MONOGRAPH

An Informal Derivation of the Standard Model of Computation.

written by
Paul Vrbik

Department of Computer Science
Western University
London, Ontario, Canada

© Paul Vrbik 2015

Preface

This book is an adaptation of lectures notes written by the author and given to undergraduate computer-science (CS) students in lieu of a text. The goal of the class was to develop a precise mathematical meaning of computation in order to demonstrate there are propositions falling outside the class of computable problems.

The text is designed to be given over a single semester and can be understood by students with basic training in the rules of inference. The narrative has been tailored to appeal to CS students by framing the problem as '*What is a computer?*', which inevitably leads to the investigation of the notion of 'computation' as an abstract concept.

After an introduction to set theory and brief refresher of logic and proof techniques, the book unfolds in the following way:

1. We first establish 'computation' is fundamentally just membership testing on languages, and strive then to build 'machines' which do just that.

2. Finite State Machines (read-only machines) do not work as they cannot read the simple language $\{a^n b^n : n \in \mathbb{N}\}$, and neither do their non-deterministic counterparts (as there really is no functional distinction between the two).

3. Write-only machines allow for the detection of $\{a^n b^n : n \in \mathbb{N}\}$ (or more generally the context free languages); but not the modestly more complicated $\{a^n b^n c^n : n \in \mathbb{N}\}$.

4. Adding stack memory to Finite State Machines to get Push Down Automata does not fare any better: they are able to read $\{a^n b^n : n \in \mathbb{N}\}$ but $\{a^n b^n c^n : n \in \mathbb{N}\}$ still escapes detection.

5. Finally, we describe machines which read *and* write (Turing machines) and demonstrate these machines are 'maximal' in their detection capability.

6. Unfortunately, a side effect of Turing machines are machines which never terminate. This necessitates a division of problems we can solve with computers into ones which terminate on all inputs (computable ones) and those which terminate on some (if any) inputs (incomputable ones).

Moreover, the text is designed with a realistic assessment of the mathematical background of the typical upper-year CS student (for which this course is usually standard curriculum). Specifically, I assume the students have

1. limited exposure to proofs, and a
2. heightened understanding of algorithms.

Sensitive to this, I have provided a great level of detail in formal proofs, presenting everything in the language of logic and justifying each step as clearly as possible. In cases where proofs are constructive, algorithms are preferred.

Lastly, I have opted to use a deliberately elaborate type of notation, e.g.

$$\left\{ \left(qw_0w_1,\, q''\right) \;:\; \exists w_0, w_1 \in \Sigma;\; (q) \xrightarrow{w_0} (q') \xrightarrow{w_1} (q'') \right\}.$$

This serves to simplify the presentation as a clear correspondence is established between transition state diagrams (the preferred way of defining machines) and the structures they encode.

Finally, into the book I have inserted many side notes with historical anecdotes and general problem-solving strategies. This roots the material in a historical context and exposes students to strategies and practical advice which are sometimes overlooked.

Contents

Preface **ii**

1 Preliminaries **1**
1.1 Set Theory . . . 1
1.1.1 Basics . . . 1
1.1.2 Class Abstraction . . . 8
1.1.3 Operations on Sets . . . 8
1.2 Binary and n-ary Relations . . . 12
1.2.1 Properties of Relations . . . 15
1.2.2 Equivalence Relations . . . 21
1.3 Alphabets, Words, and Languages . . . 23
1.3.1 More Set Constructs . . . 23
1.4 Alphabets and Languages . . . 24
1.4.1 Operations on words . . . 26
1.4.2 Operations on Languages . . . 29
1.5 Proof Methods . . . 33
1.5.1 Direct Proof . . . 33
1.5.2 The Principle of Mathematical Induction . . . 34
1.5.3 Contradiction . . . 36
1.5.4 The Pigeonhole Principle . . . 37
1.6 End of Chapter Exercises . . . 39
1.7 Exercise Solutions . . . 41

2 Finite Automata **43**
2.1 Finite State Machines . . . 43
2.1.1 Complete Machines . . . 48
2.2 The language of a Machine . . . 50
2.2.1 Goto . . . 50
2.2.2 Eventually goes to ($\vdash^*$) . . . 51
2.3 Regular Languages . . . 54
2.3.1 FSM Shortcuts . . . 55

2.3.2 Irregular Languages 57
2.3.3 FSM Scenic Routes 57
2.3.4 Pumping Lemma 59
2.4 Nondeterministic FSM 62
2.4.1 NDFSM → DFSM conversion 65
2.4.2 NDFSM/DFSM equivalence 69
2.5 Properties of NDFSM Languages 69
2.5.1 ε-NDFSM 70
2.5.2 Closure Properties 76
2.6 End of Chapter Exercises 83

3 Other Regular Constructs 84
3.1 Regular Expressions 84
3.1.1 Kleene's Theorem 86
3.2 Regular Grammars 99
3.2.1 Linear Grammar Machines 99
3.2.2 The language of a LGM 102

4 Context Free Grammars 105
4.1 Introduction 105
4.2 Context Free Grammars 107
4.3 CFG Language Proofs 109
4.4 CFG Simplificitation 112
4.4.1 Terminating Symbols 112
4.4.2 Reachable Symbols 114
4.4.3 Empty productions 114
4.4.4 Reduction 116
4.4.5 ε-removal 118
4.5 Chompsky Normal Form 120
4.5.1 Unit Production Removal 121
4.5.2 Long Production Removal 122
4.5.3 Converting to CNF 123

5 Pushdown Automata 128
5.1 Preliminaries 128
5.1.1 The Stack 128
5.2 Push Down Automata 131
5.2.1 Using A Stack 131
5.2.2 Formalizing PDAs 132
5.2.3 Acceptance by PDA 135

5.3 Deterministic pdas 136
5.3.1 Closure Properties of DPDA 138
5.4 Context Free Pumping Lemma 139
5.4.1 Maximum Yields 141

6 Turing Machines 143
6.1 Preliminaries 143
6.2 Formalizing Turing Machines 145
6.2.1 Configurations 147
6.2.2 Moving The Read Head 148
6.2.3 GOTO 148
6.3 Recursive and Recursively Enumerable Languages 148
6.4 Computing with Turing Machines 150
6.5 Recursive Functions 153
6.5.1 Representing $\mathbb{N}$ 154
6.5.2 Primitive Recursive Functions 156
6.6 μ-recursion 160
6.7 Undecidable Problems 162
6.7.1 The Halting Problem 162
6.7.2 Reduction to the Halting Problem 163

Chapter 1

Preliminaries

"Begin at the beginning," the King said, gravely, "and go on till you come to an end; then stop."

– Lewis Carroll, *Alice in Wonderland*

Sets, multisets, sequences, functions, relations, proof techniques, alphabets, words, and languages.

§1.1 Set Theory

History of set theory. Cantor, Dedekind, axiom of choice, construction of the integers.

§Basics

We start with a set.

Definition 1 (set)**.** A Set, in the mathematical sense, is a *finite* or *infinite* collection of *unordered* and *distinct* objects.

Anything surrounded by curly braces '{ }' is a set.

Example 1. A set of integers.

$$A = \{3, 8, 9, 10, 42, -3\} .$$

Definition 2. A set's cardinality (denoted by '| |' or '#') is the number of elements the set contains (finite or otherwise).

Example 2. A has cardinality 6.

$$|A| = |\{3, 8, 9, 10, 42, -3\}| \qquad \#A = \#\{3, 8, 9, 10, 42, -3\}$$

$= 6$ $= 6$

Notation. There are standard sets with fixed names. We use the following:

1. the NATURAL NUMBERS: $\mathbb{N} = \{0, 1, 2, \ldots\}$;
2. the WHOLE NUMBERS: $\mathbb{N}^{>0} = \{1, 2, \ldots\}$;
3. the INTEGERS: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$;[i] and
4. the RATIONAL NUMBERS: $\mathbb{Q} = \left\{\frac{a}{b} : a, b \in \mathbb{Z} \wedge b \neq 0\right\}$.

The most concise, or perhaps only, way to work with sets is to express everything with a FORMAL LANGUAGE of logical symbols. Let us quickly review these symbols taking for granted the definitions and notions of implication ($\implies$ and $\iff$) as well as the meaning of 'or' and 'and' ($\vee$ and $\wedge$).

Definition 3 (Mapping)**.** A mapping 'connects' elements of one set with another. Writing

$$\begin{aligned} M : A &\to B \\ a &\mapsto b. \end{aligned}$$

expresses: M maps $a \in A$ to $b \in B$.

Since functions are also maps—for instance, $f(x) = x^2$ can be given as the mapping:

$$\begin{aligned} f : \mathbb{Z} &\to \mathbb{Z} \\ x &\mapsto x^2 \end{aligned}$$

—the term 'map' and 'function' are sometimes used interchangeably.

When $B = \{\text{true, false}\} = \{\top, \bot\}$ in Definition 3 the map is called a PREDICATE.

Definition 4 (Predicate)**.** An operator of logic that returns true ($\top$) or false ($\bot$) over some universe (e.g. $\mathbb{Z}$, days of the week, or reserved words).

Example 3. A predicate given by

$$P : \mathbb{Z} \to \{\top, \bot\}$$

---

[i] $\mathbb{Z}$ because the German word for 'number' is 'Zahlen'

$$P : x \mapsto \begin{cases} \top & \text{if } x \text{ is prime} \\ \bot & \text{otherwise} \end{cases}$$

evaluates to true only when x is a prime number:

$$P(7) = \top \qquad P(8) = \bot \qquad P(101) = \top.$$

Sometimes a predicate is used so often as to merit a special symbol (mathematicians are, by necessity, inherently lazy when writing things down). There are many of these in set theory.

Definition 5 (Element)**.** The symbol $\in$, read 'is an element of', is a predicate with definition

$$\begin{aligned} \in &: (\text{element}, \text{Universe}) \to \{\top, \bot\} \\ \in &: (e, U) \mapsto e \text{ is an element of } U \end{aligned}$$

(Note: The symbol ε, which will later denote the empty word, should *not* be used for set inclusion.)

Example 4. Over the universe of *even* numbers $E = \{2, 4, 6, \ldots\}$ we deduce

$$\begin{aligned} 2 \in E \iff \in (2, E) &\iff 2 \text{ is an element of } \{2, 4, 6, \ldots\} \\ &\iff \top \end{aligned}$$

and

$$\begin{aligned} 17 \in E \iff \in (17, E) &\iff 17 \text{ is an element of } \{2, 4, 6, \ldots\} \\ &\iff \bot \end{aligned}$$

which can be abbreviated as $2 \in E$ and $17 \notin E$ (both evaluate to true).

Example 4 utilises a widely employed short form for invoking binary mappings (those mappings taking two inputs to one). Consider the addition function which takes two integers and maps them to their sum

$$\begin{aligned} + &: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \\ + &: (x, y) \mapsto x + y. \end{aligned}$$

It is universally understood that $2 + 5$ is a short form for $+(2, 5)$. When functions and mappings on sets are defined, keep it in the back of your mind that we are applying binary mappings in this way.

Definition 6 (Existence)**.** The logical statement '*there exists*' (alternatively '*there is*') is denoted by $\exists$. It is used to express that there is some element of the predicate's universe for which the predicate is true:

$$\exists x \in \{x_0, x_1, \ldots\}\,;\, P(x) \iff P(x_0) \vee P(x_1) \vee \cdots.$$

Example 5. Using the prime test predicate of Example 3:

$$\exists x P(x) = \top$$

when P's universe is $\mathbb{Z}$ (in fact $\mathbb{Z}$ contains *all* the primes). *But* when P's universe is $\{4, 6, 8, \ldots\}$

$$\exists x P(x) = \bot \iff \neg\exists x P(x)$$

as no even number—excluding 2—is prime![ii] (The later statement reads 'there is no $x \in \{4, 6, 8, \ldots\}$ for which x is a prime'.)

To address this subtlety it is common to make the universe explicit,

$$\exists x \in \mathbb{Z};\, P(X).$$

Definition 7 (Every)**.** The symbol $\forall$ denotes '*for all*' (alternatively '*for every*', '*for each*') and is called the UNIVERSAL QUANTIFIER. $\forall$ is used to make assertions 'universally' over an entire set:

$$\forall x \in \{x_0, x_1, \ldots\}\,;\, P(x) \iff P(x_0) \wedge P(x_1) \wedge \cdots$$

Proposition 1. For any predicate $P : U \to \{\top, \bot\}$

$$\neg\forall x \in U;\, P(x) \equiv \exists x \in U;\, \neg P(x).$$

In prose: $P(x)$ is *not* true for every $x \in U$ only when there is $x \in U$ for which $P(x)$ is false (and vice versa)

Proof. □

Example 6. Let $Q(x)$ given over $\mathbb{Z}$ be true only when x is divisible by two:

$$Q(x) \iff 2 \,\big|\, x \iff \exists y \in \mathbb{Z};\, 2y = x.$$

It is *not* true that

$$\forall x \in \mathbb{Z};\, Q(x)$$

[ii] There is some disagreement as to whether 2 should be a prime as it is only so by sheer accident.

because

$$\exists x \in \mathbb{Z};\ \neg Q(x).$$

That is to say, there is some $x \in \mathbb{Z}$ (7 for instance) which is *not* divisible by 2.

If we provide another predicate $R(x) \iff 2 \,\big|\, x - 1$ then

$$\forall x \in \mathbb{Z};\ Q(x)\ \vee\ R(x) = \top$$

since it *is* true that every integer is either even or one more than an even (i.e. all integers can be expressed as $2x$ or $2x + 1$).

Now we are finally able to enumerate the basic rules of set theory, of which there are seven, and from which *all* of set theory (and to some extent mathematics) is a consequence of. Important, basic and *unproveable* assumptions of this kind are called 'axioms'. Let us investigate some (but not all) axioms of set theory:

Our first axiom states the condition for two sets, A and B, to be identical.

Axiom 1 (Axiom of Extensionality)**.** If every element of A is also in B (and vice versa) then A and B are equal:

$$\big[\forall x;\ x \in A \iff x \in B\big] \overset{\text{def.}}{\iff} A = B.$$

Example 7. Although seemingly trivial, we deduce two important properties from Axiom 1.

$$\{1, 2, 3\} = \{3, 1, 3, 2, 1\} = \{2, 1, 3, 2, 1, 3, 3\}\,.$$

Sets are not ordered! And. Duplicate elements are ignored!

Exercise 1. Is $\{2, 2, 2, 3, 3, 4\}$ a set? If so, what is its cardinality?

