CSCI 3155: Recitation # 7
Set Theory for Understanding Programming Languages

Christoph Reichenbach <reichenb@colorado.edu>

October 16, 2003

1 Basic Concepts

As noted in the previous worksheet, Set Theory is a fundamental part of modern mathematics; it also is one of the most frequently used mathematical theories in Computer Science. To a limited extent, we've already observed its usefulness in our discussions on grammars and the languages they generate; another example where they are useful is in modelling data types.

There are many ways to explain set theory; in here, we choose the approach of defining it (informally) through *binary relations*.

1.1 Relations

A binary *relation* between two mathematical objects is usually written as an infix symbol, such as $(<)$. For example, for the mathematical objects 1 and 2, we can validly state $1 < 2$; however, the claim $2 < 1$ would be false. We call such a relation "binary" because it is between precisely two objects; while it is easily possible to define relations of arbitrary arity (meaning that they relate arbitrarily many mathematical objects to each other), we will stick to binary ones here.

Set Theory provides one fundamental binary relation it is centered on: The "is-element-of" relation, usually written $(\in)$. For example, if $\mathbb{N}$ is the set of natural numbers, we can validly claim $1 \in \mathbb{N}$, whereas $\textit{apple} \in \mathbb{N}$ would not be true.

Another example would be a set $\mathcal{T}$ with the following defining property:

$$x \in \mathcal{T} \text{ if and only if } x \in \mathbb{N} \text{ and } 0 \leq x \leq 9$$

and now discuss whether elements are contained in it, e.g. whether $7 \in \mathcal{T}$ (definitely true) or $11 \in \mathcal{T}$ (definitely not true). In some sets, such as the set $\mathbb{P}$ of all prime numbers in existance (an infinite set), actually checking whether this is true is very hard computationally; we know that $5 \in \mathbb{P}$, but what about

147710487433? Worse, there are some infinite sets for which checking membership is impossible in the general case.

Set theory also makes use of another binary relation, the "is-subset-of" relation[1], written ($\subseteq$). We can define this as follows[2]:

$$S \subseteq T \text{ if and only if, for any } x \in S,\ x \in T \text{ holds.}$$

So, considering our earlier examples, we can now state the following:

- $\mathbb{P} \subseteq \mathbb{N}$
- $\mathcal{T} \subseteq \mathbb{N}$
- $\mathbb{N} \subseteq \mathbb{N}$
- $\mathcal{T} \not\subseteq \mathbb{P}$

We usually strike through relations to explain that they do not hold, e.g. $4 \notin \mathbb{P}$.

In some cases, it may be useful to talk about the equality of two sets S and T. The definition for set equality is as follows:

$$S = T \iff (S \subseteq T) \text{ and } (T \subseteq S)$$

(where $A \iff B$ means that A is true if and only if B is true, in turn meaning that it also is false if and only if B is false.)

1.1.1 Summary

We have introduced the binary relations ($\in$) and ($\subseteq$), and their inverted counterparts, ($\notin$) and ($\not\subseteq$). From their definition, we see that, for any two mathematical objects A and B, we always have exactly one of $A \in B$ or $A \notin B$, but not both at the same time. The same holds for $A \subseteq B$ and $A \not\subseteq B$. Furthermore, we have defined set equality ($=$) as mutual set inclusion.

1.1.2 Relations to data types

We can think of certain sets as being mathematical models for data types. For example, some languages, such as Haskell, allow integral numbers of arbitrary size to be represented; for these, we can think of $\mathbb{Z}$, the set of all integral numbers (including the non-negative ones), as being a good model. The general approach for a given type is to determine the set of all values a variable of that type can take, and use this as the type's mathematical model.

As an example, the subrange type `C3 = [0 TO 3]` can be modelled by a set $C_3 = \{0, 1, 2, 3\}$; the subrange type `C1 = [0 TO 1]` by the set $C_1 = \{0, 1\}$

[1]A more precise name would be "is-subset-of-or-equal-to". There also is a relation for strict or true subsets which excludes the set itself, but we do not consider this here.

[2]While there are compact mathematical languages to express statemtents like the one here, it turns out that almost anything we can unambiguously write in English can be translated into the more popular of these.

