a bitmap
- a set is a string
- a set is a number

Due to the bitmapping, all hereditarily finite sets are equivalent with arrays of naturals (including zero). For example:

$$\{\{\{\}\{\{\}\}\{\}\{\{\}\}\}\{\}\{\{\}\{\{\}\}\}\} = 166 = \{7, 5, 2, 1\}$$

But, due to our axioms, we have nothing else than hereditarily finite sets. So there can be no other objects than naturals (or zero) within the curly bracket arrays $\{a_1, a_2, ..., a_n\}$. Moreover, the Extensionality axiom says that it has no sense to account for a member of such a set more than *once* upon a time. Last but not least, the *order* of the elements in a set is not at all relevant. As exemplified with set number 166, the following sets must be the same:

$$\{7, 5, 2, 1\} = \{7, 5, 2, 5, 1, 7\} = \{5, 7, 1, 2\} = \{1, 2, 5, 7\}$$

Effectively, our general TObject arrays have become obsolete with the above. And "hereditarily finite set" is just another name for: implementable set.
Definition. The *standard* implementation of a set, as an array, is an *ordered* array, consisting of integers greater than or equal to zero. It is a *terse* set as well, meaning that all duplicates have been removed.
With the above definition, sets arrays can be implemented far more efficiently than in the "general" TObject case. Sorting and wiping out the duplicates with set creation, binary search with member finding, these shall be useful methods to enhance efficiency, indeed. See accompanying source code for details.
Now that sets and naturals have become equivalent, it's important to note some other equivalencies. The one for equality ($A = B$) being more than trivial.
Theorem.

$$\begin{aligned} A \subseteq B \quad &\Longrightarrow \quad A \leq B \\ A \supseteq B \quad &\Longrightarrow \quad A \geq B \end{aligned}$$

It may be not a coincidence that $\leq$ and $\geq$ are used with Object Pascal set implementations as a substitute for $\subseteq$ and $\supseteq$. The reverse is not true, though: $7 < 8$ but $7 \not\subset 8$.

Complements and Powers

We have arrived at chapter V and more in Halmos' book: Complements and Powers. Well, complements in an implementable theory of sets are faced with

exactly the same problems as complements in common set theory: there is *no universal set* (i.e. set of everything) available, not in theory and not in practice. Therefore only *relative complements* can be conceived, as well as implemented. Let's take a break and take a look at the way kind of set theory has been implemented in my favorite programming language: Delphi Pascal. The complements problem has been solved here in a rather obvious way, which is not quite bad for a start. When seen from the set arrays implementation, all members of an Object Pascal set have an "ordinal value" which is in the range [0..255]. This is essentially the same as in the standard implementation of set arrays, as presented in this paper. These set arrays map onto bitmaps which are at most 256 bits wide, meaning that the compiler should reserve 256 bits of memory for each set. Because of this limitation, complements of sets are constructed easily: reverse all bits 0..255. It leaves no doubt that the elementary set operations, like union and intersection, are carried out by bitwise **or**'s respectively **and**'s. It's all very simple and straightforward, once accepted the limitations of a much restricted 0..255 bits universe.
It is noted that there does **not** exist a 'typecast' in Object Pascal that maps sets onto integers or vice versa, such as:

```
number := Integer(mySet);
mySet := Set(number);
```

Not a missed opportunity, I think, but merely a hitherto unknown possibility. So far so good about complements.

Where the original overline notation for bitmapped sets is retained and it is assumed that any original set is *terse*, i.e. it contains no duplicate members. We have promised that the implementable sets can fill up all feasible computer memory, without leaving any room for other things. Let's repeat some easy facts about common computer memory:

$$\begin{aligned} 1\ \text{Byte} &= 2^3\ \text{Bit} \\ 1\ \text{KiloByte} = 2^{10}\ \text{Byte} &= 2^{13}\ \text{Bit} \\ 1\ \text{MegaByte} = 2^{10}\ \text{KiloByte} &= 2^{23}\ \text{Bit} \\ 1\ \text{GigaByte} = 2^{10}\ \text{MegaByte} &= 2^{33}\ \text{Bit} \\ 1\ \text{TeraByte} = 2^{10}\ \text{GigaByte} &= 2^{43}\ \text{Bit} \end{aligned}$$