> Exercises are placed liberally throughout. *Practicing is vital to learning mathematics.* Attempt as many as possible. Each can be answered using information that precedes it. For instance, Example 1 and Definition 1.

The notion of set equality can be weakened,

Definition 8 (Subset)**.** A is a subset of B when each element of A is also an element of B,

$$\big[\forall x;\ x \in A \implies x \in B\big] \overset{\text{def.}}{\iff} A \subseteq B$$

and weakened again,

Definition 9 (Proper/Strict subset)**.**

$$A \subset B \overset{\text{def.}}{\iff} \left[[A \subseteq B] \wedge \neg [B \subseteq A]\right].$$

Unfortunately there is some disagreement (mostly cross-culturally and cross-disciplinary) regarding the symbols used to distinguish subsets from proper subsets. Although we use $\subseteq$ and $\subset$, it is handy to know that other people/texts instead use $\subset$ and $\subsetneq$ to distinguish subset and proper subset (i.e. subset but not equal).

Example 8. For $A = \{1, 2, 3\}$, $B = \{3, 1, 2\}$ and $C = \{1, 2, 3, 4\}$

$$A \subseteq A, \qquad A \subseteq B, \qquad A \subseteq C,$$

and

$$\neg [A \subset A], \qquad \neg [A \subset B] \qquad [A \subset C].$$

(In logic everything written should be true. To express something false, state the negation as true.)

Exercise 2. Show $\forall A;\, A \subset A \equiv \bot$.

Proposition 2. $A \subseteq B \wedge B \subseteq A \implies A = B$.

Proof. The premise $A \subseteq B \wedge B \subseteq A$ can be rewritten as

$$\begin{aligned}
&A \subseteq B \wedge B \subseteq A \\
&\iff \left[\forall x;\, x \in A \implies x \in B\right] \wedge \left[\forall x;\, x \in B \implies x \in A\right] && \text{Defn. 8} \\
&\iff \forall x;\, x \in A \implies x \in B \wedge x \in B \implies x \in A \\
&\iff \forall x;\, x \in A \iff x \in B \\
&\iff A = B && \text{Axiom 1}
\end{aligned}$$

(Only statements that are logical consequences of the line before can forgo justification.) ☐

Axiom 2 (Set Existence Axiom)**.** There is at least one set.

$$\exists A : A = A.$$

(Notice this axiom uses '=' from Axiom 1 and thus could not have been first.)

The wonderful (and deliberate) consequence of this axiom is the existence of the empty set $\varnothing$.

Theorem 1. There is a *unique* set (say, $\varnothing$) with no members,

$$\exists! \varnothing \, \forall x; \, x \notin \varnothing.$$

('!' is a short form for 'unique'.)

Proof. Challenge. Unfortunately, this proof (and more proofs to come) are beyond the scope of this course. All proofs labelled 'Challenge' are optional. □

Definition 10 (Empty set)**.** The empty set

$$\varnothing$$

is the unique set with no members.

Distinguish carefully between $\varnothing$ (the empty set) and ϕ/φ (the greek letter 'phi')!

Proposition 3. The empty set satisfies:

1. $\forall x; \, x \notin \varnothing$,
2. $\forall A; \, \varnothing \subseteq A$, and
3. $\forall A; \, A \subseteq \varnothing \implies A = \varnothing$

Proof. □

A natural question to raise is that of the UNIVERSAL SET—the set containing everything (the complement of the empty set).

Intuitively, the Universal set can not exist because it would, paradoxically, contain itself (see Exercise 2). German mathematician Ernst Zermelo (1831–1916) formalized and proved this statement.

Theorem 2 (Russell's Paradox)**.** There is no universal set.

$$\neg \exists A : \forall x; \, x \in A.$$

Proof. Google. □

(The name 'RUSSELL'S PARADOX' is an example of a weird tendency among mathematicians to sometimes name things after the *last* person to publish them, in this case Bertrand Russell).

§Class Abstraction

There are several ways to express a set. We could specify the members outright:

$$A = \{1, 2, 4, 9, 16, \dots\} ;$$

use SET BUILDER NOTATION or CLASS ABSTRACTION by providing a predicate (with implicit universe):

$$\begin{aligned} B &= \{x \text{ `such that' } x \text{ is a prime number}\} \\ &= \{x : x \text{ is a prime number}\} \\ &= \{2, 3, 5, 7, 11, \dots\} , \\ C &= \{x : x \text{ is an english word and also a palindrome}\} \\ &= \{\text{a, dad, mom}, \dots\} , \\ D &= \{x : x \text{ is a positive integer divisible by 3 and less than 10}\} \\ &= \{3, 6, 9\} ; \end{aligned}$$

or, recursively (§ **??**)

1. $2 \in E$,
2. $a, b \in E \iff a \cdot b \in E$.

> Notice the '...' of C are somewhat meaningless and that D is a finite set.

Exercise 3. What is the definition of E using set building notation?

§Operations on Sets

We have already discussed the subset operation on sets and, expectedly, there are more.

The Power Set

Definition 11 (Power set)**.** The power set of A, denoted $\mathscr{P}(A)$, is the set of all subsets of A.

$$\mathscr{P}(A) = \{B : B \subseteq A\} .$$

Example 9. If $A = \{1, 2, 3\}$ then the power set of A is

$$\mathscr{P}(A) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

As $\varnothing \subset A$ for any set A, notice $\varnothing$ is automatically a member of *every* power set—including the power set of itself!

Exercise 4. What is $\mathscr{P}(\varnothing)$? In particular, is $\mathscr{P}(\varnothing) = \varnothing$?

The notion of $\varnothing$ as distinct from $\{\varnothing\}$, is a subtle yet incredibly powerful idea. All objects of mathematics, at their very core, are sets of sets. For instance the natural numbers (as described by Italian mathematician Giuseppe Peano in 1890) are given in this way:

$$\begin{aligned}
0 &= \varnothing \\
1 &= 0 \cup \{0\} = \varnothing \cup \{\varnothing\} \\
2 &= 1 \cup \{1\} = \varnothing \cup \{\varnothing\} \cup \{\varnothing \cup \{\varnothing\}\} \\
&\vdots \\
n+1 &= n \cup \{n\}
\end{aligned}$$

Later we will define addition over these numbers and 'prove'

$$1 + 1 = 2$$

something that is (supposedly) quite difficult.

Some prefer to regard the power set from the perspective of Combinatorics (a branch of mathematics concerned with counting arrangements of discrete objects). Occupiers of this camp would characterize the power set as enumerating the ways one can include/exclude elements.

Example 10. Again let $A = \{1, 2, 3\}$ and consider this encoding of the subsets, where we will regard $\square$ as an element that has been removed.

$$\begin{aligned}
\texttt{000} &= \{\square, \square, \square\} = \{\} = \varnothing \\
\texttt{001} &= \{\square, \square, 3\} = \{3\} \\
\texttt{010} &= \{\square, 2, \square\} = \{2\} \\
\texttt{011} &= \{\square, 2, 3\} = \{2, 3\} \\
\texttt{100} &= \{1, \square, \square\} = \{1\} \\
\texttt{101} &= \{1, \square, 3\} = \{1, 3\}
\end{aligned}$$

$$\texttt{110} = \{1, 2, \square\} = \{1, 2\}$$
$$\texttt{111} = \{1, 2, 3\} = \{1, 2, 3\}.$$

Realizing the encoding is simply the 3-bit binary numbers (of which there are $2^3 = 8$) written in ascending order, and that there is a one-to-one correspondence between these binary encodings and the subsets they encode, we can easily write down an equation for the cardinality of a power set.

Proposition 4. The cardinality of A's power set is $2^{|A|}$,

$$|\mathscr{P}(A)| = 2^{|A|}.$$

(This is likely the motivation for the alternative notation 2^A for the power set of A.)

Proof. Challenge. □

Example 11. The set $B = \{0, 1, \ldots, 299\}$ corresponds to a power set with cardinality 2^{300} (i.e. huge—for comparison, the number of atoms in the known universe is approximately 2^{265}).

Basic Operations on Sets

Let us review the notions of Intersection, Set Difference, and Union. For the sake of brevity, but mostly because what follows is widely known, we eschew a lengthy discussion and assume that those who desire it will sample the literature[1].

Definition 12 (Set difference)**.** 'A without B' written $A \setminus B$ is given

$$A \setminus B = \{z : z \in A \wedge z \notin B\}.$$

Example 12. $\{1, 2, 3\} \setminus \{3, 4, 5\} = \{1, 2\}$.

Proposition 5. $z \in A \setminus B \iff \{z \in A \wedge z \notin B\}$.

Proof. An important lesson about context. □

Definition 13 (Intersection)**.** 'A intersect B' denoted $A \cap B$ is given by

$$A \cap B = \{z : z \in A \wedge z \in B\}.$$

Proposition 6. $z \in A \cap B \iff z \in A \wedge z \in B$.

Proof. □

Example 13. $\{1,2,3,4,5\} \cap \{2,3,4\} = \{2,3,4\}$.

Exercise 5. What is $C \cap \varnothing$?

Definition 14 (Disjoint)**.** We say that A and B are DISJOINT when

$$A \cap B = \varnothing.$$

Example 14. $\{1,2\}$ and $\{3,4\}$ are disjoint. $\{1,2,3\}$ and $\{3,4\}$ are *not* disjoint as $\{1,2,3\} \cap \{3,4\} = \{3\} \neq \varnothing$.

Definition 15 (Union)**.** A 'union' B denoted $A \cup B$ is given by

$$z \in A \cup B \iff (z \in A \vee z \in B).$$

Example 15. If $A = \{a,b,c\}$ and $B = \{b,c,d,e\}$ then $A \cup B = \{a,b,c,d,e\}$.

Exercise 6. Prove

Identity. $X \cup \varnothing = X$,

Associativity. $\{X \cup Y\} \cup Z = X \cup \{Y \cup Z\}$,

Reflexivity. $X \cup X = X$, and

Commutativity. $X \cup Y = Y \cup X$.

of the union operation.

The various parts of the next theorem, which combines $\setminus$, $\cap$, and $\cup$, are the namesake of British mathematician Augustine De Morgan (1806-1871) who also formalized the 'Principal of Mathematical induction' (Theorem 4).

Theorem 3 (De Morgan's Laws)**.** For sets A, B, and C

1. $A \cap \{B \cup C\} = \{A \cap B\} \cup \{A \cap C\}$,
2. $A \cup \{B \cap C\} = \{A \cup B\} \cap \{A \cup C\}$,
3. $A \setminus \{B \cap C\} = \{A \setminus B\} \cap \{A \setminus C\}$, and
4. $A \setminus \{B \cap C\} = \{A \setminus B\} \cap \{A \setminus C\}$.

Recognize these points as being exactly similar to the De Morgan's Laws of logic with the substitutions

$$\neg \leftarrow \setminus \qquad \vee \leftarrow \cup \qquad \wedge \leftarrow \cap$$

Figure 1.2: Indian born British logician Augustine De Morgan was teased as a child because a vision problem in his left eye prevented him from participating in sports. Later, a crater on the moon would be named in his honour.

Proof of 2. □

Proof of 1,2, and 4. Challenge. □

§1.2 Binary and n-ary Relations

Sets define ordered pairs.

Definition 16 (Ordered Pair)**.** The ORDERED PAIR 'x followed by y' is denoted (x, y) and satisfies

$$(x, y) = \{\{x\}, \{x, y\}\}$$

Proposition 7.

$$(x, y) = (A, B) \iff x = A \wedge y = B$$

Proof. Challenge. □

Example 16. $(2, 3)$ and $(3, 2)$ are (both) ordered pairs. These two ordered pairs are distinct despite despite having identical elements.

In general, an ordered n-tuple is written

$$(x_0, \ldots, x_{n-1})$$

and is the natural extension of Definition 16. (As a matter of convention we call say a: 2-tuple is a 'tuple'; 3-tuple is a 'triple'; 4-tuple is a 'quadruple'; and so on.)

A binary relation, or simply relation, is just a collection (set) of ordered pairs (and in general ordered n-tuples).

Example 17. $R = \{(0,0),(2,5),(7,5)\}$ is a relation.

Notation. We write aRb when $(a,b) \in R$. In Example 17 $0R0$, $2R5$, and $7R5$ (but *not* $5R2$ or $5R7$).

This is an important fact that bears repeating: *any set of ordered pairs is a relation.*

Read $\oplus$ as 'o-plus'.

Definition 17 (Relation)**.** $\oplus$ is an n-ary relation when

$$\forall x;\, x \in \oplus \implies x \text{ is an ordered } n\text{-tuple.}$$

In particular $\oplus$ is a (binary) relation when

$$\forall x;\, x \in \oplus \implies x \text{ is an ordered pair.}$$

Exercise 7. Show $\varnothing$ satisfies the definition of (and is consequently) a relation.

The cartesian product quickly builds sets of ordered pairs and thus relations as well.

Definition 18 (Cartesian product)**.** The Cartesian product of sets A and B is denoted $A \times B$ and given by

$$A \times B = \{(a,b) \,:\, a \in A \wedge b \in B\}\,.$$

Example 18.

$$\{2,3\} \times \{x,y,z\} = \{(2,x),(2,y),(2,z),(3,x),(3,y),(3,z)\}$$

Notation. When $A = B$ in Definition 18 we may write A^2 for $A \times A$.

Relations are utterly pervasive in mathematics. For instance, $<$ and $=$ are relations.

Over the naturals $\mathbb{N}$ the 'less than' symbol is the relation

$$\begin{aligned} `<' &= \{(0,1),(0,2),\dots,(1,2),(1,3),\dots\} \\ &= \{(a,b) \,:\, a<b \wedge a,b\in\mathbb{N}\} \end{aligned}$$

and, 'equals to' (also over $\mathbb{N}$) is

$$\begin{aligned} `=' &= \{(0,0),(1,1),(2,2),(3,3),\dots\} \\ &= \{(a,a) \,:\, a\in\mathbb{N}\}\,. \end{aligned}$$

This viewpoint is somewhat obscured by the implied understanding of $x<y$ as short form for $(x,y)\in `<'$.

Example 19. Taking '$<$' as a relation over $\mathbb{N}$

$$1<2 \iff (1,2)\in `<' \iff \top$$

and

$$3<0 \iff (3,0)\in `<' \iff \bot.$$

In fact *any* binary predicate can be fashioned into a relation using class abstraction:

Proposition 8. Suppose P is a predicate with definition

$$\begin{aligned} P : A\times A &\to \{\top,\bot\} \\ (a,b) &\mapsto P(a,b) \end{aligned}$$

then

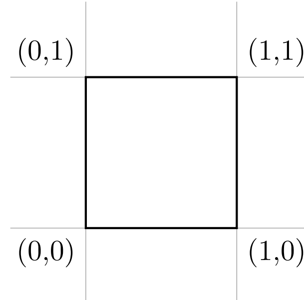
$$\{(a,b) \,:\, P(a,b) \wedge (a,b)\in A\times A\}$$

is a relation.

