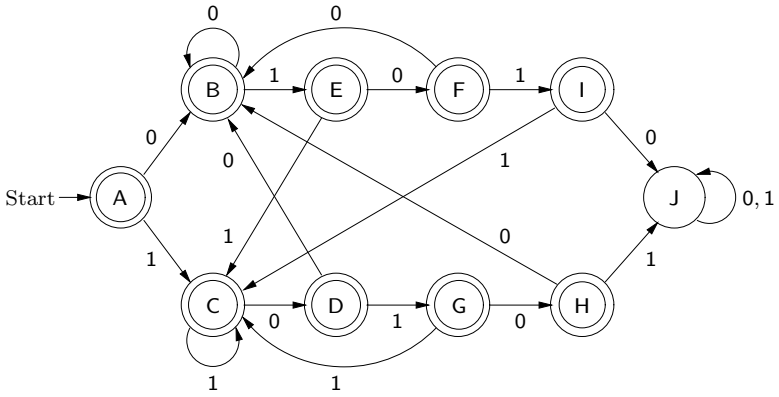
Formal Language Theory

Integrating Experimentation and Proof

Alley Stoughton

Draft of September 2018



Contents

List of Figures

Preface

Background

Since the 1930s, the subject of formal language theory, also known as automata theory, has been developed by computer scientists, linguists and mathematicians. Formal languages (or simply languages) are sets of strings over finite sets of symbols, called alphabets, and various ways of describing such languages have been developed and studied, including regular expressions (which "generate" languages), finite automata (which "accept" languages), grammars (which "generate" languages) and Turing machines (which "accept" languages). For example, the set of identifiers of a given programming language is a formal language—one that can be described by a regular expression or a finite automaton. And, the set of all strings of tokens that are generated by a programming language's grammar is another example of a formal language.

Because of its applications to computer science, most computer science programs offer both undergraduate and graduate courses in this subject. Perhaps the best known applications are to compiler construction. For example, regular expressions and finite automata are used when specifying and implementing lexical analyzers, and grammars are used to specify and implement parsers. Finite automata are used when designing hardware and network protocols. And Turing machines—or other machines/programs of equivalent power—are used to formalize the notion of algorithm, which in turn makes possible the study of what is, and is not, computable.

Formal language theory is largely concerned with algorithms, both ones that are explicitly presented, and ones implicit in theorems that are proved constructively. In typical courses on formal language theory, students apply these algorithms to toy examples by hand, and learn how they are used in applications. Although much can be achieved by a paper-and-pencil approach to the subject, students would obtain a deeper understanding of the subject if they could experiment with the algorithms of formal language theory using computer tools.

Consider, e.g., a typical exercise of a formal language theory class in which students are asked to synthesize a deterministic finite automaton that accepts some language, L. With the paper-and-pencil approach, the student is obliged

to build the machine by hand, and then (hopefully) prove it correct. But, given the right computer tools, another approach would be possible. First, the student could try to express L in terms of simpler languages, making use of various language operations (e.g., union, intersection, difference, concatenation, closure). The student could then synthesize automata accepting the simpler languages, enter these machines into the system, and then combine these machines using operations corresponding to the language operations used to express L. Finally, the resulting machine could be minimized. With some such exercises, a student could solve the exercise in both ways, and could compare the results. Other exercises of this type could only be solved with machine support.

Integrating Experimentation and Proof

To support experimentation with formal languages, I designed and implemented a computer toolset called Forlan [Sto05, Sto08]. Forlan is implemented in the functional programming language Standard ML (SML) [MTHM97, Pau96], a language whose notation and concepts are similar to those of mathematics. Forlan is a library on top of the Standard ML of New Jersey (SML/NJ) implementation of SML [AM91]. It's used interactively, and users are able to extend Forlan by defining SML functions.

In Forlan, the usual objects of formal language theory—finite automata, regular expressions, grammars, labeled paths, parse trees, etc.—are defined as abstract types, and have concrete syntax. Instead of Turing machines, Forlan implements a simple functional programming language of equivalent power, but which has the advantage of being much easier to program in than Turing machines. Programs are also abstract types, and have concrete syntax. Although mainly *not* graphical in nature, Forlan includes the Java program JForlan, a graphical editor for finite automata and regular expression, parse and program trees. It can be invoked directly, or via Forlan.

Numerous algorithms of formal language theory are implemented in Forlan, including conversions between regular expressions and different kinds of automata, the usual operations (e.g., union) on regular expressions, automata and grammars, equivalence testing and minimization of deterministic finite automata, a general parser for grammars, etc. Forlan provides support for regular expression simplification, although the algorithms used are works in progress. It also implements the functional programming language used as a substitute for Turing machines.

This undergraduate-level textbook and Forlan were designed and developed together. I have attempted to keep the conceptual and notational distance between the textbook and toolset as small as possible. The book treats most concepts and algorithms both theoretically, especially using proof, and through experimentation, using Forlan.

In contrast to some books on formal language theory, the book emphasizes

the concrete over the abstract, providing numerous, fully worked-out examples of how regular expressions, finite automata, grammars and programs can be designed and proved correct. In my view, students are most able to learn how to write proofs—and to see the benefit of doing so—if their proofs are about things that they have designed.

I have also attempted to simplify the foundations of the subject, using alternative definitions when needed. E.g., finite automata are defined in such a way that they form a set (as opposed to a proper class), and so that more restricted forms of automata (e.g., deterministic finite automata (DFAs)) are proper subsets of that set. And finite automata are given semantics using labeled paths, from which the traditional δ notation is derived, in the case of DFAs. Furthermore, the book treats the set theoretic foundations of the subject more rigorously than is typical.

Readers of this book are assumed to have significant experience reading and writing informal mathematical proofs, of the kind one finds in mathematics books. This experience could have been gained, e.g., in courses on discrete mathematics, logic or set theory. The book assumes no previous knowledge of Standard ML. In order to understand and extend the implementation of Forlan, though, one must have a good working knowledge of Standard ML, as could be obtained by working through [Pau96] or [Ull98].

Drafts of this book were successfully used at Kansas State University in a semester long, undergraduate course on formal language theory.

Outline of the Book

The book consists of five chapters. Chapter 1, *Mathematical Background*, consists of the material on set theory, induction and recursion, and trees and inductive definitions that is required in the remaining chapters.

In Chapter **??**, *Formal Languages*, we say what symbols, strings, alphabets and (formal) languages are, show how to use various induction principles to prove language equalities, and give an introduction to the Forlan toolset. The remaining three chapters introduce and study more restricted sets of languages.

In Chapter **??**, *Regular Languages*, we study regular expressions and languages, five kinds of finite automata, algorithms for processing and converting between regular expressions and finite automata, properties of regular languages, and applications of regular expressions and finite automata to searching in text files, lexical analysis, and the design of finite state systems.