(subsection 1.2.1 explains this notation for those who are unfamiliar with curly braces).

As we can see, $C_1 \subseteq C_3$– translated back into programming languages, this means that any value of type `C1` is also a value of type `C3`. As it turns out, this works in a much more general way, so that $A \subseteq B$ for any A and B means that the types they model are in a subtype relationship (please refer to the handout on subtyping for details).

1.2 Set Construction

We have discussed three ways to talk about sets, but one important thing is missing: We have not discussed how sets can actually be constructed. Set Theory defines a number of ways in which this can be done, but its axioms only give us one pre-existing set:

$$\emptyset = \{\}$$

The empty set can be written in two ways; its defining properties are that, for whichever X we pick out of the universe of mathematical objects, the following properties hold:

- $X \notin \emptyset$
- $X \subseteq \emptyset$ in precisely one case, namely if $X = \emptyset$.

Beyond that, there are a number of acceptable methods to construct sets, which can be given arbitrary names (much as a record type be introduced in a programming language as soon as the need for it arises).

1.2.1 Writing down all elements

The easiest way to write down sets is to list all of their elements explicitly:

$$\mathit{lunch} := \{\mathit{steak}, \mathit{potatoes}, \mathit{vegetables}\}$$

Note that we use ":=", not "=", to define this set. "=" is also used for this purpose on occasion, but writing "$A := B$" is generally considered to be the "more correct" way to define A in terms of B.

In the above definition, the set *lunch* has three elements, listed explicitly. But this does not indicate any special order about these elements– sets are completely unordered, meaning that, with

$$\mathit{lunch}' := \{\mathit{vegetables}, \mathit{steak}, \mathit{potatoes}\}$$

$$\mathit{lunch}'' := \{\mathit{potatoes}, \mathit{steak}, \mathit{vegetables}\}$$

it is true that $\mathit{lunch}' = \mathit{lunch}$, and also that $\mathit{lunch}'' = \mathit{lunch}$, and thus $\mathit{lunch}' = \mathit{lunch}''$.

Writing down all set elements has one notational disadvantage, though: It seems as if we could include an element in a set more than once, i.e. $S =$

$\{1, 1, 1\}$ or similar. This does not make mathematical sense– the only thing we know about this set is that $1 \in S$ (and that nothing else is in there); the "is-element-of" relation does not yield any mechanism for describing "numbers of occurences"[3]. As such, there is no such thing as a list that contains an element "more than once"; either the element is contained, or it is not.

Note the strong correspondence to enumeration types here– for any enumeration type, we also write down explicitly all the values we want variables of that type to be able to assume as well, e.g. we could define

```
TYPE lunch_type = {vegetables, steak, potatoes};
```

in Modula-3, and use one of the sets above as a model for `lunch_type`.

1.2.2 Comprehending sets

One very powerful way for set construction is by restriction of existing sets[4]. An example of this is the following:

$$\mathcal{T} := \{x | x \in \mathbb{N}, x < 10\}$$

which gives us another way to define the set containing the ten smallest natural numbers. This approach is also called *set comprehension*; it works as follows:

For $\{X | c_1, \ldots c_n\}$, the set it describes are all X for which all of the conditions $c_1, \ldots c_n$ are true. In our example above, X is just the variable x. If we did not have any conditions, x could be any mathematical object; however, we restrict it in two ways (note that order does not matter for conditions, either):

- $x \in \mathbb{N}$: x can only be one of the elements of $\mathbb{N}$, i.e. the relation $x \in \mathbb{N}$ must hold.
- $x < 10$: x must be less than ten, i.e. the relation $x < 10$ must hold.

The second restriction is an example of using relations we know from outside of set theory (arithmetics, in this case); in general, we can pick any condition on x that can be defined concisely.

Note that this gives us a great model for the type `[0 TO 9]`; more generally, we see that, for a subrange type `T = [a TO b]`, we can construct a model for it as

$$\mathcal{M}_{\mathtt{T}} := \{x | x \in \mathbb{Z}, a \leq x \leq b\}$$

Furthermore, this also tells us what our model in the case of $a > b$ is: The empty set.