Proof. Immediate consequence of the definition of Relation. □

To disavow ourselves of the idea that all relations have infinite size, consider the following relation of finite cardinality.

Example 20. Place the unit square in $\mathbb{Z} \times \mathbb{Z}$ as such



and let $\oplus = \{(0,0),(0,1),(1,0),(1,1)\}$. This relation tests if an element of $\mathbb{Z} \times \mathbb{Z}$ is a vertex: $0 \oplus 2 \iff \bot$, $0 \oplus 2 \iff \bot$, $0 \oplus 1 \iff \top$, and so on.

Relations of finite size can be drawn as graphs.

Definition 19 (Directed graph)**.** A directed graph $G = (N, E)$ is a collection of NODES N, and a set of DIRECTED EDGES, $E \subseteq N \times N$.

(Notice the striking similarities between the definition of graph and that of a relation. In particular, both are essentially defined as a set of ordered pairs.)

Example 21. The relation of Example 20 as a directed graph.

This graph has nodes $N = \{0, 1\}$ and, denoting the directed path $a \longrightarrow b$ as (a, b), directed edges $E = \{(0,0),\ (0,1),\ (1,0),\ (1,1)\}$.

Exercise 8. How many distinct directed edges can a graph of n-nodes have?

§Properties of Relations

A relation $\oplus \subseteq A \times A$ can be reflexive, symmetric, antisymmetric, and/or transitive.

Definition 20 (Reflexive)**.** $\oplus$ is REFLEXIVE when

$$a \in A \implies a \oplus a.$$

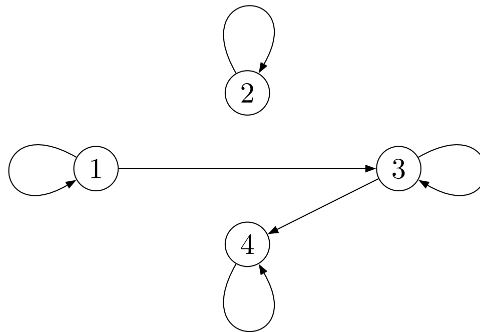
(Each node in the graph loops back to itself.)

Proposition 9. $\oplus$ is reflexive $\iff$ $\{(a, a) : a \in A\} \subseteq \oplus$.

Proof. □

Example 22 (Reflexive)**.** $A = \{1, 2, 3, 4\}$ and

$$\oplus = \{(1, 1), (2, 2), (3, 3), (4, 4)\} \cup \{(1, 3)\} .$$



is reflexive.

Exercise 9. Let $A = \{0, 1, 2, 3, 4\}$ in Example 22. Is $\oplus$ still reflexive?

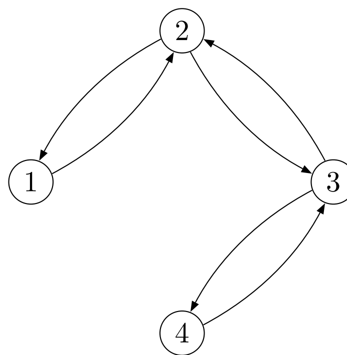
Definition 21 (Symmetric)**.** $\oplus$ is SYMMETRIC when

$$a \oplus b \implies b \oplus a$$

(If $a \longrightarrow b$ then $b \longrightarrow a$.)

Example 23 (Symmetric)**.** $A = \{1, 2, 3, 4\}$ and

$$\oplus = \{(1, 2), (2, 1)\} \cup \{(2, 3), (3, 2)\} \cup \{(3, 4), (4, 3)\} .$$



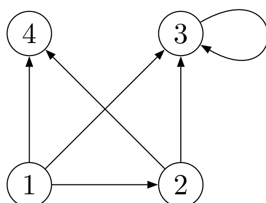
Definition 22 (Antisymmetric)**.** $\oplus$ is ANTISYMMETRIC when

$$a \oplus b \wedge b \oplus a \implies a = b.$$

The 'anti' in antisymmetry describes how these types of relations *prohibit* nontrivial symmetry among its nodes. The 'trivial' symmetries, which are just loops, are okay.

Example 24 (Antisymmetric)**.** $A = \{0, 1, 2, 3, 4\}$ and

$$\oplus = \{(3, 3)\} \cup \{(1, 2), (1, 3), (1, 4), (2, 4), (2, 4)\}.$$



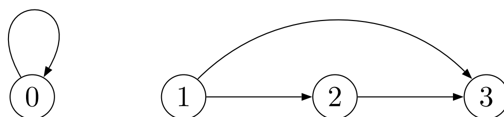
Definition 23 (Transitive)**.** $\oplus$ is TRANSITIVE when

$$a \oplus b \wedge b \oplus c \implies a \oplus c.$$

(If there is an INDIRECT PATH from a to c then there *must* be a DIRECT PATH as well.)

Example 25. $A = \{0, 1, 2, 3\}$ and

$$\oplus = \{(0, 0)\} \cup \{(1, 2), (2, 3)\} \cup \{(1, 3)\}.$$



Exercise 10. Is $\oplus = \{(1, 2), (4, 8), (3, 5)\}$ transitive?

Exercise 11. What is the 'smallest' (by set cardinality) transitive relation?

Composition and Closures

Recall two functions

$$f : A \to B \qquad \text{and} \qquad g : B \to C.$$

can be combined into a new function

$$f \circ g : A \to B \to C$$

via function composition.

This composition is applicable to relations as well.

Definition 24 (Composition of a relation)**.** The COMPOSITION of a relation $R \subseteq A \times B$ with $S \subseteq B \times C$ is denoted $R \circ S$ and defined

$$R \circ S = \{(a, c) \,:\, (a, b) \in R \wedge (b, c) \in S\} \,.$$

Or, to express this with $\oplus = R$ and $\otimes = S$:

$$\oplus \circ \otimes = \{(a, c) \,:\, a \oplus b \wedge b \otimes c\}$$

Example 26. Let

$$R = \{(1, a), (2, a), (2, b)\} \text{ and } S = \{(a, \alpha), (b, \beta), (b, \gamma)\}$$

then

$$R \circ S = \{(1, \alpha), (2, \alpha), (2, \beta), (2, \gamma)\} \,.$$

If R is a relation over $A \times A$ then R can be composed with itself:

$$\begin{aligned} R^0 &= \{(a, a) \,:\, a \in A\} \\ R^1 &= R \\ R^2 &= R \circ R \\ &\vdots \\ R^\ell &= R \circ R^{\ell-1}. \end{aligned}$$

In particular, when $\ell \to \infty$ we define

Definition 25 (Transitive closure)**.** The TRANSITIVE CLOSURE of $R \subseteq A \times A$ is denoted R^+ and given by

$$R^+ = \bigcup_{i=1}^{\infty} R^i.$$

Definition 26 (reflexive transitive closure)**.** The REFLEXIVE TRANSITIVE CLOSURE of $R \subseteq A \times A$ is denoted R^* and given by

$$R^* = R^+ \cup R^0.$$

It is best to demonstrate these with examples, but first notice Definition 25 and Definition 26 imbue R with transitivity and reflexivity. In other words, the transitive closure of R is transitive (even if R was not) and similarly, the transitive reflexive closure of R is reflexive *and* transitive.

> One way of calculating the transitive closure is to draw the relation as a graph and add the *minimum* amount of edges to draw in direct paths when there are indirect paths. Reflexivity is easily obtained by giving every node a closed loop.

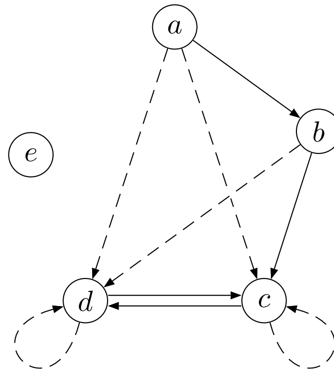
Example 27. Let $A = \{a, b, c, d, e\}$ and

$$R = \{(a, b), (b, c), (c, d), (d, c)\} .$$

$$\begin{aligned} R^1 &= R \\ R^2 &= R^1 \circ R = \{(a, c), (b, d), (c, c), (d, d)\} \\ R^3 &= R^2 \circ R = \{(a, d), (c, d), (d, c), (b, c)\} \\ R^4 &= R^3 \circ R = \{(a, c), (d, c), (c, d), (b, d)\} && \text{(no new pairs)} \end{aligned}$$

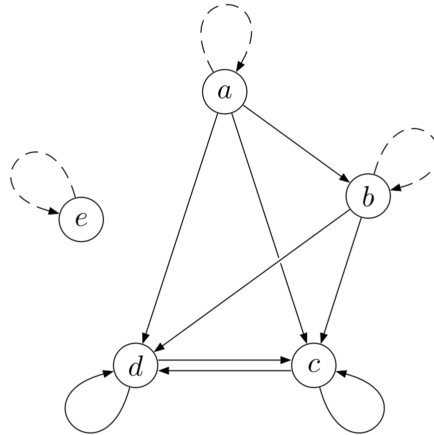
Thus, the transitive closure is

$$R^+ = R \cup \{(a, c), (a, d), (b, d), (c, c), (d, d)\}$$



and the reflexive transitive closure (found by adding all the remaining closed loops to R^+) is

$$R^* = R^+ \cup \{(a, a), (b, b), (e, e)\}.$$

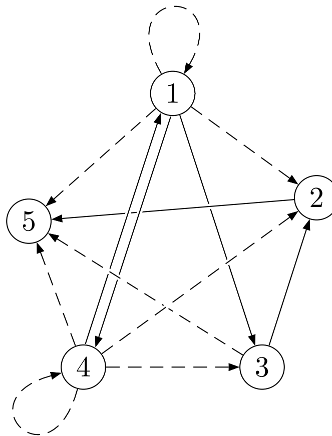


Example 28. Let $A = \{1, 2, 3, 4, 5\}$ and

$$R = \{(1,3), (1,4), (2,5), (3,2), (4,1)\}.$$

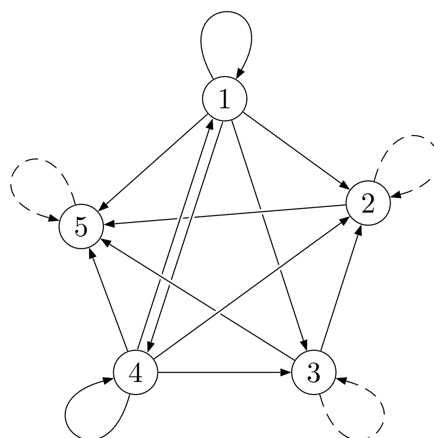
The transitive closure is

$$R^+ = R \cup \{(1,1), (1,2), (1,5), (3,5), (4,2), (4,3), (4,4), (4,5)\}$$



and the reflexive transitive closure is

$$R^* = R^+ \cup \{(2,2), (3,3), (5,5)\}$$



§Equivalence Relations

Recall the relation '$=$' induces.

Exercise 12. '$=$' defines the following relation (over $\mathbb{N}$)

$$`=\text{'} = \{(x, x) : x \in \mathbb{N}\}.$$

Demonstrate why this relation is reflexive, symmetric, and transitive but *not* antisymmetric.

More generally, relations like that of Exercise 12 define a class of relations called 'equivalence relations'.

Definition 27 (Equivalence relation)**.** A binary relation $\simeq$ over A (that is to say, $\simeq \subseteq A \times A$) is an EQUIVALENCE RELATION when $\simeq$ is

1. reflexive,
2. symmetric, and
3. transitive.

Example 29. Recall the C-operator '%' (modulo) that computes remainders. For example,

$$\begin{aligned} 7\%2 &\equiv 1 & &\text{since } 7 = 2 \cdot 3 + \mathbf{1} \\ 12\%3 &\equiv 0 & &\text{since } 9 = 3 \cdot 4 + \mathbf{0} \\ 21\%8 &\equiv 5 & &\text{since } 21 = 8 \cdot 2 + \mathbf{5}. \end{aligned}$$

(If you are having trouble understanding this, imagine you are counting on a clock. 17:00≡5:00pm because $17\%12 \equiv 5$).

Now consider a relation that relates two numbers if they have the same remainder 'modulo' 5.

$$\begin{aligned}
\text{`\%5'} = \{&(0,0),(5,0),(10,0),\ldots,(0,5),(5,5),(10,5),\ldots \\
&(1,1),(6,1),(11,1),\ldots,(1,6),(6,1),(11,1),\ldots \\
&\vdots \\
&(4,4),(9,4),(13,4),\ldots\} \\
= \{&(x,y) : x,y \in \mathbb{N} \wedge x\%5 = y\%5\}
\end{aligned}$$

(We will abandon trying to give quasi-explicit versions of relations as it will become progressively futile.)

Exercise 13. Verify '%5' is an equivalence relation.

An equivalence relation ' $\simeq$ ' over A partitions A into disjoint subsets called 'equivalence classes'.[iii]

Definition 28 (Equivalence Class)**.** The EQUIVALENCE CLASS of $a \in C$ for the equivalence relation $\simeq$ is denoted $[a]_{\simeq}$ and given by

$$[a]_{\simeq} = \{b : b \in C \wedge a \simeq b\}.$$

Example 30. The equivalence classes induced by %5 are the disjoint partitioning of the integers corresponding to the following modular images. (Remember $x\%5 = 0$ is notation for $\exists y \in \mathbb{N}$ such that $(x,y) \in$ '%5'.)

$$\begin{aligned}
[0]_{\%5} &= \{x \in \mathbb{N} : x\%5 = 0\} \\
&= \{0,5,10,15,\ldots\} \\
[1]_{\%5} &= \{x \in \mathbb{N} : x\%5 = 1\} \\
&= \{1,6,11,16,\ldots\} \\
&\vdots \\
[4]_{\%5} &= \{x \in \mathbb{N} : x\%5 = 4\} \\
&= \{4,9,14,19,\ldots\}
\end{aligned}$$

so that

$$\bigsqcup_{i=0}^{4} [i]_{\%5} = [0]_{\%5} \sqcup \cdots \sqcup [4]_{\%5} = \mathbb{N}$$

where $\sqcup$ is used to indicate that the union is over disjoint sets.

[iii] Since $\sim$ is a called a 'sim', some call $\simeq$ "sim-equals".