Suppose now that we make the following set. It can be done with Delphi Pascal:

```
procedure Pascal;
{
  Sets in Object Pascal
}
var
  A : Set of byte;
  k : integer;
begin
  A := [];
  for k := 0 to 42 do
    A := A + [ k ];
end;
```

Looks rather innocent, and way below the compiler's boundary of 256 bits wide. But watch out! The power set is *somewhat* more powerful:

$$P(\{0, 1, 2, 3, .., 42\}) = \{0, 1, 2, 3, .., 2^{43} - 1\}$$

Oops! This is a bit sequence of a .. *TeraByte*, with all bits up, i.e not a single bit left unused. Thus we we see that our implementable set theory can easily consume up all contemporary and future computer resources. As promised.

Axiom of Foundation

Let Google be your friend and type "Axiom of Foundation". Then we find:
Every non-empty set x contains some member y such that x and y are disjoint sets. Every non-empty set is disjoint from at least one of its elements.
Some consequences:
- No set is an element of itself
- A set contains no infinitely descending (membership) sequence
- Any set contains a (membership) minimal element
- foundation states that every set can be built up from the empty set
- [...] least useful ingredient of ZFC

My first confrontation with the Axiom of Foundation has been when I tempted to argue that an element should be equal to the set containing only that element. This has been called the "ZFC Killer Axiom" (but only by *this* author):

$$a = \{a\}$$

Adding it to ZFC is indeed disastrous. It's killing power comes from the fact that it is in direct conflict with the axiom of Foundation. But it's much worse than this:

$$\{\} = 0 \quad \Longrightarrow \quad 0 = \{0\} = 1$$

Do we have to proceed? One can easily prove herefrom that every number is zero. Thus $a = \{a\}$ does not only kill Set Theory, but all Natural numbers as well - they become zero - and therefore it kills All Mathematics. Who said that Foundation is generally considered as the 'least useful ingredient of ZFC' .. ?
Let's start with a couple of examples, to prove the validity of the axiom within Alexander Abian's "simple model" / our implementable set theory.

$$S = \{0, 1, 2, 3, 4, 5\}$$

And examine Foundation:

$$\begin{aligned} 0 \cap S = \{\} \cap S = \emptyset \\ 1 \cap S = \{0\} \cap S = \{0\} = 1 \\ 2 \cap S = \{1\} \cap S = \{1\} = 2 \\ 3 \cap S = \{0, 1\} \cap S = \{0, 1\} = 3 \\ 4 \cap S = \{2\} \cap S = \{2\} = 4 \\ 5 \cap S = \{2, 0\} \cap S = \{2, 0\} = 5 \end{aligned}$$

There is only one element disjoint from the set, namely 0 (because it's empty from itself). Therefore let's drop that first element:

$$S = \{1, 2, 3, 4, 5\}$$

And examine Foundation again:

$$\begin{aligned} 1 \cap S = \{0\} \cap S = \{\} = \emptyset \\ 2 \cap S = \{1\} \cap S = \{1\} = 2 \\ 3 \cap S = \{0, 1\} \cap S = \{1\} = 2 \\ 4 \cap S = \{2\} \cap S = \{2\} = 4 \\ 5 \cap S = \{2, 0\} \cap S = \{2\} = 4 \end{aligned}$$

So again there is only one element disjoint from the set, namely 1 .
The following function, named ${}^2log$ and coded as below in Delphi Pascal, is essential for the rest of our argument. The function ${}^2log$ essentially determines the position of the leftmost bit in an integer ≥ 0.

```
function log2(n : integer) : integer;
{
  Logarithm base 2
  ---------------- }
var
  k : integer;
begin
  log2 := 0;
  if n < 2 then Exit;
  k := 0;
  while n > 0 do
  begin
    n := n shr 1 ; k := k + 1;
  end;
  log2 := k-1;
end;
```

Lemma. ${}^2log(n) < n$ for each integer $n \geq 0$
Theorem. *The minimal element in a set, when implemented as a natural array, is always disjoint from the set.*
Proof. This is a direct consequence of the above lemma: the leftmost bit position of an integer ≥ 0 has a numeric value which is always less than the numeric value of the integer itself.
This completes the proof, of the assertion that the axiom of Foundation is just a Theorem in our implementable set theory.