[3]If these are of relevance, *lists* or *multisets*, both of which can be expressed as complicated sets, can be used.

[4]Set Theory does not define any pre-existing sets other than the empty one. However, legions of talented mathematicians have spent an entire century defining sets based on the axioms Set Theory gave them, so we need not worry about how the sets of natural numbers and rational numbers can be constructed.

Getting back to set comprehension in general: Recall that we generalised a set comprehension expression as $\{X|c_1,\ldots,c_n\}$. Now, X does not have to be just a variable; it can be any mathematical expression. For example, it could be

$$T' := \{x^2|x \in \mathbb{N}, x < 10\}$$

which would be the set of the square numbers of all natural numbers less than 10. The restriction $x < 10$ only restricts x, not x^2, meaning that the resulting set really has ten elements. If we wanted to only get all square numbers less than ten (from natural numbers), we could write the following:

$$T'' := \{x^2|x \in \mathbb{N}, x^2 < 10\}$$

and get

$$T'' := \{0, 1, 4, 9\}$$

Other examples are the following:

1. $\{x + y|x \in T, y \in T\} = \{0, 1, \ldots, 17, 18\}$
2. $\{x|x \in \mathbb{N}, x^2 - 2x = 0\} = \{0, 2\}$
3. $\{\{x, y\}|x \in \{1, 2\}, y \in \{a, b\}\} = \{\{1, a\}, \{1, b\}, \{2, a\}, \{2, b\}\}$
4. $\{\{x, y\}|x \in \{1, 2\}, y \in \{1, 2\}\} = \{\{1\}, \{2\}, \{1, 2\}\}$

In the last two exapmles, we actually define *sets of sets*. This is a completely valid construction, as sets are mathematical objects and, therefore, may be elements of sets. The very last example again illustrates that sets may not contain elements more than once.

Not all of these are useful models for types. Set Theory is useful for modelling many areas beside types, and this is one of the places where this manifests.

1.2.3 Combining sets

Another way in which sets can be constructed is by taking a number of existing sets and combining them to form a new one. For this, we have a number of operations available, all of which we can define in terms of set comprehension:

- **Intersection sets**

$$A \cap B := \{x|x \in A \text{ and } x \in B\}$$

- **Union sets**

$$A \cup B := \{x|x \in A \text{ or } x \in B\}$$

- **Difference sets**

$$A \setminus B := \{x|x \in A \text{ but } x \notin B\}$$

- **Powersets**
$$\mathcal{P}(A) = 2^A := \{S|S \subseteq A\}$$
- **Separated Union sets**
$$A \uplus B := \{\langle 0, x\rangle | x \in A\} \cup \{\langle 1, x\rangle | x \in B\}$$
- **Product sets**
$$A \times B := \{\langle x, y\rangle | x \in A, y \in B\}$$

The last two definitions make use of *pairs* to allow distinguishing left from right components[5]; this is not a built-in construction of Set Theory. However, to be formally correct, we can define pairs as follows:

$$\langle a, b\rangle := \{\{a\}, \{a, b\}\}$$

which allows us to recover both elements straightforwardly: If the resulting set $S = \{a\}$ only contains one element, both elements of the pair are identical: It is $\langle a, a\rangle$. If there are two elements $\{a, b\} = S$, then we pick the element $x \in S$ such that the other element $y \in S$, with $x \neq y$, is contained in it: $y \subseteq x$ leads us to deconstruct to $\langle y, x\rangle$.

Note that some of these type construction again correspond to constructions we also perform in programming languages:

- **Intersection sets** don't really correspond to any particular type.
- **Union sets** seem similar to union types, but it turns out that separated union sets are really what allows us to model these.
- **Difference sets**, again, are not really useful for us.
- **Powersets** describe a set of all subsets of some existing set. If the set in question is the model of a type, its powerset is an exact model for a set type based on it.
- **Separated Union sets** are models for both tagged and untagged union types. For tagged union types, the extra value we attach (0 and 1 in our example) is the tag. Untagged unions require more sophistication: With untagged types, the actual type of the value we expect matters, but this requires the notion of set-theoretic functions for proper modelling.
- **Product sets** can be used both for arrays and records. Let's assume that I_{32} is a model for a limited 32-bit integer type, and S is a model for string types. Then $I_{32} \times S$ makes up a nice model for a record containing both a 32 bit integer and a string, although it does not include field names. An array of five of these integers could be modelled by $I_{32} \times I_{32} \times I_{32} \times I_{32} \times I_{32}$; in general, arrays of length n of type T can be modelled by $T3^n$ (where we use $\times$ for multiplication).