In Chapter **??**, *Context-free Languages*, we study context-free grammars and languages, algorithms for processing grammars and for converting regular expressions and finite automata to grammars, top-down (recursive descent) parsing, and properties of context-free languages. We prove that the set of context-free languages is a proper superset of the set of regular languages.

Finally, in Chapter **??**, *Recursive and Recursively Enumerable Languages*, we study the functional programming language that we use instead of Turing machines to define the recursive and recursively enumerable languages. We study algorithms for processing programs and for converting grammars to programs, and properties of recursive and recursively enumerable languages. We prove that the set of context-free languages is a proper subset of the set of recursive languages, that the set of recursive languages is a proper subset of the set of recursively enumerable languages, and that there are languages that are not recursively enumerable. Furthermore, we show that there are problems, like the halting problem (the problem of determining whether a program halts when run on a given input), that can't be solved by programs.

Further Reading and Related Work

This book covers most of the material that is typically presented in an undergraduate course on formal language theory. On the other hand, typical textbooks on formal language theory cover much more of the subject than we do. Readers who are interested in learning more, or who would like to be exposed to alternative presentations of some of the material in this book, should consult one of the many fine books on formal language theory, such as [HMU01, Koz97, LP98, Mar91, Lin01].

Neil Jones [Jon97] pioneered the use of a programming language with structured data as an alternative to Turing machines for studying the limits of what is computable. In Chapter **??**, we have followed Jones's approach in some ways. On the other hand, our programming language is functional, not imperative (assignment-oriented), and it has explicit support for the symbols and strings of formal language theory.

The existing formal languages toolsets fit into two categories. In the first category are tools, like JFLAP [BLP$^+$97, HR00, Rod06], Pâté [BLP$^+$97, HR00], the Java Computability Toolkit [RHND99], and Turing's World [BE93], that are graphically oriented and help students work out relatively small examples. The books [Rod06] (on JFLAP) and [Lin01] (an introduction to formal language theory) are intended to be used in conjunction with each other. The second category consists of toolsets that, like Forlan, are embedded in programming languages, and so support sophisticated experimentation with formal languages. Toolsets in this category include Automata [Sut92], Grail+ [RW94, RWYC17], HaLeX [Sar02], Leiß's Automata Library [Lei10] and Vaucanson [LRGS04]. I am not aware of other textbook/toolset packages whose toolsets are members of this second category.

Notes, Exercises and Website

In the "notes" subsections that conclude most sections of the book, I have restricted myself to describing how the book's approach differs from standard practice. Readers interested in the history of the subject can consult [HMU01, Koz97, LP98].

The book contains numerous fully worked-out examples, many of which consist of designing and proving the correctness of regular expressions, finite automata, grammars and programs. Similar exercises, as well as other kinds of exercises, are scattered throughout the book.

The Forlan website

`http://alleystoughton.us/forlan`

contains:

- instructions for downloading and installing the Forlan toolset, and JForlan;
- the Forlan manual;
- instructions for reporting errors or making suggestions; and
- the Forlan distribution, including the source for Forlan and JForlan, as well as the LaTeX source for this book.

Acknowledgments

Leonard Lee and Jessica Sherrill designed and implemented graphical editors for Forlan finite automata (JFA), and regular expression and parse trees (JTR), respectively. Their work was unified and enhanced (of particular note was the addition of support for program trees) by Srinivasa Aditya Uppu, resulting in an initial version of JForlan. Subsequently, Kenton Born carried out a major redevelopment of JForlan, resulting in JForlan Version 1.0. A further revision, by the author, led to JForlan Version 2.0.

It is a pleasure to acknowledge helpful discussions relating to the Forlan project with Eli Fox-Epstein, Brian Howard, Rod Howell, John Hughes, Nathan James, Patrik Jansson, Jace Kohlmeier, Dexter Kozen, Matthew Miller, Aarne Ranta, Ryan Stejskal, Colin Stirling, Lucio Torrico and Lyn Turbak. Much of this work was done while I was employed by Kansas State University, and some of the work was done while I was on sabbatical at the Department of Computing Science of Chalmers University of Technology.

Chapter 1

Mathematical Background

This chapter consists of the material on set theory, induction, trees, inductive definitions and recursion that is required in the remaining chapters.

1.1 Basic Set Theory

In this section, we will cover the material on sets, relations, functions and data structures that will be needed in what follows. Much of this material should be at least partly familiar.

1.1.1 Describing Sets by Listing Their Elements

We write $\emptyset$ for the *empty set*—the set with no elements. Finite sets can be described by listing their elements inside set braces: $\{x_1, \ldots, x_n\}$. E.g., $\{3\}$ is the *singleton set* whose only element is 3, and $\{1, 3, 5, 7\}$ is the set consisting of the first four odd numbers.

1.1.2 Sets of Numbers

We write:

- $\mathbb{N}$ for the set $\{0, 1, \ldots\}$ of all natural numbers;
- $\mathbb{Z}$ for the set $\{\ldots, -1, 0, 1, \ldots\}$ of all integers;
- $\mathbb{R}$ for the set of all real numbers.

Note that, for us, 0 *is* a natural number. This has many pleasant consequences, e.g., that the size of a finite set or the length of a list will always be a natural number.

1.1.3 Relationships between Sets

As usual, we write $x \in Y$ to mean that x is one of the elements (members) of the set Y. Sets A and B are equal ($A = B$) iff (if and only if) they have the same elements, i.e., for all x, $x \in A$ iff $x \in B$.

Suppose A and B are sets. We say that:

- A is a *subset* of B ($A \subseteq B$) iff, for all $x \in A$, $x \in B$;
- A is a *proper subset* of B ($A \subsetneq B$) iff $A \subseteq B$ but $A \neq B$.

In other words: A is a subset of B iff every everything in A is also in B, and A is a proper subset of B iff everything in A is in B, but there is at least one element of B that is not in A.

For example, $\emptyset \subsetneq \mathbb{N}$, $\mathbb{N} \subseteq \mathbb{N}$ and $\mathbb{N} \subsetneq \mathbb{Z}$. The definition of $\subseteq$ gives us the most common way of showing that $A \subseteq B$: we suppose that $x \in A$, and show (with no additional assumptions about x) that $x \in B$. Similarly, if we want to show that $A = B$, it will suffice to show that $A \subseteq B$ and $B \subseteq A$, i.e., that everything in A is in B, and everything in B is in A. Of course, we can also use the usual properties of equality to show set equalities. E.g., if $A = B$ and $B = C$, we have that $A = C$.

Note that, for all sets A, B and C:

- if $A \subseteq B \subseteq C$, then $A \subseteq C$;
- if $A \subseteq B \subsetneq C$, then $A \subsetneq C$;
- if $A \subsetneq B \subseteq C$, then $A \subsetneq C$;
- if $A \subsetneq B \subsetneq C$, then $A \subsetneq C$.

Given sets A and B, we say that:

- A is a *superset* of B ($A \supseteq B$) iff, for all $x \in B$, $x \in A$;
- A is a *proper superset* of B ($A \supsetneq B$) iff $A \supseteq B$ but $A \neq B$.