Abian Set Theory

One of my key references is the paper 'on the consistency and independence of some set-theoretical axioms' by Alexander Abian and Samuel LaMacchia:

`http://projecteuclid.org/DPubS/Repository/1.0/Disseminate ?view=body&id=pdf_1&handle=euclid.ndjfl/1093888220`

Their sets, denoted with the overline notation, were independently discovered by myself, as becomes evident from the 'sci.math' thread named "Bit mapped Set Theory":

`http://groups.google.nl/group/sci.math/msg/d0e63882f157ecbe`

So "Bit mapped Set Theory", unfortunately, turns out to be a re-invention of the wheel. But, fortunately, Abian's paper covers so many other interesting topics, that I feel compensated more than enough. An obvious flaw is spotted in very first sentence, though: *In this paper by means of simple models.* Oh no, mister Abian! The way memory is organized in common digital computers is not just a "simple model". It is the main Laboratory of Applied Mathematics! Whatever .. The most interesting result in the paper - I think - is: Theorem 1.

The axioms of Extensionality, Replacement, Power-Set, Sum-Set, and Choice form a consistent set of axioms. These are 5 out of the 9 ZFC axioms.

Proof: with overline (bitmapped) sets as a "model". It's no surprise at all that the *Axiom of Infinity* (9) turns out to be independent of these 5 axioms. But, at least to me, it's a surprise that the following axioms are not even mentioned:

6. axiom of Subset
7. axiom of Pairing
8. axiom of Foundation

It's such a surprise because the proofs presented in the paper are way more complicated than the proofs which would be needed to include the axioms (6-8) as well. It's trivial from the outset (see section "Pairing and Union") that the axiom of Pairing is valid within Abian's "model". In the previous section, it has been proved that the axiom of Foundation is valid too. Including the axiom of Subset - at last - must be easy, when compared with the axiom of Replacement. In the style of Abian's article, one could say that, given e.g. a set $\overline{166} = \{1, 2, 5, 7\}$, it is possible with a predicate e.g. $P(x) = (x \text{ is odd})$ to make a subset $\{1, 5, 7\}$. A more elaborate example could be the following. Suppose we have a table named A, consisting of two columns, the first one containing $\overline{166}$ and the second one named 'colors'. Note that the first column is commonly called a 'primary key' in database environments.

```
key | color
-----------
 1  | blue
 2  | red
 5  | red
 7  | blue
```

Then we can create a subset $B = \{2, 5\}$, with an SQL database query:

```
create table B as (select key from A where color = 'red');
```

Anyway, it is clear that the Subset axiom is an *implementable* piece of Set Theory. So here comes our **Theorem.**
The first eight (8) axioms of Zermelo Fraenkel (Choice) set theory (ZFC) form a consistent set of axioms. The axiom of Infinity (9) is independent of the other 8 axioms. ZFC without Infinity is constructively complete with only the four axioms of Extensionality, empty set, Pairing and Union. The other 4 axioms are redundant - that is: they are theorems - for finitary ZFC.
If all of the argumentation for the above Theorem constitutes a mathematically valid proof, then this does establish, once more, that the proof mechanism of an implementable (set) theory can be extremely powerful. Cross fertilization between Theory and Practice, synergy, win win strategy ...

Ordered pairs

Ordered pairs are defined in common set theory as Kuratowski ordered pairs:

$$(a,b) := \{\{a\},\{a,b\}\}$$

Elaborate:

$$\{\{a\},\{a,b\}\} = \{2^a, 2^a + 2^b\} = 2^{2^a} + 2^{2^a+2^b} = 2^{2^a}\left(1 + 2^{2^b}\right)$$

Examples:

$$(0,1) = 2^{2^0}\left(1 + 2^{2^1}\right) = 2(1+4) = 10$$
$$(1,0) = 2^{2^1}\left(1 + 2^{2^0}\right) = 4(1+2) = 12$$
$$(0,2) = 2^{2^0}\left(1 + 2^{2^2}\right) = 2(1+16) = 34$$
$$(2,0) = 2^{2^2}\left(1 + 2^{2^0}\right) = 16(1+2) = 48$$
$$(1,2) = 2^{2^1}\left(1 + 2^{2^2}\right) = 4(1+16) = 68$$
$$(2,1) = 2^{2^2}\left(1 + 2^{2^1}\right) = 16(1+4) = 80$$