Proposition 10. Let $\simeq$ be an equivalence relation in $\mathbb{Z}^2$, then $\forall(x, y) \in \mathbb{Z}\times\mathbb{Z}$

$$[x]_\simeq = [y]_\simeq \vee [x]_\simeq \cap [y]_\simeq = \varnothing.$$

That is to say, two equivalence classes can not have nontrivial intersection.

Proof. Suppose $[a]_\simeq \cap [b]_\simeq \neq \varnothing$,

$$
\begin{aligned}
&[a]_\simeq \cap [b]_\simeq \neq \varnothing \\
&\implies \exists x : x \in [a]_\simeq \wedge x \in [b]_\simeq && \text{Defn. of } \cap \\
&\implies a \simeq x \wedge b \simeq x && \text{Defn. of class} \\
&\implies a \simeq b && \text{symmetry and transitivity} \\
&\implies \forall c \in \mathbb{Z};\ a \simeq c \implies b \simeq c && \text{symmetry and transitivity} \\
&\implies \forall x \in [a]_\simeq;\ \big[a \simeq x \wedge b \simeq x\big] \implies x \in [b]_\simeq && \text{Defn. of class} \\
&\implies [a]_\simeq \subseteq [b]_\simeq
\end{aligned}
$$

By similar argument $[b]_\simeq \subseteq [a]_\simeq$ giving $[a]_\simeq = [b]_\simeq$. Thus either $[a]_\simeq = [b]_\simeq$ or $[a]_\simeq \cap [b]_\simeq = \varnothing$. □

§1.3 Alphabets, Words, and Languages

We have already seen how sets extend to define ordered n-tuples. Now we extend sets to our principal object of study: LANGUAGES.

§More Set Constructs

In contrast to the definition of a set, a *multiset* is a set where duplicates *are* counted (ordering is still ignored).

Definition 29 (Multiset)**.** A MULTISET is a set where *duplicate elements are counted.*

Example 31. $A = \{1, 2, 2, 3, 3, 3\}$ and $B = \{2, 1, 3, 3, 3, 2\}$ are multisets satisfying

$$A = B$$

and $|A| = |B| = 6$.

Unfortunately we have no way of distinguishing sets from multisets in writing (both use $\{\}$). We assume that sets omit duplicates and sets with duplicates present are mutisets.

Exercise 14. Is $\{x\} = \{x, x\}$?

Definition 30 (Sequence)**.** A SEQUENCE is an *ordered*-multiset. Sequences are denoted

$$S = (x_0, x_1, \dots, x_{n-1}) \tag{1.1}$$

and have this short form: $S_i = x_i$ (like array indexing). The LENGTH of a sequence is the cardinality of the sequence when viewed as a multiset.

Two sequences, A and B are equal when $\forall i \; A_i = B_i$.

Infinite sequences are given explicitly, recursively, or using a type of class abstraction called the 'closed form'.

> The which Fibonacci sequence models a rabbit population on a secluded island.

Example 32 (Fibonacci sequence)**.** The FIBONACCI SEQUENCE is given/generated by

Explicitly $F = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots)$
(explicit is a misnomer as '...' means the reader is left to determine the pattern.)

Recursively $F_{n-2} = F_n - F_{n-1}$ with $F_0 = 0$, $F_1 = 1$.

Closed Form Let $\varphi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$

$$F_n = \frac{\varphi^n - \psi^n}{\varphi - \psi} = \frac{\varphi^n - \psi^n}{\sqrt{5}}. \tag{1.2}$$

> $\varphi = \frac{1+\sqrt{5}}{2}$ or the GOLDEN RATIO is pervasive in industrial and everyday design. For instance the pantheon and any credit card share the same relative dimensions because of the golden ratio.

§1.4 Alphabets and Languages

Outside of mathematics a LANGUAGE is a collection of WORDS which in turn are sequences of LETTERS taken from some ALPHABET.[iv]

Example 33. The English language has an alphabet consisting of letters 'a' through 'z'

[iv] Languages also have grammars §??.

$$\{\text{a, b, c, } \ldots \text{, z}\},$$

and a language consisting of english words taken from the dictionary.

To make this mathematically precise:

Definition 31 (Alphabet)**.** An ALPHABET is a set of symbols denoted by Σ.

Definition 32 (Word)**.** A WORD *over an alphabet* Σ is a finite sequence of letters taken from Σ.

Equivalently a word is any element of

$$\Sigma^N = \underbrace{\Sigma \times \cdots \times \Sigma}_{N-\text{times}} \tag{1.3}$$

for $N < \infty$ (that is, for N arbitrarily large but finite).

Example 34. For $\Sigma = \{\text{a}, \ldots, \text{z}\}$

$$(\text{e, x, a, m, p, l, e})$$

is a word.

Notation. When a sequence is a word it is understood that

$$\text{example}$$

is written in place of the more cumbersome

$$(\text{e, x, a, m, p, l, e}).$$

Furthermore, we write unknown words, like x, as

$$x = x_0 x_1 \cdots x_n.$$

Definition 33 (Language)**.** A LANGUAGE is a set of words (languages *can* be, and usually are, infinite).

Example 35. The collection of words

$$\{\text{They, came, from, behind}\}$$

is a language (and a SUBLANGUAGE of the full english language).

Exercise 15. (a, 1) and a1 are two ways to write a word of

$$\{a, b\} \times \{1, 2, 3, 4\}$$

write the remaining seven words (in both ways).

Definition 34 (Universal language)**.** The UNIVERSAL LANGUAGE, denoted Σ^*, is the set of all words constructible from an alphabet Σ.

$$\Sigma^* = \Sigma \times \Sigma \times \Sigma \times \cdots = \Sigma^\infty. \tag{1.4}$$

To build infinite languages of more precise form we must first define the following operations on words and letters.

§Operations on words

Definition 35 (word length)**.** The LENGTH of a word x is denoted

$$|x|$$

and is the length of the sequence x is associated to. In other words, it is the numbers of letters that make up the word.

Example 36. Over $\Sigma = \{a, \ldots, z\}$

$$|\text{woot}| = |(\text{w, o, o, t})| = 4.$$

However, over $\Sigma = \{a, \ldots, z\} \cup \{oo\}$

$$|\text{w oo t}| = |(\text{w, oo, t})| = 3.$$

Definition 36 (σ-length)**.** The σ-LENGTH of a word x is denoted

$$|x|_\sigma$$

and is equal to the number of occurrences of the letter σ in x.

Example 37. Over $\Sigma = \{a, \ldots, z\}$

$$|\text{korra}|_\text{r} = 2 \qquad |\text{korra}|_\text{v} = 0.$$

Definition 37 (Empty word)**.** The EMPTY WORD ε is the unique word satisfying

$$|\varepsilon| = 0. \tag{1.5}$$

It corresponds to the empty sequence ().

Definition 38 (Concatenation)**.** The CONCATENATION of two words

$$x = x_0 x_1 \cdots x_n$$
$$y = y_0 y_1 \cdots y_n$$

is written xy and given by

$$xy = x_0 x_1 \cdots x_n y_0 y_1 \cdots y_n.$$

Example 38. Let $w_1 = \text{fancy}$ and $w_2 = \text{pants}$,

$$w_1 w_2 = \text{fancypants}.$$

Proposition 11. The empty word satisfies

1. $x\varepsilon = \varepsilon x = x$, and
2. $\varepsilon\varepsilon = \varepsilon$.

Proof. □

The POWER OF A WORD, written x^n for a word x is the word obtained by concatenating x with itself n-times:

$$x^n = \underbrace{x\,x\,\cdots\,x}_{n-\text{times}}.$$

Consider the 'normal' power operation:

$$2^n = 2 \cdot 2 \cdot 2 \cdots 2.$$

The power of a word is exactly similar to this if we swap multiplication for concatenation.

More concisely, we express the power of a word as follows.

Definition 39 (Power of a word)**.**

$$x^0 = \varepsilon$$
$$x^i = xx^{i-1} \text{ for } i \geq 1$$

Example 39. Let $b =$ badger, $m =$ mushroom, and $s =$ snake then

$$b^3m^2 = \text{badgerbadgerbadgermushroommushroom}$$

and

$$s^0 = \varepsilon.$$

Exercise 16. Take b, m and s as in Example 39. What is $b(bm)^2$? Namely, is $b(bm)^2 = b^3m^2$?

Definition 40 (Prefix)**.** x is a PREFIX of y when

$$\exists z : xz = y.$$

Example 40. Let $t =$ TARDIS[v], then

$$\varepsilon,\ \text{T},\ \text{TA},\ \text{TAR},\ \text{TARD},\ \text{and}\ ,\ \text{TARDI},\ \text{TARDIS}$$

are the *prefixes* of t.

Proposition 12. For x a word

1. $\forall x$, ε is a prefix of x, and
2. $\forall x$, x is a prefix of x.

Proof. □

Definition 41 (Suffix)**.** x is a SUFFIX of y when

$$\exists z : zx = y.$$

Example 41. Let $t =$ TARDIS, then

$$\varepsilon,\ \text{S},\ \text{IS},\ \text{DIS},\ \text{RDIS},\ \text{ARDIS},\ \text{and}\ \ \text{TARDIS}$$

are the *suffixes* of t.

Definition 42 (subword)**.** Let $x \trianglelefteq y$ denote 'x is a SUBWORD of y', then

$$x \trianglelefteq y \overset{\text{def.}}{\iff} \exists w, z : wxz = y.$$

Example 42. The subwords of bite are

$$\varepsilon,\ \text{b},\ \text{bi},\ \text{bit},\ \text{bite},\ \text{i},\ \text{it},\ \text{ite},\ \text{t},\ \text{te},\ \text{e}$$

[v] Time and Relative Dimension in Space.

Exercise 17. What are the subwords of 0000?

Exercise 18. Is every suffix a subword? Is every subword a suffix?

Exercise 19. What is the maximum number of subwords a word of n letters can have?

Exercise 20. Suppose we are given a word x with $|x| = n$. How many subwords of x would we need to reconstruct x? Note: we do not get to pick the subwords.

Definition 43 (Proper prefix, suffix, subword)**.** x is a PROPER prefix, proper suffix, or proper subword of y when $x \neq \varepsilon$ and $x \neq y$.

Definition 44 (Reversal)**.** The REVERSAL of the word x, denoted x^R, is the word x written in reverse and has recursive defintion

1. $x = \varepsilon \implies x^R = x$
2. $x = ay \implies x^R = y^R a$

for $a \in \Sigma$ and y a word.

Example 43. The reversal of $w = badong$ is $w^R = gnodab$.[vi]

Definition 45 (Palindrome)**.** A word x is a PALINDROME when

$$x = x^R.$$

That is, when the word is the same written forwards and backwards.

Example 44. 'racecar' is a palindrome as well as

"able was I ere I saw elba"

which is a famous english palindrome, attributed to Napoleon Bonaparte after he was exiled to Elba, a Mediterranean island in Tuscany.

§Operations on Languages

Here we extend the operations on words to the languages they inhabit. Let us first give an alternate—recursive—definition of the universal language (Definition 34).

[vi] "Killing is wrong. And bad. There should be a new, stronger word for killing like badwrong or badong. Yes, killing is badong. From this moment I will stand for the opposite of killing: gnodab."
Excerpt from the movie "Kung Pow!".

Definition 46 (Universal language (recursive))**.** Given Σ (some alphabet), Σ^* has recursive definition

1. $\varepsilon \in \Sigma^*$, and
2. $x \in \Sigma^* \implies \forall a \in \Sigma,\ ax \in \Sigma^*$

Exercise 21. Show Definition 46 and Definition 34 are equivalent.

Recall a language $\mathbb{L}$ over an alphabet Σ is merely a subset of Σ^*, i.e. $\mathbb{L} \subseteq \Sigma^*$. Consequently over any Σ,

1. $\varnothing$ is a language, and
2. $\{\varepsilon\}$ is a language.

Exercise 22. What is $\varnothing^*$?

Definition 47 (Concatenation of languages)**.** The CONCATENATION of two languages $\mathbb{L}_1$ and $\mathbb{L}_2$ over Σ is denoted $\mathbb{L}_1\mathbb{L}_2$ and given by

$$\mathbb{L}_1\mathbb{L}_2 = \{uv \,:\, u \in \mathbb{L}_1 \wedge v \in \mathbb{L}_2\}\,.$$

We may view $\mathbb{L}_1\mathbb{L}_2$ as the collection of all possible concatenations of words from $\mathbb{L}_1$ with words from $\mathbb{L}_2$.

Exercise 23. The concatenation of languages is *not* symmetric (more correctly: not COMMUTATIVE). Namely, $\mathbb{L}_1\mathbb{L}_2$ need not equal $\mathbb{L}_2\mathbb{L}_1$. Under what conditions *would* the concatenation operation be commutative?

Example 45. Let $\mathbb{L}_1 = \{\text{super},\ \text{ad}\}$ and $\mathbb{L}_2 = \{\text{man},\ \text{market},\ \text{visor}\}$.

$$\begin{aligned}\mathbb{L}_1\mathbb{L}_2 = \{&\text{superman},\ \text{supermarket},\ \text{supervisor},\\ &\text{adman},\ \text{admarket},\ \text{advisor}\}\end{aligned}$$

Example 46.

$$\{a, ab\}\{bc, c\} = \{abc, ac, abbc\}\,.$$

Note: *abc* was generated *twice*!

Exercise 24. Suppose $|\mathbb{L}_1| = n$ and $|\mathbb{L}_2| = m$. What is the maximum (easy) and minimum (hard) cardinality of $\mathbb{L}_1\mathbb{L}_2$?

To apply the concatenation of languages multiple times we use language power.

Definition 48 (Powers of Languages)**.** For $\mathbb{L}$ a language

$$\begin{aligned}\mathbb{L}^0 &= \{\varepsilon\} \\ \mathbb{L}^{n+1} &= \mathbb{L}^n L, \text{ for } n \geq 1.\end{aligned}$$

Exercise 25. What is $\{\varepsilon\}^0$? What is $\{\varnothing\}^0$?

Exercise 26. It is *not* necessarily the case that $\mathbb{L}^i \subseteq \mathbb{L}^{i+1}$. What is the condition on $\mathbb{L}$ for $\mathbb{L}^i \subseteq \mathbb{L}^{i+1}$?

Exercise 27. Suppose $|L| = n$, what is $\left|\mathbb{L}^j\right|$? Prove your answer using induction (see §**??**).

Notation. For $\mathbb{L}$ a language

$$\mathbb{L}^+ = \bigcup_{i=1}^{\infty} \mathbb{L}^i \qquad \mathbb{L}^* = \bigcup_{i=0}^{\infty} \mathbb{L}^i = \{\varepsilon\} \cup \mathbb{L}^+.$$

Note $\varepsilon \in \mathbb{L}^+ \iff \varepsilon \in \mathbb{L}$.