[5]The construction of separated unions also assumes that natural numbers exist; we explicitly construct these in subsection 2.2.

1.3 Sizes of finite sets

For each set, we can define its size, defined as the number of distinct elements it contains. This is also called the set's *cardinality* and written as follows:

$$\text{card}(S) = \#S = |S|$$

FOr finite sets, we can define cardinality recursively:

$$\begin{array}{rcl} |\emptyset| & = & 0 \\ |(S \cup \{x\})| & = & 1 + |S| \iff x \notin S \end{array}$$

The last requirement, $x \notin S$, is neccessary to guarantee that we really "take x out of" the set we are considering.

The size of a set is quite important for discussing memory requirements of elements of certain sets, when represented physically. For these purposes, the canonic notation for cardinality can be helpful.

2 Advanced reading

The material covered in this section is relevant to the general understanding of sets, but is not very relevant to the use of sets in the field of Programming Languages. However, some of it can be used to model, on a mathematical level, parts of concrete programs; this can be very useful to prove or argue about their properties.

2.1 Sizes of infinite sets

Sizes for infinite sets have also been specified; the size of the set of natural numbers, for example, is denoted by the following *transfinite ordinal* :

$$\aleph_0 \quad := \quad |\mathbb{N}| = |\mathbb{R}|$$

The set of rational numbers and the set of all functions from natural numbers to natural numbers are examples of sizes with cardinalities greater than $\aleph_0$. For programming language practice, these are not of importance[6].

$\aleph_n$, for $0 < n$, is defined as the cardinality of the "smallest" set "bigger" than any set of cardinality $\aleph_{n-1}$. What this means concretely is hard to define without giving concrete ways for comparing set sizes without their cardinality, which is beyond the scope of this document.

[6]There is one exception: The theoretical field of Domain Theory goes to some lengths to ensure that the set of functions it considers is only $\aleph_0$ in size; this is done by constructing functions in such a way that they correspond to only the computable functions.

One common assumption is the following:

$$|2^S| \quad = \quad \aleph_{n+1} \iff |S| = \aleph_n$$

which is called the *Continuum Hypothesis*, and discussed in more detail in mathematical literature.

2.2 Representing natural numbers in Set Theory

In Set Theory, the set of natural numbers is constructed from the empty set as follows:

$$\begin{aligned} 0 &:= \emptyset \\ 1 &:= \{\emptyset\} \\ 2 &:= \{\emptyset, \{\emptyset\}\} \\ &\vdots \end{aligned}$$

i.e. we set zero to be the empty set, and every number above it to be the set containing the set representations of all the numbers arithmetically below it. This allows us to observe the followind identification for all $n, m \in \mathbb{N}$:

$$n \leq m \iff n \subseteq m$$

This is often used as a justification for why we can use natural numbers within the realm of set theory at all.

2.3 Relations as sets

Earlier on, we constructed the basic ideas behind Set Theory by expressing relations. It turns out that another way to represent relations (or *predicates*) is by describing sets. For example, we can describe a set L as follows:

$$L = \{\langle x, y\rangle | x, y \in \mathbb{N}. x < y\}$$

This describes a set of ordered pairs containing $\langle x, y\rangle$ precisely if they are natural numbers, and x is less than y (or *contained* in y, if we recall our definition of the natural numbers). This allows us to test for whether one natural number $a \in \mathbb{N}$ is less than another number $b \in \mathbb{N}$ by doing the following:

$$a < b \iff \langle a, b\rangle \in L$$

Because of this equivalence, set theory takes the stance that $(<)$ and L are identical, i.e. $(<)$ is interpreted as the set L with precisely the properties it describes.

The advantage of this approach is that we can construct relations from certain sets, relations can be subsets of each other, and set constructions in general can be applied to relations as required by concrete situations.