Of course, for all sets A and B, we have that: $A = B$ iff $A \supseteq B \supseteq A$; and $A \subseteq B$ iff $B \supseteq A$. Furthermore, for all sets A, B and C:

- if $A \supseteq B \supseteq C$, then $A \supseteq C$;
- if $A \supseteq B \supsetneq C$, then $A \supsetneq C$;
- if $A \supsetneq B \supseteq C$, then $A \supsetneq C$;
- if $A \supsetneq B \supsetneq C$, then $A \supsetneq C$.

1.1.4 Set Formation

We will make extensive use of the $\{ \cdots \mid \cdots \}$ notation for forming sets. Let's consider two representative examples of its use.

For the first example, let

$$A = \{ n \mid n \in \mathbb{N} \text{ and } n^2 \geq 20 \} = \{ n \in \mathbb{N} \mid n^2 \geq 20 \}.$$

(where the third of these expressions abbreviates the second one). Here, n is a bound variable and is universally quantified—changing it uniformly to m, for instance, wouldn't change the meaning of A. By the definition of A, we have that, for all n,

$$n \in A \quad \text{iff} \quad n \in \mathbb{N} \text{ and } n^2 \geq 20.$$

Thus, e.g.,

$$5 \in A \quad \text{iff} \quad 5 \in \mathbb{N} \text{ and } 5^2 \geq 20.$$

Since $5 \in \mathbb{N}$ and $5^2 = 25 \geq 20$, it follows that $5 \in A$. On the other hand, $5.5 \notin A$, since $5.5 \notin \mathbb{N}$, and $4 \notin A$, since $4^2 \not\geq 20$.

For the second example, let

$$B = \{ n^3 + m^2 \mid n, m \in \mathbb{N} \text{ and } n, m \geq 1 \}.$$

Note that $n^3 + m^2$ is a term (expression), rather than a variable. The variables n and m are existentially quantified, rather than universally quantified, so that, for all l,

$$\begin{aligned} l \in B \quad &\text{iff} \quad l = n^3 + m^2, \text{ for some } n, m \text{ such that } n, m \in \mathbb{N} \text{ and } n, m \geq 1 \\ &\text{iff} \quad l = n^3 + m^2, \text{ for some } n, m \in \mathbb{N} \text{ such that } n, m \geq 1. \end{aligned}$$

Thus, to show that $9 \in B$, we would have to show that

$$9 = n^3 + m^2 \text{ and } n, m \in \mathbb{N} \text{ and } n, m \geq 1,$$

for some values of n, m. And, this holds, since $9 = 2^3 + 1^2$ and $2, 1 \in \mathbb{N}$ and $2, 1 \geq 1$.

We use set formation in the following definition. Given $n, m \in \mathbb{Z}$, we write $[n : m]$ for $\{ l \in \mathbb{Z} \mid l \geq n \text{ and } l \leq m \}$. Thus $[n : m]$ is all of the integers that are at least n and no more than m. For example, $[-2 : 1]$ is $\{-2, -1, 0, 1\}$ and $[3 : 2]$ is $\emptyset$.

Some uses of the $\{ \cdots \mid \cdots \}$ notation are too "big" to be sets, and instead are *proper classes*. A *class* is a collection of universe elements, and a class is *proper* iff it is not a set. E.g., $\{ A \mid A \text{ is a set and } A \notin A \}$ is a proper class: assuming that it is a set is inconsistent—makes it possible to prove anything. This is the so-called Russell's Paradox. To know that a set formation is valid, it suffices to find a set that includes all the elements in the class being defined. E.g., the above set formations are valid, because the classes being defined are subsets of $\mathbb{N}$.

1.1.5 Operations on Sets

Next, we consider some standard operations on sets. Recall the following operations on sets A and B:

$$\begin{aligned} A \cup B &= \{\, x \mid x \in A \text{ or } x \in B \,\} && \text{(union)} \\ A \cap B &= \{\, x \mid x \in A \text{ and } x \in B \,\} && \text{(intersection)} \\ A - B &= \{\, x \in A \mid x \notin B \,\} && \text{(difference)} \\ A \times B &= \{\, (x, y) \mid x \in A \text{ and } y \in B \,\} && \text{(product)} \\ \mathcal{P}\, A &= \{\, X \mid X \subseteq A \,\} && \text{(power set).} \end{aligned}$$

The axioms of set theory assert that all of these set formations are valid (intersection and difference are obviously valid).

Of course, union and intersection are both commutative and associative ($A \cup B = B \cup A$, $(A \cup B) \cup C = A \cup (B \cup C)$, $A \cap B = B \cap A$ and $(A \cap B) \cap C = A \cap (B \cap C)$, for all sets A, B, C). Furthermore, we have that union is idempotent ($A \cup A = A$, for all sets A), and that $\emptyset$ is the identity for union ($\emptyset \cup A = A = A \cup \emptyset$, for all sets A). Also, intersection is idempotent ($A \cap A = A$, for all sets A), and $\emptyset$ is the zero for intersection ($\emptyset \cap A = \emptyset = A \cap \emptyset$, for all sets A).

It is easy to see that, for all sets X and Y, $X \subseteq X \cup Y$ and $Y \subseteq X \cup Y$. We say that sets X and Y are *disjoint* iff $X \cap Y = \emptyset$, i.e., iff X and Y have nothing in common.

$A - B$ is formed by removing the elements of B from A, if necessary. For example, $\{0, 1, 2\} - \{1, 4\} = \{0, 2\}$. $A \times B$ consists of all ordered pairs (x, y), where x comes from A and y comes from B. For example, $\{0, 1\} \times \{1, 2\} = \{(0, 1), (0, 2), (1, 1), (1, 2)\}$. Remember that an ordered pair (x, y) is different from $\{x, y\}$, the set containing just x and y. In particular, we have that, for all x, x', y, y', $(x, y) = (x', y')$ iff $x = x'$ and $y = y'$, whereas $\{1, 2\} = \{2, 1\}$. If A and B have n and m elements, respectively, for $n, m \in \mathbb{N}$, then $A \times B$ will have nm elements. Finally, $\mathcal{P}\, A$ consists of all of the subsets of A. For example, $\mathcal{P}\, \{0, 1\} = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. If A has n elements, for $n \in \mathbb{N}$, then $\mathcal{P}\, A$ will have 2^n elements.

We let $\times$ associate to the right, so that, e.g., $A \times B \times C = A \times (B \times C)$. And, we abbreviate $(x_1, (x_2, \cdots (x_{n-1}, x_n) \cdots))$ to $(x_1, x_2, \ldots, x_{n-1}, x_n)$, thinking of it as an *ordered n-tuple*. For example $(x, (y, z))$ is abbreviated to (x, y, z), and we think of it as an *ordered triple*.

As an example of a proof involving sets, let's prove the following simple proposition, which says that intersections may be distributed over unions:

Proposition 1.1.1
Suppose A, B and C are sets.