Proposition 13. For a language $\mathbb{L}$

$$\begin{aligned}\mathbb{L}^+ &= \{w \in \Sigma^* : w = w_1 w_2 \cdots w_n \wedge w_i \in \mathbb{L} \wedge n \geq 1\}, \\ \mathbb{L}^* &= \{w \in \Sigma^* : w = w_1 w_2 \cdots w_n \wedge w_i \in \mathbb{L} \wedge n \geq 0\}.\end{aligned}$$

Proof. Exercise. □

Example 47. Suppose $\mathbb{L} = \{ab, b\}$.

$$\begin{aligned}\mathbb{L}^0 &= \{\varepsilon\} \\ \mathbb{L}^1 &= L = \{ab, b\} \\ \mathbb{L}^2 &= \{abab, abb, bab, bb\}\end{aligned}$$

and

$$\begin{aligned}\mathbb{L}^+ &= \mathbb{L}^1 \cup \mathbb{L}^2 \cup \cdots = \{ab, b\} \cup \{abab, abb, bab, bb\} \\ \mathbb{L}^* &= \mathbb{L}^+ \cup \{\varepsilon\}.\end{aligned}$$

Definition 49 (reversal of languages)**.** The reversal of a language $\mathbb{L}$, denoted $\mathbb{L}^R$, is simply the language consisting of word reversals of $\mathbb{L}$.

$$\mathbb{L}^R = \left\{x^R : x \in L\right\}.$$

Exercise 28. What is the condition on $\mathbb{L}$ for $\mathbb{L}^R = \mathbb{L}$?

Proposition 14 (properties of reversals)**.** For languages A and B

1. $(A \cup B)^R = A^R \cup B^R$,
2. $(A \cap B)^R = A^R \cap B^R$,
3. $(AB)^R = B^R A^R$,
4. $(A^+)^R = (A^R)^+$,
5. $(A^*)^R = (A^R)^*$.

Proof. Exercise. □

An interesting concept is that of the COMPLEMENT of a language, that is, all words a language *does not* include.

Definition 50 (language compliment)**.** Given a language $\mathbb{L} \subseteq \Sigma^*$, the COMPLIMENT of $\mathbb{L}$ denoted $\overline{\mathbb{L}}$ is

$$\overline{\mathbb{L}} = \Sigma^* \setminus \mathbb{L}.$$

Complementation is tied to those words originally available; that is, the complement of identical languages over different alphabets will be different. Moreover, languages with explicit definition do not (necessarily) admit complements where explicit definition is possible.

Example 48. Let $\mathbb{L} = \{a^n b^n : n \geq 0\}$, namely, the collection of words which are a sequence of n many a's followed by n many b's. Over $\Sigma = \{a, b\}$

$$\overline{\mathbb{L}} = \left\{a^i b^j \ : \ i \neq j \ \wedge \ i, j \in \mathbb{N}\right\}$$

whereas over $\Sigma = \{a, b, c\}$

$$\overline{\mathbb{L}} = \{a^i b^j c^k \ : \ i \neq j \ \wedge \ i, j, k \in \mathbb{N}\}.$$

Example 49. Let $\Sigma = \{a\} \implies \Sigma^* = \left\{a^i \ : \ i \in \mathbb{N}\right\}$ and $\mathbb{L} = \left\{a^{2i} \ : \ i \in \mathbb{N}\right\}$, then

$$\overline{\mathbb{L}} = \left\{a^{2i+1} \ : \ i \in \mathbb{N}\right\}.$$

Exercise 29. What is $\overline{\Sigma^*}$ and $\overline{\Sigma^+}$?

Summary of Language Operations

Properties of language concatenation. For languages $\mathbb{L}_1, \mathbb{L}_2$ and $\mathbb{L}_3$,

Associativity $\mathbb{L}_1(\mathbb{L}_2\mathbb{L}_3) = (\mathbb{L}_1\mathbb{L}_2)\mathbb{L}_3$

Identity $\mathbb{L}\{\varepsilon\} = \{\varepsilon\}\mathbb{L} = \mathbb{L}$

Zero $\mathbb{L}\varnothing = \varnothing\mathbb{L} = \varnothing$

Distributivity (of concatenation) $\mathbb{L}_1(\mathbb{L}_2 \cup \mathbb{L}_3) = \mathbb{L}_1\mathbb{L}_2 \cup \mathbb{L}_1\mathbb{L}_3$

Distributivity (of $\cup$) $(\mathbb{L}_1 \cup \mathbb{L}_2)\mathbb{L}_3 = \mathbb{L}_1\mathbb{L}_3 \cup \mathbb{L}_2\mathbb{L}_3$

Note: distributivity does *not* hold with respect to $\cap$.

Properties of plus and kline.

Proposition 15. For $\mathbb{L}$ a language

1. $\mathbb{L}^* = \mathbb{L}^+ \cup \{\varepsilon\}$,
2. $\mathbb{L}^+ = \mathbb{L}\mathbb{L}^*$,
3. $(\mathbb{L}^+)^+ = \mathbb{L}^+$,
4. $(\mathbb{L}^*)^* = \mathbb{L}^*$,
5. $\varnothing^+ = \varnothing$,
6. $\varnothing^* = \{\varepsilon\}$,
7. $\varnothing^0 = \{\varepsilon\}$,
8. $\{\varepsilon\}^+ = \{\varepsilon\}$,
9. $\{\varepsilon\}^* = \{\varepsilon\}$,
10. $\mathbb{L}^+\mathbb{L} = \mathbb{L}\mathbb{L}^+$,
11. $\mathbb{L}^*\mathbb{L} = \mathbb{L}\mathbb{L}^*$.

Proof. Exercise. □

§1.5 Proof Methods

Much to the dismay of students, there is no 'recipe' for doing proofs (or solving problems in general). However, there are some established strategies and tools that often do the trick. We review them now.

§Direct Proof

Let us substitute a formal description *of* a direct proof with a story *about* a direct proof.

The misbehaving (and still unrealized genius of number theory) Gauß (1777-1855), was exiled to the corner of his classroom and told not to return until he had calculated the sum of the first ten thousand numbers. (Suffice it to say the teacher was fed up.)

Sadly, and to the astonishment of the teacher, Gauß returned with the correct answer in a matter of minutes—the pupil had deduced what the teacher could not:

$$\begin{aligned}
&1 + 2 + 3 + \cdots + 5\,000 + 5\,001 + \cdots + 9\,998 + 9\,999 + 10\,000 \\
&= (1 + 10\,000) + (2 + 9\,999) + (3 + 9\,998) + \cdots + (5\,000 + 5\,001) \\
&= (10\,001) + (10\,001) + (10\,001) + \cdots + (10\,001)
\end{aligned}$$

Figure 1.3: Gauß as pictured on the German 10-Deutsche Mark banknote (1993; discontinued). The Normal, or 'Gaussian' distribution is depicted to his left.

$$= \left(\frac{10\,000}{2}\right) \cdot (10\,000 + 1)$$

An application—perhaps even discovery—of the general form of this equation given in Theorem 16.

§The Principle of Mathematical Induction

The PRINCIPLE OF MATHEMATICAL INDUCTION (or 'PMI' or just 'induction'), states simply that a proposition $P : m \to \{\top, \bot\}$ is true for all $m \in \mathbb{N}$ if

1. $P(m) \implies P(m+1)$ for any $m \in \mathbb{N}$; and
2. $P(0)$.

(It is implicit that $P(0)$ means $P(0) \equiv \top$.)

To elaborate, the first point is a *weakening* of the intended conclusion and is equivalent to validating the following chain of implications:

$$P(0) \implies P(1) \implies \cdots \implies P(n) \implies P(n+1) \implies \cdots$$

Provided that $P(0) \equiv \top$ (the second point) *each* proposition in the chain is shown true and thus it is proved that $\forall n P(n)$.

Only $\phi \equiv \top$ satisfies $(\top \implies \phi) \equiv \top$.

Remark 1. *How* one proves $\forall n;\, P(n) \implies P(n+1)$ is often the source of confusion. The direct way of demonstrating $\phi \implies \psi$ is to assume ϕ and show ψ is a consequence. The case of $\phi \equiv \bot$ is not ignored, but is considered too trivial to write as $\bot \implies \phi \equiv \top$ (such statements are called 'vacuous').

As is the case, induction proofs contain the bizarre statement 'assume for any $n \in \mathbb{N}$ that $P(n) \equiv \top$' which is seemingly what requires proof. However, there is gulf of difference between

$$(\forall n P(n)) \implies P(n+1)$$

and

$$\forall n \left(P(n) \implies P(n+1)\right). \tag{1.6}$$

We are *not* assuming $\forall n P(n)$, rather we are assuming, for any n, the antecedent of (1.6).

In the same spirit, assuming 'there is' some n for which $P(n) \equiv \top$ is also wrong. Demonstrating $\exists n;\, P(n) \implies P(n+1)$ is insufficient.

Formally, the PMI is given this way.

Theorem 4 (The Principle of Mathematical Induction)**.** For *every* predicate $P : \mathbb{N} \to \{\top, \bot\}$

$$\{P(0) \wedge \forall m \in \mathbb{N}\, [P(m) \implies P(m+1)]\} \implies \{\forall m \in \mathbb{N}\, [P(m)]\}$$

(we use superfluous bracketing to express something more meaningful).

To apply this theorem, let us prove 'Gauss' formula' using induction rather than deduction.

Proposition 16. For any $n \in \mathbb{N}$

$$0 + 1 + 2 + \cdots n = \sum_{i=0}^{n} i = \frac{n \cdot (n+1)}{2}.$$

Proof. Proceeding with induction it is clear that $n = 0$ is satisfied as

$$\left[\sum_{i=0}^{n} i\right]_{n=0} = \sum_{i=0}^{0} i = 0 \tag{1.7}$$

and

$$\left[\frac{n\cdot(n+1)}{2}\right]_{n=0} = \frac{0\cdot(0+1)}{2} = 0.$$

(the 'Base case').

Assume for arbitrary $m \in \mathbb{N}$ the validity of

$$0+1+2\cdots+m = \sum_{i=0}^{m} i = \frac{m\cdot(m+1)}{2}$$

(the 'Induction hypothesis').

Using this deduce

$$\begin{aligned}
0+1+2\cdots+m+(m+1) &\overset{\text{IH}}{=} \frac{m\cdot(m+1)}{2}+(m+1) \\
&\quad \text{(by Induction Hypothesis)} \\
&= \frac{m\cdot(m+1)+2\,(m+1)}{2} \\
&= \frac{m^2+3m+2}{2} \\
&= \frac{(m+1)(m+2)}{2}
\end{aligned}$$

which shows (1.7) holds for $n = m+1$ provided it holds for $n = m$.

By the PMI (1.7) is valid $\forall n \in \mathbb{N}$. □

§Contradiction

Contradiction is a proof technique where, in order to show some predicate P true, we assume $\neg P$ and deduce $\bot$. (That is, we show invalid P has an absurd consequence).

Logically, proof by contradiction can be expressed as

Theorem 5 (Proof by Contradiction)**.** For any predicate P

$$(\neg P \implies \bot) \implies P.$$

We give two standard examples (in ascending difficulty) for which contradiction is applicable.

Example 50. $\sqrt{2}$ can not be expressed as a fraction (i.e. $\sqrt{2}$ is an irrational number)

(Note: Read $a \mid b$ as 'a divides b' which means $\exists c : ac = b$. E.g. $2 \mid 6$.)

Proof. Towards a contradiction (TAC for short), suppose $\sqrt{2}$ *can* be expressed as

$$\sqrt{2} = \frac{a}{b} \tag{1.8}$$

with $a, b \in \mathbb{N}$. Assume further that $\frac{a}{b}$ is a *reduced fraction* so that $\gcd(a, b) = 1$.

Squaring (1.8) yields

$$2 = \frac{a^2}{b^2} \implies 2b^2 = a^2.$$

Trivially $2 \mid 2b^2$ and so $2 \mid a^2$.

But 2 cannot be decomposed (it is prime) so it must be that $2 \mid a$ and thus $4 \mid a^2$. Similarly (applying this argument in the reverse direction) $4 \mid 2b^2 \implies 2 \mid b^2$ and thus $2 \mid b$.

However, if *both* a and b are divisible by 2 it must be the case that $\frac{a}{b}$ is *not* reduced, i.e. $\gcd(a, b) = 2 \neq 1$. ↯[vii] □

Example 51. There are infinitely many prime numbers.

Proof. TAC suppose there are *finitely* many prime numbers $\mathbb{P} = \{p_0, p_1, p_2, \ldots, p_\ell\}$ and consider $n \in \mathbb{N}$ given by

$$n = p_0 \cdot p_1 \cdot p_2 \cdots p_\ell + 1.$$

As every integer has a prime divisor (this is the fundamental theorem of algebra) there must be some $p_i \in \mathbb{P} : p_i \mid n$. Clearly $p_i \mid p_0 \cdot p_1 \cdots p_i \cdots p_\ell$ so it follows

$$p_i \mid (n - p_0 \cdot p_1 \cdots p_n)$$

(It is readily shown that $a \mid b \wedge a \mid c \implies a \mid (b - c)$.)

Consequently, $p_i \mid 1 \implies p_i = 1$ (by definition 1 is *not* a prime) and thus $p_i \notin \mathbb{P}$. ↯ □

§The Pigeonhole Principle

The PIGEONHOLE PRINCIPLE is the mathematical formalization of the statement:

[vii] There are many symbols for contradiction, among them: $\Rightarrow\Leftarrow$, ↯, $\nleftrightarrow$, and ※. Feel free to use whichever one you like!

> If you put $n+1$ pigeons into n holes then there is (at least) one hole with two pigeons.[viii]

This result may seem fairly obvious but there are some interesting *computer science* applications none-the-less. For instance, the PHP is used to show

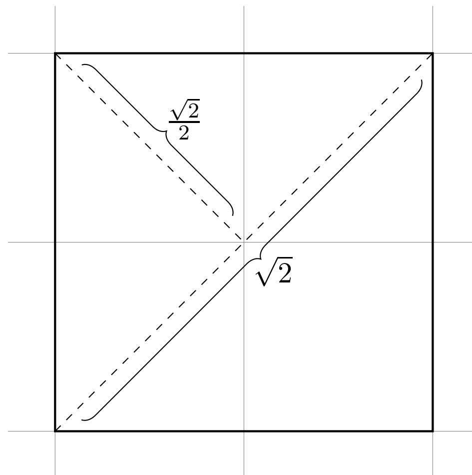
1. collisions are inevitable in a hash tables no matter how sophisticated the hash function is, and
2. lossless compression algorithms will *always* make some images *larger*.