For example, we can now write:

- $(<) \subseteq (\leq)$
- $(\leq) = (<) \cup (=)$
- $(\neq) = \mathbb{N}^2 \setminus (=)$

to discuss or define binary relations on $\mathbb{N}$. This principle extends straightforwardly to relations of arbirary arity.

2.3.1 Properties of binary relations

For binary relations, a large number of common properties have been identified. Below, some of the more popular ones– often used when discussing relations– are listed.

- **Symmetry**: A binary relation R is said to be *symmetric* if and only if $aRb \iff bRa$. An example of such a relation is equality.
- **Reflexivity**: A binary relation R is said to be reflexive if and only if aRa holds for all a it is defined on. Examples for this are $(=)$ and $(\leq)$.
- **Transitivity**: A binary relation R is said to be transitive if, for aRb and bRc, we know that aRc. For example, $(<)$ is transitive; if $a < b$ and $b < c$, we also know that $a < c$.
- **Antisymmetry**: This is the opposite of symmetry: When aRb, it must not hold that bRa. Again, $(<)$ is an example for this.

2.4 Tuples and relations of arbitrary arity

Earlier on, we observed the usefulness of pairs for describing certain set constructions. Recall the construction of such a pair:

$$\langle a, b \rangle := \{\{a\}, \{a, b\}\}$$

This notion is often generalised as follows:

$$\langle x_0, \ldots, x_{n-1}, x_n \rangle := \langle x_n, \langle \ldots \langle x_{n-1}, x \quad \underbrace{\rangle \ldots \rangle}_{n-1 \text{ times}}$$

to allow us to construct *n-tuples* for arbitrary n[7].

This gives rise to relations of arbitrary arity: A set containing exclusively n-tuples can thus be thought of as an n-ary relation.

[7]An alternative is to interpret tuples as functions, similar to how read-only arrays can be thought of as functions with a weird syntax, if we ignore the implementation details.

2.5 Set-theoretical functions

There is a special case of relations which is often examined and used in various contexts, namely *graphs of functions.* These are relations R with the property that, if $\langle x_1, \ldots, x_n, x\rangle, \langle x_1, \ldots, x_n, y\rangle \in R$, then $x = y$, i.e. the last entry of the $n + 1$ tuples making up R is uniquely determined by the other entries.

The reason why we call these kinds of relations *graphs of functions* is that they describe, as a "graph", all of the properties of a certain function, which we can recover from such a relation R as follows:

$$f_R(x_1, \ldots, x_n) := x \iff \langle x_1, \ldots, x_n, x\rangle \in R$$

This notation– name plus '(' plus list of arguments plus ')'– corresponds to the set-theoretical notion of the application of a function to the list of arguments, a syntax copied by many programming languages.

Note that the above equation tells us all there is to know about a function (if we already know the relation it is based on); as such, it is not uncommon to equate function and equation here.

2.5.1 Domain and codomain

In the above equation, we took a relation and recovered a function from it. When doing this for a binary relation, the process is straightforward. However, for relations of higher arity, this raises the question of where we should draw the line between the function parameters and the values "returned" by it; in the above example, we could have reasonably argued that the function might, in fact, be

$$f_R(x_1, \ldots, x_{n-1}) := \langle x_n, x\rangle \iff \langle x_1, \ldots, x_n, x\rangle \in R$$

if the uniqueness criterion mentioned earlier holds for $x_1, \ldots x_{n-1}$ with respect to $\langle x_n, x\rangle$.

Given no further information, we can pick whatever is most appropriate to the problem at hand. In order to distinguish these cases, however, we can– for a given function– explicitly describe the set of values it may take as parameters, and also the set of values it "maps to", i.e. (computationally speaking) the set representing the type of its return value.

For this, let us consider a concrete example, which may also serve to illustrate another way to specify functions:

$$\text{abs}(x) := \begin{cases} -x & \iff x < 0 \\ x & \iff x \geq 0 \end{cases}$$

i.e. the function which computes the absolute value of numbers. If we explicitly restrict this function to integers, we might argue that it maps the set of integers, $\mathbb{Z}$, to the set of natural numbers, $\mathbb{N}$. We write this as follows:

$$\text{abs}\colon \mathbb{Z} \to \mathbb{N}$$

and call $\mathbb{N}$ the *image* of 'abs', and $\mathbb{Z}$ its *inverse image* or *preimage*[8].