(1) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

(2) $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$.

Proof. We show (1), the proof of (2) being similar. It will suffice to show that $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$.

$(A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C))$ Suppose $x \in A \cap (B \cup C)$. We must show that $x \in (A \cap B) \cup (A \cap C)$. By our assumption, we have that $x \in A$ and $x \in B \cup C$. Since $x \in B \cup C$, there are two cases to consider.

- Suppose $x \in B$. Then $x \in A \cap B \subseteq (A \cap B) \cup (A \cap C)$, so that $x \in (A \cap B) \cup (A \cap C)$.
- Suppose $x \in C$. Then $x \in A \cap C \subseteq (A \cap B) \cup (A \cap C)$, so that $x \in (A \cap B) \cup (A \cap C)$.

$((A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C))$ Suppose $x \in (A \cap B) \cup (A \cap C)$. We must show that $x \in A \cap (B \cup C)$. There are two cases to consider.

- Suppose $x \in A \cap B$. Then $x \in A$ and $x \in B \subseteq B \cup C$, so that $x \in A \cap (B \cup C)$.
- Suppose $x \in A \cap C$. Then $x \in A$ and $x \in C \subseteq B \cup C$, so that $x \in A \cap (B \cup C)$.

□

Exercise 1.1.2
Suppose A, B and C are sets. Prove that union distributes over intersection, i.e., for all sets A, B and C:

(1) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

(2) $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$.

Next, we consider generalized versions of union and intersection that work on sets of sets. If X is a set of sets, then the *generalized union* of X ($\bigcup X$) is

$$\{\, a \mid a \in A, \text{for some } A \in X \,\}.$$

Thus, to show that $a \in \bigcup X$, we must show that a is in at least one element A of X. For example

$$\bigcup \{\{0,1\},\{1,2\},\{2,3\}\} = \{0,1,2,3\} = \{0,1\} \cup \{1,2\} \cup \{2,3\},$$
$$\bigcup \emptyset = \emptyset.$$

(Again, we rely on set theory's axioms to know this set formation is valid.)

If X is a *nonempty* set of sets, then the *generalized intersection* of X ($\bigcap X$) is

$$\{\, a \mid a \in A, \text{for all } A \in X \,\}.$$

Thus, to show that $a \in \bigcap X$, we must show that a is in every element A of X. For example

$$\bigcap\{\{0,1\},\{1,2\},\{2,3\}\} = \emptyset = \{0,1\} \cap \{1,2\} \cap \{2,3\}.$$

If we allowed $\bigcap \emptyset$, then it would contain all elements a of our universe that are in all of the nonexistent elements of $\emptyset$, i.e., it would contain all elements of our universe. But this collection is a proper class, not a set. Let's consider the above reasoning in more detail. Suppose a is a universe element. To prove that $a \in A$, for all $A \in \emptyset$, suppose $A \in \emptyset$. But this is a logical contradiction, from which we may prove anything, in particular our desired conclusion, $a \in A$.

1.1.6 Relations and Functions

Next, we consider relations and functions. A *relation* R is a set of ordered pairs. The *domain* of a relation R (**domain** R) is $\{\, x \mid (x,y) \in R, \text{for some } y \,\}$, and the *range* of R (**range** R) is $\{\, y \mid (x,y) \in R, \text{for some } x \,\}$. We say that R is a *relation from* a set X *to* a set Y iff **domain** $R \subseteq X$ and **range** $R \subseteq Y$, and that R is a *relation on* a set A iff **domain** $R \cup$ **range** $R \subseteq A$. We often write $x\ R\ y$ for $(x,y) \in R$.

Consider the relation

$$R = \{(0,1),(1,2),(0,2)\}.$$

Then, **domain** $R = \{0,1\}$, **range** $R = \{1,2\}$, R is a relation from $\{0,1\}$ to $\{1,2\}$, and R is a relation on $\{0,1,2\}$. Of course, R is also, e.g., a relation between $\mathbb{N}$ and $\mathbb{R}$, as well as relation on $\mathbb{R}$.

We often form relations using set formation. For example, suppose $\phi(x,y)$ is a formula involving variables x and y. Then we can let $R = \{\, (x,y) \mid \phi(x,y) \,\}$, and it is easy to show that, for all x, y, $(x,y) \in R$ iff $\phi(x,y)$.

Given a set A, the *identity relation* on A ($\mathbf{id}_A$) is $\{\, (x,x) \mid x \in A \,\}$. For example, $\mathbf{id}_{\{1,3,5\}}$ is $\{(1,1),(3,3),(5,5)\}$. Given relations R and S, the *composition of* S and R ($S \circ R$) is $\{\, (x,z) \mid (x,y) \in R \text{ and } (y,z) \in S, \text{ for some } y \,\}$. Intuitively, it's the relation formed by starting with a pair $(x,y) \in R$, following it with a pair $(y,z) \in S$ (one that begins where the pair from R left off), and supressing the intermediate value y, leaving us with (x,z). For example, if $R = \{(1,1),(1,2),(2,3)\}$ and $S = \{(2,3),(2,4),(3,4)\}$, then from $(1,2) \in R$ and $(2,3) \in S$, we can conclude $(1,3) \in S \circ R$. There are two more pairs in $S \circ R$, giving us $S \circ R = \{(1,3),(1,4),(2,4)\}$. For all sets A, B and C, and relations R and S, if R is a relation from A to B, and S is a relation from B to C, then $S \circ R$ is a relation from A to C.

It is easy to show that $\circ$ is associative and has the identity relations as its identities:

(1) For all sets A and B, and relations R from A to B, $\mathbf{id}_B \circ R = R = R \circ \mathbf{id}_A$.

(2) For all sets A, B, C and D, and relations R from A to B, S from B to C, and T from C to D, $(T \circ S) \circ R = T \circ (S \circ R)$.

Because of (2), we can write $T \circ S \circ R$, without worrying about how it is parenthesized.

The *inverse* of a relation R (R^{-1}) is the relation $\{ (y, x) \mid (x, y) \in R \}$, i.e., it is the relation obtained by reversing each of the pairs in R. For example, if $R = \{(0, 1), (1, 2), (1, 3)\}$, then the inverse of R is $\{(1, 0), (2, 1), (3, 1)\}$. So for all sets A and B, and relations R, if R is a relation from A to B, then R^{-1} is a relation from B to A.

A relation R is:

- *reflexive on* a set A iff, for all $x \in A$, $(x, x) \in R$;
- *transitive* iff, for all x, y, z, if $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$;
- *symmetric* iff, for all x, y, if $(x, y) \in R$, then $(y, x) \in R$;
- *antisymmetric* iff, for all x, y, if $(x, y) \in R$ and $(y, x) \in R$, then $x = y$;
- *total on* a set A iff, for all $x, y \in A$, either $(x, y) \in R$ or $(y, x) \in R$;
- a *function* iff, for all x, y, z, if $(x, y) \in R$ and $(x, z) \in R$, then $y = z$.