Recall the Euclidean distance between two points in the plane:

$$\left|\overline{(x_1, y_1)(x_2, y_2)}\right| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Proposition 17. If 5 points are drawn within the interior (i.e. *not* on the edge) of unit square then there are two points that have Euclidean distance $< \frac{1}{\sqrt{2}}$.

Proof. The diagonal of a unit square has length $\sqrt{2}$ (Pythagoras' theorem) and a subsquare $\frac{1}{4}$ the area has diagonal length $\frac{\sqrt{2}}{2}$,



Therefore, any points within the same subsquare can be at *most* $\frac{\sqrt{2}}{2} = \frac{1}{\sqrt{2}}$ units apart.

By PHP, if five points are drawn within the interior of a square, then two points must be in the same subsquare (there are only four such subsquares). Thus there are two points with Euclidean distance less than $\frac{1}{\sqrt{2}}$. □

[viii] The motivation for placing pigeons into holes eludes the Author (who is also bothered by the notion of cramming two pigeons into a hole meant for one).

§1.6 End of Chapter Exercises

Exercise 1. Prove that the power set operation is *monotonic*, that is,

$$A \subseteq B \implies \mathscr{P}(A) \subset \mathscr{P}(B). \tag{1.9}$$

Exercise 2. What is $\mathscr{P}(\mathscr{P}(\varnothing))$?

Exercise 3. Suppose $\neg(A \subseteq B)$ and $\neg(B \subseteq A)$. What conclusions, if any, can be made about $A \cap B$?

Exercise 4. Prove the following are equivalent for sets X and Y.

1. $X \subset Y$,
2. $X \cup Y = Y$, and
3. $X \cap Y = X$.

(Hint: you only need to show $1 \implies 2 \implies 3 \implies 1$ to establish equivalence—in fact *any* permutation of 1,2,3 will do.)

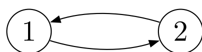
Exercise 5. Is there a relation satisfying *all* properties of Definition **??**? That is, is there a reflexive, symmetric, antisymmetric, and transitive relation? (If you decide to provide an example, ensure you establish it is indeed a relation.)

Exercise 6. Draw a graph with at least 3 nodes that is reflexive, symmetric, and transitive (i.e. all but antisymmetry).

Exercise 7. For each property below, draw a graph which does *not* have that property.

1. reflexive,
2. transitive,
3. symmetric.

Exercise 8. In the definition of antisymmetry loops like



were disallowed. In the same spirit, what should be disallowed for graphs that are antisymmetric *and* transitive?

Exercise 9. For $a, b, c \in \mathbb{N}$ show

$$a \,\Big|\, b \wedge a \,\Big|\, c \implies a \,\Big|\, (b - c)\,.$$

Exercise 10. Prove the concatenation operations is *associative*, namely, for words x, y, and z:

$$(xy)z = x(yz).$$

Exercise 11. Prove plus and kline are monotonic. That is

$$A \subseteq B \implies A^+ \subseteq B^+$$

and

$$A \subseteq B \implies A^* \subseteq B^*.$$

§1.7 Exercise Solutions

Exercise 1. $\{2,2,2,3,3,4\} = \{2,3,4\}$ is a set with cardinality 3.

Exercise 2. Placeholder.

Exercise 3. $E = \{2^n : n \in \mathbb{N}\}$.

Exercise 4. Placeholder.

Exercise 5. Placeholder.

Exercise 6. Placeholder.

Exercise 7. Placeholder.

Exercise 8. Placeholder.

Exercise 9. As $(0,0) \notin \oplus$ and $0 \in A$, $\oplus$ can not be reflexive.

Exercise 10. Yes. It never *fails* to be transitive.

Exercise 11. $\varnothing$ is the smallest transitive relation (it has no members and thus cannot fail to be transitive).

Exercise 12. Placeholder.

Exercise 13. Placeholder.

Exercise 14. For our purposes $\{x\} \neq \{x, x\}$ because the later set is assumed to be a multiset.

Exercise 15. (a,2), (a,3), (a,4), a2, a3, a4, (b,2), (b,3), (b,4), b2, b3, b4.

Exercise 17. The subwords are 0, 00, 000, 0000.

Exercise 18. Yes every suffix is a subword. However, subwords are not necessarily suffixes.

Exercise 19. Placeholder.

Exercise 21. Placeholder.

Exercise 22. $\varnothing^* = \{\varepsilon\}$.

Exercise 23. Placeholder.

Exercise 24. Placeholder.

Exercise 25. $\{\varepsilon\}^0 = \varepsilon$ and $\{\varnothing\}^0 = \varepsilon$.

Exercise 26. Placeholder.

Exercise 28. $w \in \mathbb{L} \implies w$ is a palindrome.

Exercise 29. $\overline{\Sigma^*} = \varnothing$ and $\overline{\Sigma^+} = \{\varepsilon\}$?

Chapter 2

Finite Automata

"Destiny has cheated me
By forcing me to decide upon
The Woman that I idolize
Or the hands of an Automaton."

– Philip J. Fry, *Futurama*

Deterministic finite state machines (DFSM), nondeterministic finite state machines (NDFSM), transformation from NDFSM to DFSM, properties of finite automata, and pumping lemma.

§2.1 Finite State Machines

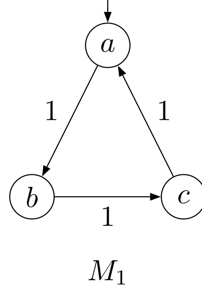
Automata is a fancy word for a machine. From here we introduce a simple machine and add mechanisms to it in order to make the machine more robust. Despite these humble beginnings our simple automatas will ultimately extend to encode and classify the space of all computable (and uncomputable!) problems.

Our first machine, a state machine, is merely a more robust directed graph. The extra sophistication is afforded by the addition of transition rules or, more simply, a labelling of previously 'naked' directed edges. These transition rules dictate how we can move amongst the states.

Definition 51 (State machine)**.** A state machine is a graph with labelled directed edges. We call the nodes of a state machine states and the directed edges transitions.

State machines are expressed via state diagrams (a literal drawing of the states and transitions as a graph).

Example 52. A state diagram, M_1, with three states and three transitions.



M_1

Notation (start state)**.** In Example 52—and in general—the arrow on →(a) is used to designate the STARTING STATE.

To move from state to state (to state), we write

$$a1 \vdash b$$

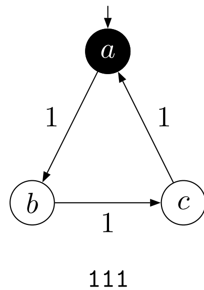
to convey: a goes to b on 1.

> For now ⊢ is used as notation but we will soon be properly define it as a relation.

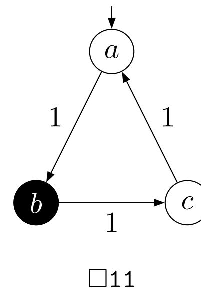
More generally we write

$$a111 \vdash b11 \vdash c1 \vdash a$$

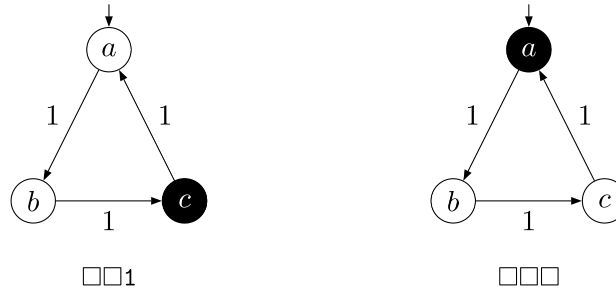
to denote the *sequence* of moves illustrated below. (Movement throughout the machine is communicated using colour: A black state is currently occupied whereas a white one is not.)



`111`



`□11`



The input word 111 depletes (left to right) as we move through the graph. This is illustrated above using □ to denote processed letters.

When the number of states is *finite* the machine is a FINITE STATE MACHINE. Moreover, when the transitions are defined in a particular way, the machine is called a DETERMINISTIC FINITE STATE MACHINE.

> A FSM is a special case of the more general STATE TRANSITION SYSTEM which *is* allowed to have an infinite number of states.

As Determinism is best explained by Nondeterminism, let us take this term for granted until §**??**.

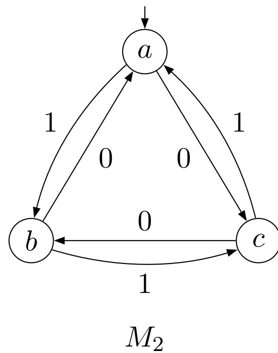
Definition 52 (FSM)**.** A DETERMINISTIC FINITE STATE MACHINE or simply FINITE STATE MACHINE is a quintuple $(Q, \Sigma, s_0, \delta, F)$ where

Q	finite, non-empty, set of states,
Σ	input alphabet,
$s_0 \in Q$	initial state,
$\delta : Q \times \Sigma \to Q$	state transition function, and
$F \subseteq Q$	*set* of final states.

Notation. Let FSM be the set of all deterministic finite state machines,

$$\text{FSM} = \{(Q,\ \Sigma,\ s_0,\ \delta,\ F)\ :\ Q, \Sigma, s_0, \delta, F \text{ are as given in Defn. 61}\}$$

Example 53. Let us extend M_1 to include transitions in the other direction:



M_2

Following Definition 61, $M_2 = (Q, \Sigma, s_0, \delta, F)$ with

$$Q = \{a, b, c\},$$
$$\Sigma = \{0, 1\},$$
$$s_0 = a, \text{ and}$$

$F = \varnothing$ (because we do not yet know how to draw them).

The transition function δ can be given in three ways:

1. state diagram (e.g. M_2),
2. transition table

δ	0	1
a	c	b
b	a	c
c	b	a

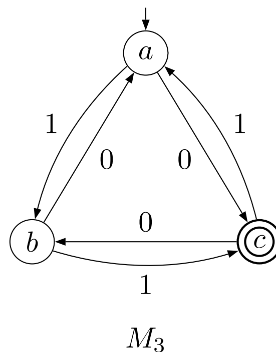
, or

3. explicitly

$$\delta(a, 0) = c \qquad \delta(b, 0) = a \qquad \delta(c, 0) = b$$
$$\delta(a, 1) = b \qquad \delta(b, 1) = c \qquad \delta(c, 1) = a$$

The first, a diagram, is typically used when M is known outright. The remaining two are usually delegated to proofs (although even then a picture may be more appropriate)

Example 54. M_3 is M_2 with final states $\{c\}$.

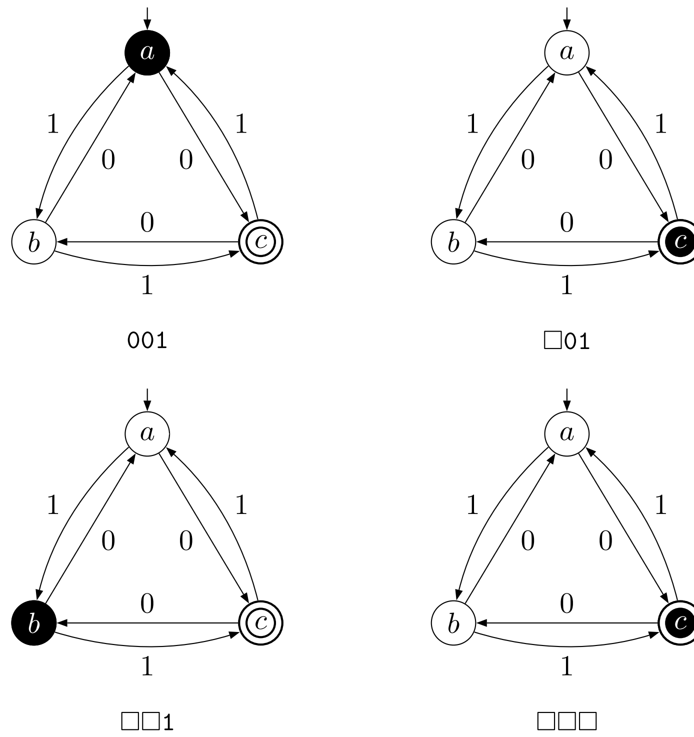


M_3

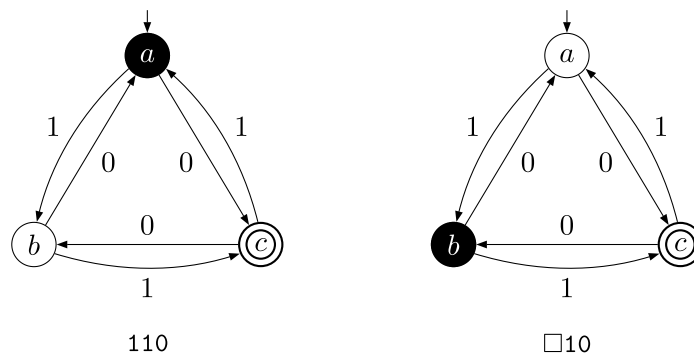
Notation (final state)**.** Concentric circles ◎ are used to designate FINAL STATES.

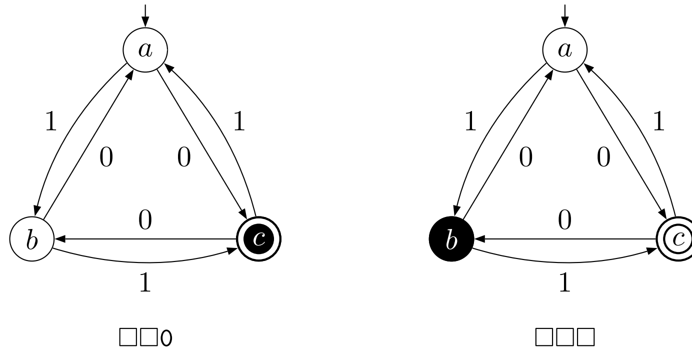
Let us (informally) call input terminating at a final state ACCEPTED. For instance,

Example 55. 001 is accepted by M_3



Example 56. 110 is *not* accepted by M_3

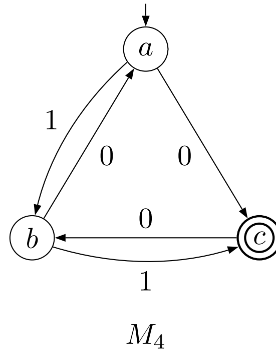




§Complete Machines

So far, we have only dealt with COMPLETE machines. That is, we have yet to encounter a state with some undefined (and required) transition. Put differently: moves were defined at every state, for every possible input.