2.5.2 Specifying functions

In the previous sections, we saw how functions (also called "maps" or "mappings" in set theory) could be recovered from relations, and witnessed one example of a function specification by case differentiation. Case differentiation uses a syntax inverse to that of programming languages: On the left-hand side we specify the result, whereas the right-hand side contains the condition for achieving it. Unlike programming languages, cases may not overlap; "default" clauses, however, are still allowed, as in the following function:

$$h(x) := \begin{cases} 2 & \iff x = 2 \\ 0 & \text{otherwise} \end{cases}$$

which obeys the following specification:

$$(h(x) \neq 0 \iff x = 2) \text{ and } (h(2) = 2)$$

The number of cases may be arbitrary (as long as they do not overlap); their conditions must be specified in an unambiguous manner, however.

Function definitions may also recurse, as in the following example:

$$x! := \begin{cases} 1 & \iff x = 0 \\ (x-1)! * x & \text{otherwise} \end{cases}$$

(where '!' is the function symbol here, written in postfix notation.) Note that not all of these specifications make sense:

$$f(x) := \begin{cases} f(x) & \iff x > 0 \\ 1 & \text{otherwise} \end{cases}$$

does not define anything meaningful for cases where its argument is greater than zero. THis corresponds to a non-terminating recursive function in programming languages.

If we do not need to distinguish cases, a short form of the above is often appropriate:

$$\text{succ}\colon x \mapsto x + 1$$

is often used instead of

$$\text{succ}(x) := x + 1$$

although both forms are generally acceptable.

It is also possible to write more complex patterns in the argument; for example, we could, equivalently, specify this successor function as follows:

$$\text{succ}(x - 1) := x$$

[8]Other names for these are *domain* and *codomain*; in the above example, $\text{dom(abs)} = \mathbb{Z}$ and $\text{cod(abs)} = \mathbb{N}$ can be used to describe these compactly. This nomenclature stems from category theory and is therefore not quite as popular in set theory.

2.5.3 Function composition

There is one other way for constructing a function, and that is by composing existing functions. Function composition is denoted as $(\circ)$, with $f \circ g$ describing a new function which first applies g to whatever it is applied to, and then f to the result. Put another way:

$$(f \circ g)(x) = f(g(x))$$

The requirement here is merely that the image of g is a subset of the inverse image of f, so that f cannot be passed any values it does not handle[9]. So,

$$\text{if } f\colon B \to C, g\colon A \to B' \text{ and } B \subseteq B', \text{ then } (f \circ g)\colon A \to C$$

2.5.4 Partial maps

In the previous example, we had a number of examples where functions did not make sense for certain parameters. One example is the following inverse of the successor function we described earlier:

$$\text{pred}\colon x + 1 \mapsto x$$

If we only consider this function on the set of natural numbers as its inverse image, then this leads to a burning question: What is pred(0)? Of course, for the natural numbers, there is no number which, when increased by one, yields zero. As such, pred, for the inverse image $\mathbb{N}$, is incomplete; we say that it is a *partial function.* This is written as follows:

$$\text{pred}\colon \mathbb{N} \rightharpoonup \mathbb{N}$$

So, partial functions, we generally replace the arrow "$\rightarrow$" by "$\rightharpoonup$".

Of course, if there are partial functions, there must be functions which are not partial. These are called *total* functions; $\text{succ}\colon \mathbb{N} \to \mathbb{N}$ is one example of such a total function.

2.6 Russell's Paradox

The fundamental question behind each mathematical theory is the following: *Is it consistent?* That is, is there absolutely no way to construct a contradiction between its axioms?

One might think that Set Theory, being the foundation of large parts of mathematics, should better be consistent. However, when exploring this question, Bertrand Russell came across the following set:

$$S = \{A | A \notin A\}$$

[9]This is not a strict requirement, but, if not obeyed, it may yield a *partial function*, described in subsection 2.5.4.

which was a perfectly legal to construct within the axioms of Set Theory, as defined by Frege. This set, which contains all sets not contained within themselves, now gives rise to the following question: Is S actually contained within itself?