Note that being antisymmetric is *not* the same as not being symmetric.

Suppose, e.g., that $R = \{(0, 1), (1, 2), (0, 2)\}$. Then:

- R is not reflexive on $\{0, 1, 2\}$, since $(0, 0) \notin R$.
- R is transitive, since whenever (x, y) and (y, z) are in R, it follows that $(x, z) \in R$. Since $(0, 1)$ and $(1, 2)$ are in R, we must have that $(0, 2)$ is in R, which is indeed true.
- R is not symmetric, since $(0, 1) \in R$, but $(1, 0) \notin R$.
- R is antisymmetric, since there are no x, y such that (x, y) and (y, x) are both in R. (If we added $(1, 0)$ to R, then R would not be antisymmetric, since then R would contain $(0, 1)$ and $(1, 0)$, but $0 \neq 1$.)
- R is not total on $\{0, 1, 2\}$, since $(0, 0) \notin R$.
- R is not a function, since $(0, 1) \in R$ and $(0, 2) \in R$. Intuitively, given an input of 0, it's not clear whether R's output is 1 or 2.

We say that R is a *total ordering on* a set A iff R is a transitive, antisymmetric and total relation on A. It is easy to see that such an R will also be reflexive on A. Futhermore, if R is a total ordering on A, then R^{-1} is also a total ordering on A.

We often use a symbol like $\leq$ to stand for a total ordering on a set A, which lets us use its mirror image, $\geq$, for its inverse, as well as to write $<$ for the relation on A defined by: for all $x, y \in A$, $x < y$ iff $x \leq y$ but $x \neq y$. $<$ is a *strict total ordering on* A, i.e., a transitive relation on A such that, for all $x, y \in A$, exactly one of $x < y$, $x = y$ and $y < x$ holds. We write $>$ for the inverse of $<$. We can also start with a strict total ordering $<$ on A, and then define a total ordering $\leq$ on A by: $x \leq y$ iff $x < y$ or $x = y$. The relations $\leq$ and $<$ on the natural numbers are examples of such relations.

The relation

$$f = \{(0, 1), (1, 2), (2, 0)\}$$

is a function. We think of it as sending the input 0 to the output 1, the input 1 to the output 2, and the input 2 to the output 0.

If f is a function and $x \in \mathbf{domain}\, f$, we write $f\, x$ for the *application of* f *to* x, i.e., the unique y such that $(x, y) \in f$. We say that f is a *function from* a set X *to* a set Y iff f is a function, $\mathbf{domain}\, f = X$ and $\mathbf{range}\, f \subseteq Y$. We write $X \to Y$ for the set of all functions from X to Y. If A has n elements and B has m elements, for $n, m \in \mathbb{N}$, then $A \to B$ will have m^n elements.

For the f defined above, we have that $f\, 0 = 1$, $f\, 1 = 2$, $f\, 2 = 0$, f is a function from $\{0, 1, 2\}$ to $\{0, 1, 2\}$, and $f \in \{0, 1, 2\} \to \{0, 1, 2\}$. Of course, f is also in $\{0, 1, 2\} \to \mathbb{N}$, but it is not in $\mathbb{N} \to \{0, 1, 2\}$.

We let $\to$ associate to the right and have lower precedence than $\times$, so that, e.g., $A \times B \to C \times D \to E \times F$ means $(A \times B) \to ((C \times D) \to (E \times F))$. An element of this set takes in a pair (a, b) in $A \times B$, and returns a function that takes in a pair (c, d) in $C \times D$, and returns an element of $E \times F$.

Suppose $f, g \in A \to B$. It is easy to show that $f = g$ iff, for all $x \in A$, $f\, x = g\, x$. This is the most common way of showing the equality of functions.

Given a set A, it is easy to see that $\mathbf{id}_A$, the identity relation on A, is a function from A to A, and we call it the *identity function* on A. It is the function that returns its input. Given sets A, B and C, if f is a function from A to B, and g is a function from B to C, then the composition $g \circ f$ of (the relations) g and f is the function h from A to C such that $h\, x = g(f\, x)$, for all $x \in A$. In other words, $g \circ f$ is the function that runs f and then g, in sequence. Because of how composition of relations works, we have that $\circ$ is associative and has the identity functions as its identities:

(1) For all sets A and B, and functions f from A to B, $\mathbf{id}_B \circ f = f = f \circ \mathbf{id}_A$.

(2) For all sets A, B, C and D, and functions f from A to B, g from B to C, and h from C to D, $(h \circ g) \circ f = h \circ (g \circ f)$.

Because of (2), we can write $h \circ g \circ f$, without worrying about how it is parenthesized. It is the function that runs f, then g, then h, in sequence.

Given $f \in X \to Y$ and a subset A of X, we write $f(A)$ for the *image of A under f*, $\{ f\,x \mid x \in A \}$. And given $f \in X \to Y$ and a subset B of Y, we write $f^{-1}(B)$ for the *inverse image of B under f*, $\{ x \in X \mid f\,x \in B \}$. For example, if $f \in \mathbb{N} \to \mathbb{N}$ is the function that doubles its argument, then $f(\{3,5,7\}) = \{6,10,14\}$ and $f^{-1}(\{1,2,3,4\}) = \{1,2\}$.

Given a function f and a set $X \subseteq \mathbf{domain}\, f$, we write $f|X$ for the *restriction of f to X*, $\{ (x,y) \in f \mid x \in X \}$. Hence $\mathbf{domain}(f|X) = X$. For example, if f is the function over $\mathbb{Z}$ that increases its argument by 2, then $f|\mathbb{N}$ is the function over $\mathbb{N}$ that increases its argument by 2. Given a function f and elements x, y of our universe, we write $f[x \mapsto y]$ for the *updating of f to send x to y*, $(f|(\mathbf{domain}\, f - \{x\})) \cup \{(x,y)\}$. This function is the same as f, except that it sends x to y. For example, if $f = \{(0,1),(1,2)\}$, then $f[1 \mapsto 0] = \{(0,1),(1,0)\}$, and $f[2 \mapsto 3] = \{(0,1),(1,2),(2,3)\}$.

We often define a function by saying how an element of its domain is transformed into an element of its range. E.g., we might say that $f \in \mathbb{N} \to \mathbb{Z}$ is the unique function such that, for all $n \in \mathbb{N}$,

$$f\,n = \begin{cases} -(n/2), & \text{if } n \text{ is even,} \\ (n+1)/2, & \text{if } n \text{ is odd.} \end{cases}$$

This is shorthand for saying that f is the set of all (n,m) such that

- $n \in \mathbb{N}$,
- if n is even, then $m = -(n/2)$, and
- if n is odd, then $m = (n+1)/2$.

Then $f\,0 = 0$, $f\,1 = 1$, $f\,2 = -1$, $f\,3 = 2$, $f\,4 = -2$, etc.