It is noted that the second example is employed in Alexander Abien's article, where it reads:

$$\{(\overline{1},\overline{0})\} = \{\{\{\overline{1}\},\{\overline{1},\overline{0}\}\}\} = \{\{\overline{2},\overline{3}\}\} = \{\overline{12}\} = \overline{4096}$$

But anyway, it all runs quickly out of hand ! For example:

$$(3,4) = 16777472 \quad \text{and} \quad (4,3) = 16842752$$

So an essential difference between standard set theory and an implementable theory of sets may be that the latter is far more "dense". Such an efficient way to store ordered pairs in computer memory is e.g. the one published by Daryl McCullough, in the 'sci.math' thread "Coding of ordered pairs":

```
Let <i,j> = (i+j)*(i+j+1)/2 + i
```

The inverse function, the one that splits $< i, j >$ into its components x and y, can be found as well, though with somewhat more effort. Here is my Delphi Pascal implementation:

```
function Daryl(n : integer) : paar;
{
  Natural -> ordered pair
}
var
  f,K : integer;
```

```
  p : paar; { (p.i,p.j) }
begin
  f := (Trunc(sqrt(8*n+1)) - 1) div 2;
  K := f*(f+1) div 2;
  p.i := n - K; p.j := f - p.i;
  Daryl := p;
end;
```

We see that the Kuratowski ordered pairs are *sparsely* distributed among the integers that represent them. And that far more dense implementations are possible. This turns out to be a common feature with common set theory.

Von Neumann successors

Let's skim quickly through the remaining chapters of Halmos' book, until we arrive at chapter XI: Numbers. And pause there for a moment, at last.
An essential difference between standard set theory and an implementable set theory is that features of the latter are far more "dense". We have seen this before with ordered pairs. Let's see how it is working with the so-called von Neumann ordinals. They are created in Halmos' book by:

$$x^+ = x \cup \{x\}$$

where $^+$ is the so-called successor function. In order to avoid confusion with our own successor function, previously named the *Binary successor*, we shall denote $x \cup \{x\}$ as the *von Neumann successor*.
Start with the empty set.

$$\begin{aligned}
\{\} &= 0 \\
\{\} \cup \{0\} = \{0\} &= 0 + 2^0 = 1 \\
\{0\} \cup \{1\} = \{0,1\} &= 1 + 2^1 = 3 \\
\{0,1\} \cup \{3\} = \{0,1,3\} &= 3 + 2^3 = 11 \\
\{0,1,3\} \cup \{11\} = \{0,1,3,11\} &= 11 + 2^{11} = 2059
\end{aligned}$$

But:

$$\{0,1,3,11\} \cup \{2059\} = \{0,1,3,11,2059\} = 2059 + 2^{2059} = ??$$

And it's already "impossible" to find the next von Neumann ordinal, that is: within the scope of common 32 bits integer (Delphi Pascal) numbers. Meaning that these "ordinals" are also very sparsely distributed among the integers that represent implementable sets, even more sparsely than ordered pairs. But isn't that just the big trick? In standard set theory, the ordinals are defined as an extremely sparse "subset" of "all" sets. It feeds the suggestion that set theory is *thus* far more general than number theory and that set theory therefore can serve as a foundation for number theory.
Quite another picture is emerging with implementable set theory. Each of our implementable (that is: hereditarily finite) sets uniquely corresponds with a

natural number (and $\{\} = 0$). And each natural number uniquely corresponds with an implementable set. Thus our theory of sets does *not* contain more sets than it contains numbers. Therefore it's *not* suggested that our implementable set theory can serve as a foundation for the whole of mathematics. It's rather suggested that sets are based upon numbers. And that a good old statement still holds: 'Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk' (Leopold Kronecker - 1886). It's the *whole numbers* that are foundational for Mathematics, not sets.
The theory of finite ordinals could be labeled as von Neumann's left hand. The common von Neumann computer architecture could be labeled as his right hand. Clearly von Neumann's left hand didn't know what his right hand was doing. Any serious implementation of the former into the latter would be so tremendously inefficient that all computation with naturals would become impossible.