Theorem 1. *Set Theory is inconsistent (Russell, 1901).*

Proof. Assume S as defined above.

1. Assume $S \in S$. Then, because $S = \{A | A \notin A\}$, S fails the condition of the set comprehension; as such, $S \notin S$– *contradiction.*
2. Assume $S \notin S$. Then, because $S = \{A | A \notin A\}$, S satisfies the condition of the set comprehension, $S \in S$– *contradiction.*

By exhaustive contradiction, Set Theory cannot be consistent. □

This was, of course, a serious problem. Several solutions were suggested; today, the solution most mathematicians use is defined by the Axiom of Foundation: No set may include itself as an element.

This axiom is one of the axioms of ZF and ZFC, two axiomatic specifications of Set Theory, which we list in Appendix A.

A Very Advanced Reading: A formal definition of Set Theory

Our previous discussion of Set Theory was focussed on an informal discussion of various properties. However, there may still be situations in which some questions cannot be fully answered. These answers can only be given by the *axioms* underlying Set Theory, i.e. the original definitions mathematicians came up with when they defined it.

There are several definitions of Set Theory, but the most popular (and generally accepted) ones are ZF, the Zermelo-Fränkel axiomatisation, and ZFC, which is ZF plus one axiom which is considered "required for some stuff, but too specific for other things".

The axioms are defined in terms of the fundamental relation between sets and elements, ($\in$); in our presentation, we will also make use of ($\subseteq$) for reasons of readability, though:

$$A \subseteq B \iff \forall x \in A . x \in B$$

As in this definition, we make use of *first-order logic* to explain these matters concisely. Readers unfamiliar with first-order logic (FOL) may find appendix B useful and are recommended to have a look at it first.

For any given set S, the following axioms apply:

A.1 Axioms

1. Extensionality:

   $$(A = B) \iff (\forall x . x \in A \iff x \in B)$$

   This merely repeats the rule for set equality we described earlier: Two sets are equal if and only if they are subsets of each other.

2. Axiom of the unordered pair:

   $$\forall a, b . \exists S . \forall x \in S . (x = a) \vee (x = b)$$

   This axiom specifies that, for any objects a and b, we can construct a set that contains only these two objects. Note that we do not allow any concrete correlation between a, b and the set $S = \{a, b\}$; in particular, we are not allowed to explicitly set $a = b = S$ (because of the axiom of foundation).

3. Axiom of the sum set: Assume that T is a set of sets:

   $$T = \{S_0, \ldots, S_n\}$$

   Then $\exists S$ such that

   $$S = \bigcup T = \{x | \exists S \in T . x \in S\}$$

   I.e. given an arbitrary (possibly infinite) number of sets, we can construct a set containing all of the contents of these sets.

4. Axiom of the power set: Assume that S is a set, then

   $$2^S = \mathcal{P}(S) = \{A | A \subseteq S\}$$

   exists.

5. Axiom of the empty set:

   $$\exists \emptyset . \forall x . x \notin \emptyset$$

6. Axiom of infinity: There exists a set S with the following two properties:

   $$\begin{array}{rcl} \emptyset & \in & S \\ \forall x \in S . (x \cup \{x\}) & \in & S . \end{array}$$

   This may seem a little strange at first, but it is nothing other than an axiomatic specification for the construction of natural numbers given in subsection 2.2.

7. Axiom of comprehension: Assume that $A(x)$ is a predicate, i.e. "something" that is true or false depending on the concrete x passed to it. One example would be a first-order logic formula, which could contain other expressions, such as set containment, arithmetic properties etc. Then we can construct a set through set comprehension as follows:

$$S' = \{x|A(x)\}$$

   This was discussed in more detail in subsection 1.2.2.

8. Axiom of replacement: Assume that $A(x, y)$ is a predicate (see above) with two arguments and S is a set, then we can construct

$$S' = \{z|x \in S, A(x, z)\}$$

9. Axiom of foundation/regularity

$$S \neq \emptyset \Rightarrow \exists x.(x \in S.x \cap S = \emptyset)$$

   This rarely instantiated axiom requires that sets have no infinite descending chains, i.e. we cannot build sets in the shape of a Klein bottle: $S = \{S\}$ is not a set according to this axiom, though $S = \{S, \emptyset\}$ is.