Exercise 1.1.3
There are three things wrong with the following "definition"—what are they? Let $f \in \mathbb{N} \to \mathbb{N}$ be the unique function such that, for all $n \in \mathbb{N}$,

$$f\,n = \begin{cases} n-2, & \text{if } n \geq 1 \text{ and } n \leq 10, \\ n+2, & \text{if } n \geq 10. \end{cases}$$

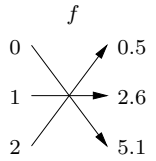
If the domain of f is a product $X_1 \times \cdots \times X_n$, for $n \geq 1$ and sets $X_1, \ldots, X_n$, we often abbreviate $f((x_1, \ldots, x_n))$ to $f(x_1, \ldots, x_n)$. We write $\sharp i_{X_1,\ldots,X_n}$ (or just $\sharp i$, if n and the X_i are clear from the context) for the i-th *projection* function, which selects the i-th component of a tuple, i.e., transforms an input $(x_1, \ldots, x_n)$ to x_i.

1.1.7 Set Cardinality

Next, we see how we can use functions to compare the sizes (or cardinalities) of sets. A *bijection f from* a set X *to* a set Y is a function from X to Y such that, for all $y \in Y$, there is a unique $x \in X$ such that $(x, y) \in f$. For example,

$$f = \{(0, 5.1), (1, 2.6), (2, 0.5)\}$$

is a bijection from $\{0, 1, 2\}$ to $\{0.5, 2.6, 5.1\}$. We can visualize f as a one-to-one correspondence between these sets:



A function f is an *injection* (or is *injective*) iff, for all x, y and z, if $(x, z) \in f$ and $(y, z) \in f$, then $x = y$, i.e., for all $x, y \in \mathbf{domain}\, f$, if $f\, x = f\, y$, then $x = y$. In other words, a function is injective iff it never sends two different elements of its domain to the same element of its range. For example, the function

$$\{(0, 1), (1, 2), (2, 3), (3, 0)\}$$

is injective, but the function

$$\{(0, 1), (1, 2), (2, 1)\}$$

is not injective (both 0 and 2 are sent to 1).

It is easy to see that:

- For all sets A, $\mathbf{id}_A$ is injective.
- For all sets A, B and C, functions f from A to B, and functions g from B to C, if f and g are injective, then so is $g \circ f$.

Exercise 1.1.4
Suppose A and B are sets. Show that for all f, f is a bijection from A to B iff

- f is a function from A to B;
- $\mathbf{range}\, f = B$; and
- f is injective.

Consequently, if f is an injection from A to B, then f is a bijection from A to $\mathbf{range}\, f \subseteq B$.

Exercise 1.1.5
Show that:

(1) For all sets A, $\mathbf{id}_A$ is a bijection from A to A.

(2) For all sets A, B and C, bijections f from A to B, and bijections g from B to C, $g \circ f$ is a bijection from A to C.

(3) For all sets A and B, and bijections f from A to B, f^{-1} is a bijection from B to A.

We say that a set X has the *same size* as a set Y ($X \cong Y$) iff there is a bijection from X to Y. By Exercise 1.1.5, we have that, for all sets A, B, C:

(1) $A \cong A$;

(2) If $A \cong B \cong C$, then $A \cong C$; and

(3) If $A \cong B$, then $B \cong A$.

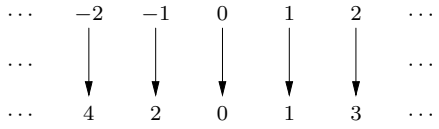
We say that a set X is:

- *finite* iff $X \cong [1 : n]$, for some $n \in \mathbb{N}$;
- *infinite* iff it is not finite;
- *countably infinite* iff $X \cong \mathbb{N}$;
- *countable* iff X is either finite or countably infinite; and
- *uncountable* iff X is not countable.

Every set X has a *size* or *cardinality* ($|X|$) and we have that, for all sets X and Y, $|X| = |Y|$ iff $X \cong Y$. The sizes of finite sets are natural numbers.

We have that:

- The sets $\emptyset$ and $\{0.5, 2.6, 5.1\}$ are finite, and are thus also countable;
- The sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathcal{P}\,\mathbb{N}$ are infinite;
- The set $\mathbb{N}$ is countably infinite, and is thus countable; and
- The set $\mathbb{Z}$ is countably infinite, and is thus countable, because of the existence of the following bijection:



- The sets $\mathbb{R}$ and $\mathcal{P}\,\mathbb{N}$ are uncountable.

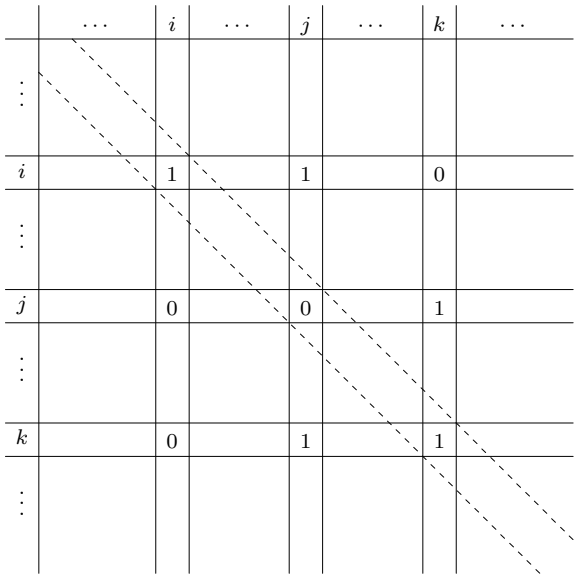


Figure 1.1: Example Diagonalization Table for Cardinality Proof

To prove that $\mathbb{R}$ and $\mathcal{P}\,\mathbb{N}$ are uncountable, we can use an important technique called "diagonalization", which we will see again in Chapter **??**. Let's consider the proof that $\mathcal{P}\,\mathbb{N}$ is uncountable.

We proceed using proof by contradiction. Suppose $\mathcal{P}\,\mathbb{N}$ is countable. If we can obtain a contradiction, it will follow that $\mathcal{P}\,\mathbb{N}$ is uncountable. Since $\mathcal{P}\,\mathbb{N}$ is not finite, it follows that there is a bijection f from $\mathbb{N}$ to $\mathcal{P}\,\mathbb{N}$. Our plan is to define a subset X of $\mathbb{N}$ such that $X \notin \mathbf{range}\, f$, thus obtaining a contradiction, since this will show that f is not a bijection from $\mathbb{N}$ to $\mathcal{P}\,\mathbb{N}$.

Consider the infinite table in which both the rows and the columns are indexed by the elements of $\mathbb{N}$, listed in ascending order, and where a cell (m, n) (m is the row, n is the column) contains 1 iff $m \in f\, n$, and contains 0 iff $m \notin f\, n$. Thus the nth column of this table represents the set $f\, n$ of natural numbers.