Axiom of Infinity

The Axiom of Infinity: *There exists a set N containing* 0 *and with each element its successor.* Where *successor* is the von Neumann successor.
The axiom of Infinity, without doubt, is the most suspect of all ZFC axioms. It has no counterpart, anyway, in our implementable set theory / Abian's "simple model" of sets / the realm of hereditarily finite sets. Abian says that the Axiom of Infinity, therefore, is *independent* of the other axioms. But I think somewhat stronger comments can be made. An obvious remark is that, with Implementable Set Theory, *there is no need* anymore to create a set of all Naturals, because they already *have* a dominant place in the system.
As a fresh start, we could replace the (common set theoretic) von Neumann successor by our own Binary successor. Surely doing such a thing will challenge the unexpected and I am not qualified enough to oversee all of the possible consequences. An example of the latter is the *transitivity* of ordinals.

$$11 = \{3, 1, 0\} \quad \Longrightarrow \quad 3 = \{1, 0\} \subset 11 \quad \text{and} \quad 1 = \{0\} \subset 11 \quad \text{and} \quad 0 = \{\} \subset 11$$

So 11 is indeed transitive. But:

$$6 = \{1, 2\} \quad \Longrightarrow \quad 2 = \{1\} \subset 6 \quad \text{and} \quad 1 = \{0\} \not\subset 6$$

So 6 is *not* transitive.
But anyway, The axiom of Infinity sounds very much the same with a Binary successor. And it will breed an offspring which looks very much the same as with the von Neumann successor, namely, in both cases, the set of "all" Naturals (where it might be that the $\aleph_0$ below is not what you think):

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots, n, \ldots, \aleph_0\}$$

Herein the numbers are the naturals as they are implemented - the binary way - in common digital computers, if with the Binary successor. And they are the finite ordinals, sparsely distributed among the binary naturals, if with the von

Neumann successor. Though there must exist a one-to-one mapping, a *bijection* between the binary naturals and the von Neumann "naturals", it is clear at the same time that the latter form a proper subset of the former. With other words: the two sets of naturals are not identical. This means that the two theories, common set theory on one hand, implementable set theory on the other hand, must be clearly distinguished from each other, as soon as the definition of the Naturals comes into play.
It can be demonstrated, in more than one way, that there are infinitely many implementable sets. Suppose namely that the number of implementable sets is finite, then they can be collected into a set. But the latter set is not an element of itself. This proves by contradiction that the number of implementable sets certainly is *not finite*. Hence infinite, in very much the same way as there are 'infinitely many', meaning 'not finitely many' prime numbers.
A variation upon the same theme employs the idea that implementable sets are equivalent with numbers. Therefore:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots, n, \ldots, \aleph_0\} = 2^{\aleph_0+1} - 1$$

The number on the right hand side is not in the set on the left hand side. Thus the set of *all* Naturals does not exist with Implementable Set Theory.
We have seen that the axiom of Specification - also called the Subset axiom - is a *theorem* within the realm of Implementable Set Theory. This means that we do not have to conform with it. Now try to define the following set, while using *unrestricted* specification:

$$\{x : x \notin x\}$$

We must *prove* that such an attempt may be successful, because it's not covered by the Subset axiom / theorem. By the axiom of Foundation we know that no set is an element of itself, so this thing must be the set of all implementable sets. But then the set should contain itself as an element, which is impossible. This contradiction - much like the infamous paradox by Russell - proves that the set $\{x : x \notin x\}$, and hence the set of all implementable sets: does not exist. Which is equivalent with the above statement that the completed set of all Naturals does not exist within Implementable Set Theory. This means that the axiom of Infinity is not so much consistent or inconsistent, but rather *completely in vain* with Implementable Set Theory.
Corollary. A final moral of this story is that, any time and any place Actual Infinity is employed, Russell's Paradox is just around the corner.

Disclaimers

Anything free comes without referee :-(
My English may be better than your Dutch :-)