Example 57. Let M_4 be given by



M_4

and consider (final) state c. This state has no outgoing transitions[i] for 1, or equivalently, $\neg\exists q \in Q \,:\, c1 \vdash q$.

FSMs (like in Example 57) with undefined moves are called INCOMPLETE. More precisely,

Definition 53 (Total function)**.** A function, $\delta : A \to B$, is a TOTAL function when

$$\forall a \in A;\ \exists b \in B \,:\, \delta(a) = b.$$

(Output is defined for every possible input.)

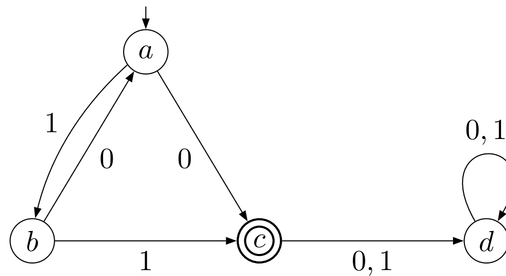
[i] The theoretical analogue of a *segmentation fault*: something is supposed to be there, but isn't.

Practically speaking, when δ is a *total* transition function it precludes the possibility of undefined transitions like that of Example 57.

Definition 54 (Complete FSM)**.** $M = (Q, \Sigma, s_0, \delta, F) \in$ FSM is COMPLETE when δ is TOTAL and INCOMPLETE otherwise.

Completing Graphs

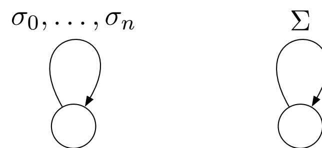
It may seem incomplete machines are not of much worth—and to some extent this is true. However, this incompleteness[ii] can be easily rectified. Consider this simple rewriting of M_4:



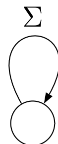
M_5

(d) is called a SINK STATE because all input at c is taken to d and becomes trapped.

Notation. Let $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$, we abuse[iii] notation and say the following are equivalent.



Definition 55 (sink state)**.** Any state which can transition only to itself. E.g. (not drawing incoming transitions):



[ii] Not to be confused with the incompleteness Gödel proved. [iii] We characterize this as abuse for transitioning on Σ technically indicates a move on the entire set, not the individual letters.

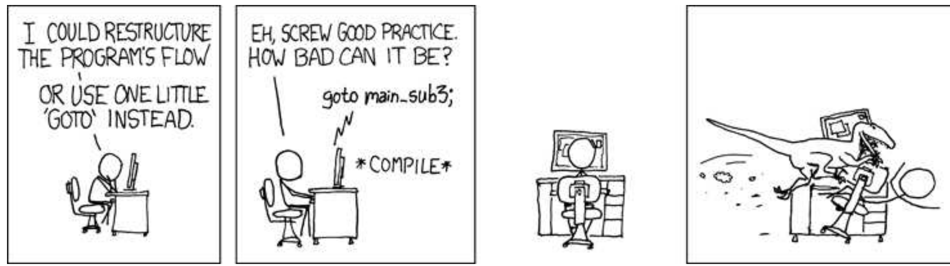


Figure 2.1: `xkcd.com/292`

In general *any* complete graph can be 'completed' by adding a sink state. Here, the natural theorem to write would say:

> For every *incomplete* FSM there is an equivalent *complete* FSM.

However at the moment we lack the tools to express this with sufficient mathematical rigour. In particular, we cannot show the language generated by $M \in \text{FSM}$ is invariant to the addition of sink state until we first describe precisely what it means for 'M to generate a language L'!

§2.2 The language of a Machine

Simply put, the language a machine generates is the set of input words that 'take' M to a final state. Expressing this mathematically requires some work.

§Goto

We first need to formalize what is meant by:

$$qw \vdash \cdots \vdash q',$$

which was intended to convey

> Input word w, starting at q, terminates at q'.

Definition 56 ($\vdash$)**.** The GOTO *function* is given by:

$$\begin{aligned} \vdash : Q\Sigma^* &\to Q\Sigma^* \\ qw_0w &\mapsto q'w \end{aligned}$$

with $w_o w$ the concatenation of $w_0 \in \Sigma$ and $w \in \Sigma^*$. (Read $qw_0 \vdash q'$ as "q goes to q' on w_0".)

Writing w_0w is a convenient way for us to remove the first letter of a word.

The intention is to have a function which outputs $q'w$ given qw_0w (i.e. a function telling us where to move next while depleting the input).

We use $\vdash_{\mathrm{R}}$ and $\vdash_{\mathrm{F}}$ to speak about goto as a relation or as a function. However, when it is clear from context[iv] we will simply use $\vdash$.

An integer *function* $f(x)$ can easily be used to generate:

$$\{(x,\, f(x)) \,:\, x \in \mathbb{Z}\}\,,$$

which is trivially a relation.

In this fashion $\vdash_{\mathrm{F}}$[v] generates:

$$`\vdash_{\mathrm{R}}\text{'} = \{(qw, \vdash_F (qw_0w)) \,:\, w \in \Sigma \wedge q \in Q\}\,.$$

Thus, although we have chosen to define goto as a function it can also be given as a relation[vi].

$$`\vdash_{\mathrm{R}}\text{'} = \left\{ \left(qw_0w, q'w\right) \,:\, (q) \xrightarrow{w_0} (q') \right\}.$$

In some sense, $qw_0w \vdash_{\mathrm{R}} q'w$ answers:

Does q go directly to q' on w_0w?

Whereas $\vdash_{\mathrm{F}} (q, w_0w)$ answers:

Where does q go on w_0w?

A subtle—yet important—difference.

§Eventually goes to ($\vdash^*$)

Notation ($\trianglelefteq$)**.** Let $\trianglelefteq$ denote subword and $\triangleleft$ denote strict subword.

Exercise 30. $\top$ or $\bot$? If $w = w_0w_1 \cdots w_n$, then

$$\forall \ell \in \{0, \ldots, n\}\,;\, w_0 \cdots w_\ell \triangleleft w.$$

[iv] 'Clear from context', is another shortcut used commonly in mathematical writing. It means ambiguous notation will be used when ambiguity is not possible. [v] The 'F' is for function. [vi] The 'R' is for relation.

Viewing $\vdash$ as a relation, let us show $qw \vdash^* qw'$ means: there is a subword of w taking q to q'.

We have established calculating R^* merely requires us to directly relate objects which are related via some transitive chain. In this context,

$$qw \vdash^* q'w' \iff \exists u = u_0 \cdots u_\ell \trianglelefteq w \in \Sigma^* \, : \, qu_0 \cdots u_\ell \vdash q_1 u_1 \cdots u_\ell \vdash \cdots \vdash q_\ell u_\ell \vdash q'\varepsilon$$

In fact this is already sufficient to demonstrate our original assertion, but let us go into more detail anyways.

Let us simplify the relation ‘ $\vdash$ ’ and say

$$\vdash = \left\{ \left(qw_0, \, q'\right) \, : \, \exists w_0 \in \Sigma; \; (q) \xrightarrow{w_0} (q') \right\}$$

That is, we do the sensible thing and only define transitions on letters (necessarily finite) and not words (usually infinite).

We generate $\vdash^*$ through repeated invocation of $\circ$ (relation composition).[vii] Namely,

$$\vdash^0 = \{(q, q) \, : \, q \in Q\}$$

which says any state can get to itself by not moving (more properly: $\forall q \in Q$; $q\varepsilon \vdash q$), and

$$\vdash^1 = \vdash$$

$$\vdash^2 = \left\{ \left(qw_0w_1, \, q''\right) \, : \, \exists w_0, w_1 \in \Sigma; \; (q) \xrightarrow{w_0} (q') \xrightarrow{w_1} (q'') \right\}$$

$$\vdots$$

$$\vdash^\ell = \left\{ \left(qw_0 \cdots w_{\ell-1}, \, q^{(\ell)}\right) \, : \, \exists w_0, \ldots, w_{\ell-1} \in \Sigma; \; (q) \xrightarrow{w_0} \; \cdots \; \xrightarrow{w_{\ell-1}} (q^{(\ell)}) \right\}$$

The following two propositions insist this process terminates/stabilizes.

Proposition 18. Let $M = (Q, \Sigma, s_0, \delta, F) \in$ FSM, then

$$|Q| < \infty \implies |\vdash| < \infty.$$

Proof. Placeholder. ∎

Proposition 19. Let $M = (Q, \Sigma, s_0, \delta, F) \in$ FSM, then

$$\exists \ell \in \mathbb{N} \, : \, \ell < \infty \, \wedge \, \vdash^\ell = \vdash^{\ell+1} = \vdash^*$$

Proof. Placeholder. ∎

[vii] By setting $R = \vdash$ in what comes after Chapter 1 Example 25.

The important lesson to take from this demonstration is

$$qw_0 \cdots w_{-1} \vdash^* q' \iff (q) \xrightarrow{w_0} \cdots \xrightarrow{w_{-1}} (q'),$$

in prose means:

1. there is some path (sequence of directed edges) taking q to q', and additionally
2. the word through this path is w.

That is, even though we only define moves on *letters* this is sufficient to move on *words*.

It should now be clear our characterization of $qw \vdash^* q\varepsilon$ meaning: "q *eventually* goes to q' on w" is not only correct, but entirely consistent with the definition of the transitive reflexive closure of $\vdash$. To wrap up a loose end, let us extend this functionality to $qw \vdash^* q'w'$, which does *not* insist the input w is exhausted. Simply take

$$qw \text{ eventually goes to } q'w'$$

to mean

$$\exists u \trianglelefteq w \ : \ qu \vdash^* q'\varepsilon$$

(there is some subword of w that takes q to q').

Definition 57 (Language of M)**.** The LANGUAGE OF $M = (Q, \Sigma, s_0, \delta, F) \in$ FSM is denoted $L(M)$ and given by

$$L(M) = \{w \in \Sigma^* \ : \ f \in F \wedge s_0 w \vdash^* f\}.$$

(Words that take s_0 to *any* final state.)

Definition 58 (Machine Equivalence)**.** $M_1 = (\Sigma, Q, s_0, \delta, F) \in$ FSM and $M_2 = (\Sigma, Q', s_0', \delta') \in$ FSM are EQUIVALENT MACHINES when

$$L(M_1) = L(M_2).$$

Even though Definition 58 presumes M_1 and M_2 are given over the same Σ, more generally we need only insist there is some 1-1 mapping between Σ_1 and Σ_2. This is reasonable if, for example, we do not care to distinguish '01' from 'ab'.

> '01' and 'ab' are equal under the mapping $0 \mapsto a$ and $1 \mapsto b$.

To simplify our presentation we assume the alphabets are identical.

We finally have the tools to show any incomplete machine can be 'completed' without effecting the language it generates.

Proposition 20.

$$\forall M \in \text{FSM}; \exists \text{ complete } M' \in \text{FSM} \,:\, L(M) = L(M').$$

Proof. Placeholder. □

§2.3 Regular Languages

Definition 59 (Regular Language)**.** A language $\mathbb{L}$ is a REGULAR LANGUAGE when

$$\exists M \in \text{FSM} \;:\; \mathbb{L} = L(M)$$

and an IRREGULAR LANGUAGE otherwise.

Notation. Denote the set of all regular languages by

$$\mathbb{L}_{\text{REG}} = \{\mathbb{L} \,:\, \exists M \in \text{FSM};\; \mathbb{L} = L(M)\}\,.$$

Exercise 31. Draw a state diagram of M satisfying $L(M) = \mathbb{L}_1$ for the language $\mathbb{L}_1 = \{w \in \{0,1\}^* \,:\, |w|_0 = 3\}$.

To demonstrate our model of computation (thus far) is still insufficient, consider this innocent change to Exercise 31:

$$\mathbb{L}_2 = \{w \in \{0,1\}^* \,:\, |w|_0 = |w|_1\}\,.$$

There is no FSM generating $\mathbb{L}_2$.

> Try to find one—however, keep in mind this advice from William Claude Dukenfield (American comedian, writer and juggler):
>
> *"If at first you dont succeed, try, try, again. Then quit. There's no use being a damn fool about it."*

But how can we dismiss the possibility of simply being unable to produce a solution? Needless to say: premising 'no such machine exists' on 'because I am unable to find one' does not a good proof make.

§FSM Shortcuts

Let us convince ourselves

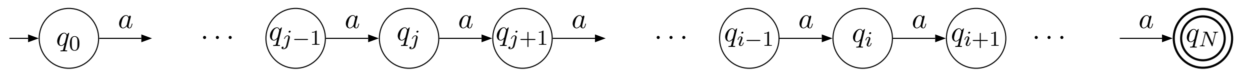
$$L\left(\rightarrow \bigcirc \xrightarrow{a} \bigcirc \xrightarrow{a} \bigcirc \xrightarrow{a} \circledcirc \right) = \{aaa\}\,.$$

Moreover, the removal of any state 'breaks' the machine.

More generally accepting only a^N requires (at least) $N+1$ states. To reason out why (at least superficially) we use PHP and contradiction.

> $\{a^n : n \in \mathbb{N}\}$ and $\{a^N\}$ with $N \in \mathbb{N}$ are very different sets. The former has an infinite number of words, the later only a^N.

Fixing $N \in \mathbb{N}$ there is an *accepting* configuration for a^N using $N+1$ states resembling:



(The states are not necessarily distinct, we are merely arguing the existence of a path like this.)

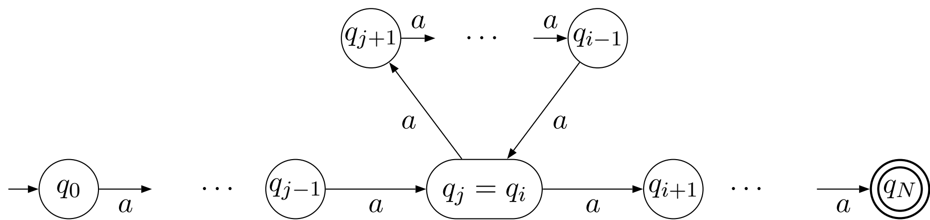
TAC suppose $|Q| \leq N$:

$$\{q_0, q_1, \ldots, q_N\} \subseteq Q \implies |\{q_0, q_1, \ldots, q_N\}| \leq |Q| \leq N$$

and consequently PHP assures $\exists i, j \in \mathbb{N} : i > j \wedge q_i = q_j$.

> As written $\{q_0, q_1, \ldots, q_N\}$ requires $N+1$ distinct states. As there are only N available states, two states must be equivalent.