Figure 1.1 shows how part of this table might look, where i, j and k are sample elements of $\mathbb{N}$. Because of the table's data, we have, e.g., that $i \in f\, i$ and $j \notin f\, i$.

To define our $X \subseteq \mathbb{N}$, we work our way down the diagonal of the table, putting n into our set just when cell (n, n) of the table is 0, i.e., when $n \notin f\, n$. This will ensure that, for all $n \in \mathbb{N}$, $f\, n \neq X$. With our example table:

- since $i \in f\, i$, but $i \notin X$, we have that $f\, i \neq X$;
- since $j \notin f\, j$, but $j \in X$, we have that $f\, j \neq X$; and
- since $k \in f\, k$, but $k \notin X$, we have that $f\, k \neq X$.

Next, we turn the above ideas into a shorter, but more opaque, proof that:

Proposition 1.1.6
$\mathcal{P}\,\mathbb{N}$ is uncountable.

Proof. Suppose, toward a contradiction, that $\mathcal{P}\,\mathbb{N}$ is countable. Thus, there is a bijection f from $\mathbb{N}$ to $\mathcal{P}\,\mathbb{N}$. Define $X \in \{\, n \in \mathbb{N} \mid n \notin f\, n \,\}$, so that $X \in \mathcal{P}\,\mathbb{N}$. By the definition of f, it follows that $X = f\, n$, for some $n \in \mathbb{N}$. There are two cases to consider.

- Suppose $n \in X$. By the definition of X, it follows that $n \notin f\, n = X$—contradiction.
- Suppose $n \notin X$. Because $X = f\, n$, we have that $n \notin f\, n$. Thus, since $n \in \mathbb{N}$ and $n \notin f\, n$, it follows that $n \in X$—contradiction.

Since we obtained a contradiction in both cases, we have an overall contradiction. Thus $\mathcal{P}\,\mathbb{N}$ is uncountable. □

We have seen how bijections may be used to determine whether sets have the same size. But how can we compare the relative sizes of sets, i.e., say whether one set is smaller or larger than another? The answer is to make use of injective functions. We say that a set X is *no bigger* than a set Y ($X \preceq Y$) iff there is an injection from X to Y, i.e., an injective function whose domain is X and whose range is a subset of Y. For example, the injection $\mathbf{id}_{\mathbb{N}}$ shows that $\mathbb{N} \preceq \mathbb{R}$. This definition makes sense, because X is no bigger than Y iff X has the same size as a subset of Y. We say that X is *strictly smaller* than Y iff $X \preceq Y$ and $X \not\cong Y$.

Using our observations about injections, we have that, for all sets A, B and C:

(1) $A \preceq A$;

(2) If $A \preceq B \preceq C$, then $A \preceq C$.

Clearly, for all sets A and B, if $A \cong B$, then $A \preceq B \preceq A$. And, a famous result of set theory, the Schröder-Bernstein Theorem, says that the converse holds, i.e., for all sets A and B, if $A \preceq B \preceq A$, then $A \cong B$. This give us a powerful method for proving that two sets have the same size.

Exercise 1.1.7
Use the Schröder-Bernstein Theorem to show that $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$. Hint: use the following consequence of the Fundamental Theorem of Arithmetic: if two finite, ascending (each element is $\leq$ the next) sequences of prime numbers (natural numbers that are at least 2 and have no divisors other than 1 and themselves) have the same product (the product of the empty sequence is 1), then they are equal.

One of the forms of the Axiom of Choice says that, for all sets A and B, either $A \preceq B$ or $B \preceq A$, i.e., either A is no bigger than B, or B is no bigger than A. Furthermore, the sizes of sets are ordered in such a way that, for all sets A and B, $|A| \leq |B|$ iff $A \preceq B$, which tells us that, given sets A and B, either $|A| \leq |B|$ or $|B| \leq |A|$. Given the above machinery, one can strengthen Proposition 1.1.6 into: for all sets X, X is strictly smaller than $\mathcal{P}\, X$, i.e., $|X| < |\mathcal{P}\, X|$.

1.1.8 Data Structures

We conclude this section by introducing some data structures that are built out of sets. We write **Bool** for the set of booleans, $\{\mathbf{true}, \mathbf{false}\}$. (We can actually let $\mathbf{true} = 1$ and $\mathbf{false} = 0$, although we'll never make use of these equalities.) We define the negation function $\mathbf{not} \in \mathbf{Bool} \to \mathbf{Bool}$ by:

$$\mathbf{not}\,\mathbf{true} = \mathbf{false}, \qquad \mathbf{not}\,\mathbf{false} = \mathbf{true}.$$

And the conjunction and disjunction operations on the booleans are defined by:

$$\begin{aligned} &\mathbf{true}\ \mathbf{and}\ \mathbf{true} = \mathbf{true}, \\ &\mathbf{true}\ \mathbf{and}\ \mathbf{false} = \mathbf{false}\ \mathbf{and}\ \mathbf{true} = \mathbf{false}\ \mathbf{and}\ \mathbf{false} = \mathbf{false}, \end{aligned}$$

and

$$\begin{aligned} &\mathbf{true}\ \mathbf{or}\ \mathbf{true} = \mathbf{true}\ \mathbf{or}\ \mathbf{false} = \mathbf{false}\ \mathbf{or}\ \mathbf{true} = \mathbf{true}, \\ &\mathbf{false}\ \mathbf{or}\ \mathbf{false} = \mathbf{false}. \end{aligned}$$

Given a set X, we write $\mathbf{Option}\, X$ for $\{\mathbf{none}\} \cup \{\, \mathbf{some}\, x \mid x \in X \,\}$, where we define $\mathbf{none} = (0, 0)$ and $\mathbf{some}\, x = (1, x)$, which guarantees that $\mathbf{none} = \mathbf{some}\, x$ can't hold, and that $\mathbf{some}\, x = \mathbf{some}\, y$ only holds when $x = y$. (We won't make use of the particular way we've defined **none** and $\mathbf{some}\, x$.) For example, $\mathbf{Option}\,\mathbf{Bool} = \{\mathbf{none}, \mathbf{some}\,\mathbf{true}, \mathbf{some}\,\mathbf{false}\}$.

The idea is that an element of $\mathbf{Option}\, X$ is an optional element of X. E.g., when a function needs to either return an element of X or indicate that an error has occurred, it could return an element of $\mathbf{Option}\, X$, using **none** to indicate an error, and returning $\mathbf{some}\, x$ to indicate success with value x.

Finally, we consider lists. A *list* is a function with domain $[1 : n]$, for some $n \in \mathbb{N}$. (Recall that $[1 : n]$ is all of the natural numbers that are at least 1 and no more than n.) For example $\emptyset$ is a list, as it is a function with domain $\emptyset = [1 : 0]$. And $\{(1, 3), (2, 5), (3, 7)\}$ is a list, as it is a function with domain $[1 : 3]$. Note that, if x is a list, then $|x|$, the size of the set x, doubles as the *length* of x.