A.2 The Axiom of Choice

The tenth axiom makes the difference between ZF and ZFC: In the former, it is omitted, whereas the latter is defined by ZF, i.e. the nine axioms we just described, plus the following:

$$S \neq \emptyset \Rightarrow \exists f_S \colon \{S\} \to S$$

i.e., for any given set, there is a function which maps that set to an element of that set (provided that it is nonempty). This function is generally called the *Choice function*; it allows us, for any set, to pick one element out of it and to operate on that.

This seems like a very natural thing to do, but its arbitrariness– "pick any one" in a situation where we don't know how many there are– may be the prime reason behind why it is not too popular.

B First-Order Logic

There are many books introducing first-order logic; in here, we only give a summary of the syntax (in BNF form) and briefly examine the intuitive meaning of its constructs, namely its formulae:

$$\begin{array}{rcl}
\langle\text{Formula}\rangle & \longrightarrow & \langle\text{Atom}\rangle \\
& & \mid \langle\text{Formula}\rangle \wedge \langle\text{Formula}\rangle \\
& & \mid \langle\text{Formula}\rangle \vee \langle\text{Formula}\rangle \\
& & \mid \langle\text{Formula}\rangle \Rightarrow \langle\text{Formula}\rangle \\
& & \mid \neg\, \langle\text{Formula}\rangle \\
& & \mid \forall\, \langle\text{Var}\rangle\, \mathbf{.}\, \langle\text{Formula}\rangle \\
& & \mid \exists\, \langle\text{Var}\rangle\, \mathbf{.}\, \langle\text{Formula}\rangle \\
& & \mid \mathbf{(}\, \langle\text{Formula}\rangle\, \mathbf{)}
\end{array}$$

…where ⟨Var⟩ is a single variable, and ⟨*Atom*⟩ is either a variable or a more complex term, such as $a > b$ or $x \in S$.

The meanings of the individual kinds of formulae is described below:

- $A \wedge B$: Both A and B must be true for this to hold. For example, $1 < 2 \wedge$ **true** holds.
- $A \vee B$: One of A and B or both of them must be true for this to hold. For example, $1 > 2 \vee 2 > 1$ holds because its second component holds.
- $A \Rightarrow B$: B must be true whenever A is true. If A is not true, B may be true or may not be true. For example, **false** $\Rightarrow (0 = 1)$ holds, because its left-hand part (the so-called "premise") is false, meaning that we don't have to worry about its right-hand side (the "conclusion").
- $\neg A$: A must NOT be true. As an example, ¬**true** does NOT hold.
- $\forall x.A$: No matter what we pick for x, A must always be true. For example, $\forall x.(x \in \mathbb{N} \Rightarrow x \geq x)$ holds because its subformula $(x \in \mathbb{N} \Rightarrow x \geq x)$ holds no matter which x we pick.
- $\exists x.A$: If we look for it long and hard enough, we can find some x which makes A true. As an example, $\exists x.(x \in \mathbb{N} \Rightarrow x = 2^x - 12)$ is true, because we can find $x = 4$ to make it true. On the other hand, $\exists x.(x \in \mathbb{N} \Rightarrow x + 1 = 0)$ is not true, because there is no such x in the set of natural numbers.

We have informally used **true** and **false** above. These "literal values" can be constructed without specifying them as special entities if there exists any variable x, as follows:

$$\begin{array}{rcl}
\textbf{true} & = & x \vee (\neg x) \\
\textbf{false} & = & x \wedge (\neg x)
\end{array}$$

Finally, note that, instead of

$$\forall x.(x \in S \Rightarrow A)$$

we also tend to write

$$\forall x \in S.A$$

for readability (analogously for “$\exists$”). Other abbreviations, such as using “$\forall$” and “$\exists$” on more than one variable at once are common, too.

C Acknowledgements

The author would like to thank Max Horn, Walter van Niftrik and Lars Skovlund for their helpful feedback on this document.