We abbreviate a list $\{(1, x_1), (2, x_2), \ldots, (n, x_n)\}$ to $[x_1, x_2, \ldots, x_n]$. Thus $\emptyset$ and $\{(1, 3), (2, 5), (3, 7)\}$ are abbreviated to $[\,]$ and $[3, 5, 7]$, respectively. If $[x_1, x_2, \ldots, x_n] = [y_1, y_2, \ldots, y_m]$, it is easy to see that $n = m$ and $x_i = y_i$, for all $i \in [1 : n]$.

Given lists f and g, the *concatenation* of f and g ($f \mathbin{@} g$) is the list

$$f \cup \{\, (m + |f|, y) \mid (m, y) \in g \,\}.$$

For example,

$$\begin{aligned}
[2,3] \mathbin{@} [4,5,6] &= \{(1,2),(2,3)\} \mathbin{@} \{(1,4),(2,5),(3,6)\} \\
&= \{(1,2),(2,3)\} \cup \{\, (m+2, y) \mid (m,y) \in \{(1,4),(2,5),(3,6)\} \,\} \\
&= \{(1,2),(2,3)\} \cup \{(1+2,4),(2+2,5),(3+2,6)\} \\
&= \{(1,2),(2,3),(3,4),(4,5),(5,6)\} \\
&= [2,3,4,5,6].
\end{aligned}$$

Given lists f and g, it is easy to see that $|f \mathbin{@} g| = |f| + |g|$. And, given $n \in [1 : |f \mathbin{@} g|]$, we have that

$$(f \mathbin{@} g)\, n = \begin{cases} f\, n, & \text{if } n \in [1 : |f|], \\ g(n - |f|), & \text{if } n > |f|. \end{cases}$$

Using this fact, it is easy to prove that:

- $[\,]$ is the identity for concatenation: for all lists f,

$$[\,] \mathbin{@} f = f = f \mathbin{@} [\,].$$

- Concatenation is associative: for all lists f, g, h,

$$(f \mathbin{@} g) \mathbin{@} h = f \mathbin{@} (g \mathbin{@} h).$$

Because concatenation is associative, we can write $f \mathbin{@} g \mathbin{@} h$ without worrying where the parentheses go.

Given a set X, we write $\mathbf{List}\, X$ for the set of all *X-lists*, i.e., lists whose ranges are subsets of X, i.e., all of whose elements come from X. E.g., $[\,]$ and $[3,5,7]$ are elements of $\mathbf{List}\, \mathbb{N}$, the set of all lists of natural numbers. It is easy to see that $[\,] \in \mathbf{List}\, X$, for all sets X, and that, for all sets X and $f, g \in \mathbf{List}\, X$, $f \mathbin{@} g \in \mathbf{List}\, X$.

1.1.9 Notes

In a traditional treatment of set theory, e.g., [End77], the natural numbers, integers, real numbers and ordered pairs (x, y) are encoded as sets. But for our purposes, it is clearer to supress this detail.

Furthermore, in the traditional approach, $\mathbb{N}$ is not a subset of $\mathbb{Z}$, and $\mathbb{Z}$ is not a subset of $\mathbb{R}$. On the other hand, there are proper subsets of $\mathbb{R}$ corresponding to $\mathbb{N}$ and $\mathbb{Z}$, and these are what we take $\mathbb{N}$ and $\mathbb{Z}$ to be, so that $\mathbb{N} \subsetneq \mathbb{Z} \subsetneq \mathbb{R}$.

Bibliography

[AM91] A. W. Appel and D. B. MacQueen. Standard ML of New Jersey. In *Programming Language Implementation and Logic Programming*, volume 528 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 1991.

[BE93] J. Barwise and J. Etchemendy. *Turing's World 3.0 for Mac: An Introduction to Computability Theory*. Cambridge University Press, 1993.

[BLP+97] A. O. Bilska, K. H. Leider, M. Procopiuc, O. Procopiuc, S. H. Rodger, J. R. Salemme, and E. Tsang. A collection of tools for making automata theory and formal languages come alive. In *Twenty-eighth ACM SIGCSE Technical Symposium on Computer Science Education*, pages 15–19. ACM Press, 1997.

[End77] H. B. Enderton. *Elements of Set Theory*. Academic Press, 1977.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, second edition, 2001.

[HR00] T. Hung and S. H. Rodger. Increasing visualization and interaction in the automata theory course. In *Thirty-first ACM SIGCSE Technical Symposium on Computer Science Education*, pages 6–10. ACM Press, 2000.

[Jon97] N. J. Jones. *Computability and Complexity: From a Programming Perspective*. The MIT Press, 1997.

[Koz97] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.

[Lei10] H. Leiß. The Automata Library. `http://www.cis.uni-muenchen.de/~leiss/sml-automata.html`, 2010.

[Lin01] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett Publishers, 2001.

[LP98] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, second edition, 1998.

[LRGS04] S. Lombardy, Y. Régis-Gianas, and J. Sakarovitch. Introducing vaucanson. *Theoretical Computer Science*, 328:77–96, 2004.

[Mar91] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw Hill, second edition, 1991.

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML—Revised 1997*. The MIT Press, 1997.

[Pau96] L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, second edition, 1996.

[RHND99] M. B. Robinson, J. A. Hamshar, J. E. Novillo, and A. T. Duchowski. A Java-based tool for reasoning about models of computation through simulating finite automata and turing machines. In *Thirtieth ACM SIGCSE Technical Symposium on Computer Science Education*, pages 105–109. ACM Press, 1999.

[Rod06] S. H. Rodger. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett Publishers, 2006.

[RW94] D. Raymond and D. Wood. Grail: A C++ library for automata and expressions. *Journal of Symbolic Computation*, 17:341–350, 1994.

[RWYC17] D. Raymond, D. Wood, S. Yu, and C. Câmpeanu. Grail+: A symbolic computation environment for finite-state machines, regular expressions, and finite languages. `http://www.csit.upei.ca/~ccampeanu/Grail/`, 2017.

[Sar02] J. Saraiva. HaLeX: A Haskell library to model, manipulate and animate regular languages. In *ACM Workshop on Functional and Declarative Programming in Education (FDPE/PLI'02)*, Pittsburgh, October 2002.

[Sto05] A. Stoughton. Experimenting with formal languages. In *Thirty-sixth ACM SIGCSE Technical Symposium on Computer Science Education*, page 566. ACM Press, 2005.

[Sto08] A. Stoughton. Experimenting with formal languages using Forlan. In *FDPE '08: Proceedings of the 2008 International Workshop on Functional and Declarative Programming in Education*, pages 41–50, New York, NY, USA, 2008. ACM.

[Sut92] K. Sutner. Implementing finite state machines. In N. Dean and G. E. Shannon, editors, *Computational Support for Discrete Mathematics, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 15, pages 347–363. American Mathematical Society, 1992.

[Ull98] J. D. Ullman. *Elements of ML Programming: ML97 Edition*. Prentice Hall, 1998.

Index