Exploiting this equivalence take $q_i = q_j$ and redraw the accepting path as



This introduces a "$(i-j)$ length shortcut" allowing $a^{N-(i-j)}$ with $(i-j) > 0$ to skip a *non-zero*

> We only care about the *existence* of a shortcut. The actual non-zero length of the shortcut is irrelevant to our proof.

number of states and become accepted. ↯

Let us give a formal presentation of the above.

Proposition 21. For any fixed $N \in \mathbb{N}$ and $M \in \textsc{fsm}$

$$L(M) = \left\{a^N\right\} \implies |Q| > N.$$

(At least $N+1$ states are required to accept a^N and reject $a^M : M \neq N$.)

Proof. Fix $N \in \mathbb{N}$ and TAC suppose

$$\exists M = (Q,\, \Sigma,\, s_0,\, \delta,\, F) \in \textsc{fsm} : |Q| \leq N \wedge L(M) = \left\{a^N\right\}.$$

Clearly $a^N \in L(M)$ so

$$s_0 a^N \vdash^* q_N$$

where $q_N \in F$.

However, this accepting configuration sequence, namely

$$q_0 a^N \vdash q_1 a^{N-1} \vdash \cdots \vdash q_i a^{N-i} \vdash \cdots \vdash q_N \varepsilon$$

[viii] or, more succinctly

$$q_0 a^N \vdash q_1 a^{N-1} \vdash^* q_i a^{N-i} \vdash^* q_N \varepsilon$$

moves amongst $\{q_0, \ldots, q_N\} \subseteq Q$. As $|\{q_0, \ldots, q_N\}| \leq |Q| \leq N$ PHP ensures $\exists i, j \in \mathbb{N} : i > j \wedge q_i = q_j$.

This equivalence enables us to build the following accepting configuration for $a^{N-(i-j)}$:

$$q_0 a^{N-(i-j)} \vdash^* q_j a^{N-i} \vdash^0 q_i a^{N-i} \vdash^* q_N \varepsilon$$

implying $a^{N-(i-j)} \in L(M) : i-j > 0$. ↯ [ix] □

[viii] Note: $a^{N-N} = \varepsilon$. [ix] Do not distress if this proof seems difficult. It will become clear as you practice.

§Irregular Languages

The inherent weakness of a FSM is its inability to count. For instance $\mathbb{L} = \{a^n b^n : n \in \mathbb{N}\}$ has a counting requirement—we must count the a's to determine if there are an equivalent number of b's.

By Definition 59 an IRREGULAR language $\mathbb{L}$ satisfies

$$\forall M \in \text{FSM};\ L(M) \neq \mathbb{L}. \tag{2.1}$$

Proving (2.1) directly requires we enumerate—infinitely many—machines and demonstrate each does not generate $\mathbb{L}$ (impossible). However, it *is* possible to prove the equivalent expression

$$\neg\exists M \in \text{FSM} : L(M) = \mathbb{L}.$$

by assuming such a M exists and deriving contradiction.

Proposition 22. $\mathbb{L} = \{a^n b^n : n \in \mathbb{N}\}$ is not a regular language.

Proof. TAC suppose

$$\exists M = (\Sigma, Q, q_0, \delta, F) \in \text{FSM} : L(M) = \mathbb{L}$$

and let $N = |Q|$ and $q_{2N} \in F$.

Consider the accepting configuration sequence for $a^N b^N \in L(M)$

$$q_0 a^N b^N \vdash q_i a^{N-i} b^N \vdash^* q_{2N}\varepsilon.$$

As

$$\{q_0, \ldots, q_N\} \subseteq Q \implies |\{q_0, \ldots, q_N\}| \leq |Q| = N$$

PHP ensures $\exists i, j \in \mathbb{N} : (j < i < N) \wedge q_i = q_j$. Consequently,

$$q_0 a^{N-(i-j)} b^N \vdash^* q_j a^{N-i} b^N \vdash^0 q_i a^{N-i} b^N \vdash^* q_{2N}\varepsilon$$

is an accepting configuration.

Thus $a^{N-(i-j)} b^N \in L(M) : i - j > 0.$ ↯ □

Exercise 32. Is $\mathbb{L} = \{a^n b : n \in \mathbb{N}\}$ regular?

§FSM Scenic Routes

Instead of exploiting circuits (by removing them) to generate words with missing pieces, we can traverse the circuit (ad nauseum) to generate words

with *extra* pieces instead.

Example 58. An alternate proof of

$$\mathbb{L} = \{a^n b^n : n \in \mathbb{N}\} \notin \mathbb{L}_{\text{REG}}$$

(Proposition 22).

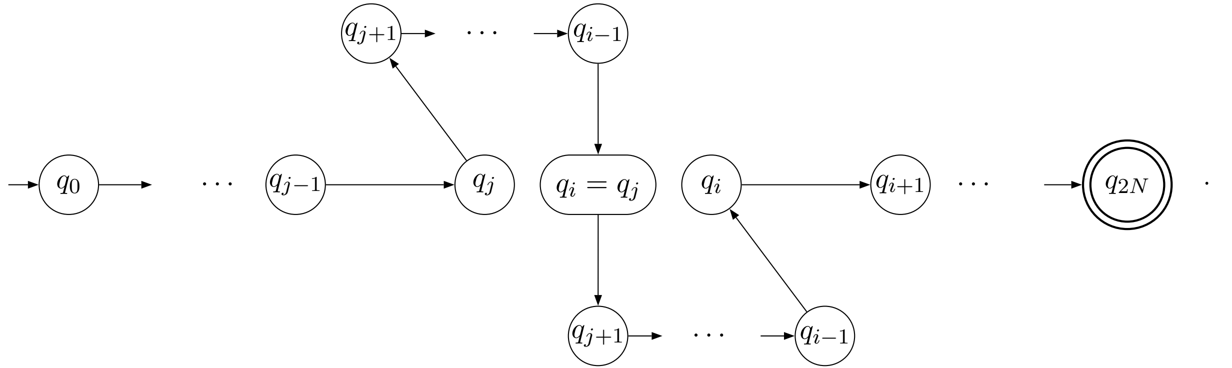
TAC assume $\exists M = (Q, \Sigma, q_0, \delta, F) \in \textsc{fsm} : \mathbb{L} = L(M)$. Let $N = |Q|$ and consider $a^N b^N \in \mathbb{L}$.

It follows

$$q_0 a^N b^N \vdash^* q_{2N} \in F$$

and PHP ensures $\exists i, j \in \mathbb{N} : (j < i < N) \wedge q_i = q_j$.

Notice this generates a $(i - j)$-length 'scenic route'



Which, practically speaking, allows us to shed an arbitrary number of a^{i-j}s from the input word.

Consider

$$a^N a^{i-j} b^N,$$

a word constructed to travel the circuit (exactly) one more time than necessary. This word has the *accepting* configuration sequence

$$q_0 a^{N+(i-j)} b^N \vdash^* q_i a^{N-j} b^N \vdash^0 q_j a^{N-j} b^N \vdash^* q_{2N} \in F$$

and thus $a^{N+(i-j)} b^N \in \mathbb{L} : (i - j) > 0$. ↯

§Pumping Lemma

Pumping lemma is the formalization of FSM shortcuts/scenic routes. It enables us to eschew the *explicit* application of PHP and use it (more rapidly) *implicitly.*

Theorem 6 (Pumping Lemma)**.** For any $\mathbb{L} \in \mathbb{L}_{\text{REG}}$

$$\exists N \in \mathbb{N};\, \forall x \in \mathbb{L};\, |x| \geq N \implies \exists u, v, w \in \Sigma^*;\, x = uvw\ :$$

1. $|uv| \leq N$,
2. $|v| \geq 1$, and
3. $\forall i \in \mathbb{N};\, uv^i w \in \mathbb{L}$.

Definition 60 (Pumping length)**.** The N of Theorem 6 is called the pumping length.

Pumping lemma ensures all words of a regular language longer than some PUMPING LENGTH can be decomposed (in a specific way) into three pieces. Moreover, repeating (or 'pumping') the middle piece just generates other words in the language. Thus,

> **if** a language has a word without such a decomposition **then** the language is not regular.

Warning: Irregular languages can satisfy pumping lemma as well. Consequently, *pumping lemma cannot be used to show a language is regular.* Using pumping lemma to 'prove' a language is regular is a widely made mistake. Remember the contrapositive is

$$[a \implies b] \equiv [\neg b \implies \neg a]$$

so

$$[\mathbb{L} \in \mathbb{L}_{\text{REG}} \implies \mathbb{L} \text{ satisfies PL}] \equiv [\mathbb{L} \text{ fails PL} \implies \mathbb{L} \notin \mathbb{L}_{\text{REG}}]\,.$$

Using pumping lemma to show a language is regular *incorrectly* presumes

$$[a \implies b] \equiv [b \implies a]\,.$$

Proof of Pumping Lemma. Let $\mathbb{L} \in \mathbb{L}_{\text{REG}} \overset{\text{def.}}{\implies} \exists M : \mathbb{L} = L(M)$. Let $M = (Q,\, \Sigma,\, q_0,\, \delta,\, F)$ and $N = |Q|$.

Any word $x = x_0 \cdots x_{N-1} \in L(M) : x_i \in \Sigma$ has an accepting configuration

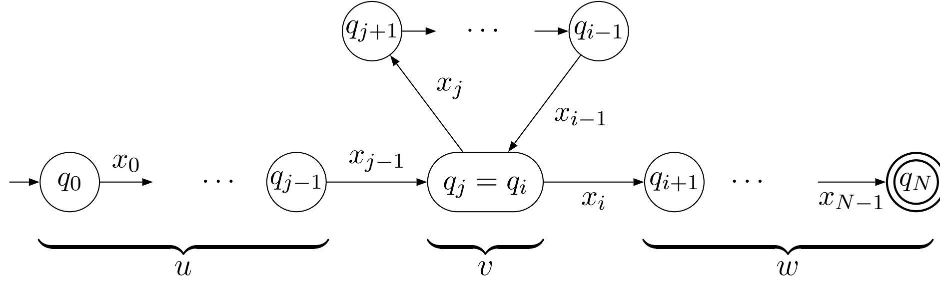
$$q_0 x_0 \cdots x_N \vdash q_1 x_1 \cdots x_{N-1} \vdash^* q_N \varepsilon \in F$$

requiring

$$\begin{aligned} \{q_0, \dots, q_N\} \subseteq Q &\implies |\{q_0, \dots, q_N\}| \leq |Q| \\ &\implies |\{q_0, \dots, q_N\}| \leq N. \end{aligned}$$

Thus, by PHP $\exists i, j \in \mathbb{N} : i > j \wedge q_i = q_j$ in this configuration sequence.

Consider $u, v, w \in \Sigma^*$ as given below:



Namely $u = x_0 \cdots x_{j-1}$, $v = x_j \cdots x_{i-1}$, and $w = x_i \cdots x_{N-1}$ where $q_0 uvw \vdash^* q_j vw \vdash^* q_i w \vdash^* q_N \varepsilon$.

Consequently,

$$|uv| = |u| + |v| = i + (j - i) \overset{\text{ass.}}{=} j \leq N \implies 1., \text{ and}$$

$$|j < i| \implies 0 < i - j \implies |v| = i = j \geq 1 \implies 2.$$

Finally, notice $q_j v \vdash^* q_i \overset{q_i = q_j}{\implies} q_j v \vdash^* q_j$. A simple induction gives

$$\forall k \in \mathbb{N}; q_j v^k \vdash^* q_j \tag{2.2}$$

which validates

$$q_0 u v^k w \vdash^* q_j v^k w \vdash^* q_j w \vdash^0 q_i w \vdash^* q_n \varepsilon.$$

Thus 3.. □

Example 59. Proving

$$\mathbb{L} = \{a^n b^n : n \in \mathbb{N}\} \notin \mathbb{L}_{\text{REG}}$$

with pumping lemma.

TAC let $M = (Q, \Sigma, s_0, \delta, F) : L(M) = \mathbb{L}$ and N be the pumping distance. Consider $a^N b^N \in \mathbb{L}$. By pumping lemma, $\exists u, v, w \in \Sigma^* : uvw = a^N b^N$ and

1. $|uv| \leq N \implies |uv|_b = 0$ (i.e. $uv = a \cdots a$),
2. $|v| \geq 1 \implies v = a^\ell : \ell \geq 1$
3. $\forall k \in \mathbb{N}; uv^k w \in L(M)$.

> We only need consider a *single* value of k to derive a contradiction—setting $k = 0$ gives Proposition 22 whereas $k = 2$ is the method of Example 58. Either way, we have contradiction.

Setting $k = 0$ we deduce

$$\begin{aligned} uv^0 w \in L(M) &\implies \left|uv^0 w\right|_a = \left|uv^0 w\right|_b \\ &\implies N - \ell = N : \ell > 0. \lightning \end{aligned}$$

Alternatively, setting $k = 2$:

$$\begin{aligned} uv^2 w \in L(M) &\implies \left|uv^2 w\right|_a = \left|uv^2 w\right|_b \\ &\implies N + \ell = N : \ell > 0. \lightning \end{aligned}$$

Sometimes applying pumping lemma requires a case analysis.

Example 60. Proving

$$\mathbb{L} = \{ww : w \in \{0,1\}^*\} \notin \mathbb{L}_{\text{REG}}$$

with pumping lemma.

Sketch of proof. Let N be the pumping length and consider

$$10^N 10^N = uvw.$$

Since $|uv| \leq N$ either

1. $u = 10^\ell : \ell \geq 0$ and $v = 0 \cdots 0$, or
2. $u = \varepsilon$ and $v = 1 \cdots 0$

In either case contradiction can be derived by removing v (i.e. setting $k = 0$).

Exercise 12. Over $\Sigma = \{0, 1\}$ prove

$$\mathbb{L} = \{w : |w|_0 < |w|_1 - 1\} \notin \mathbb{L}_{\text{REG}}.$$

Exercise 13. Over $\Sigma = \{a\}$ prove

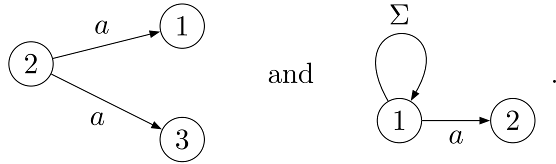
$$\mathbb{L} = \left\{a^{n^2} : n \in \mathbb{N}\right\} \notin \mathbb{L}_{\text{REG}}.$$

Exercise 14. Over $\Sigma = \{a\}$ prove

$$\mathbb{L} = \{a^p : p \text{ is prime}\} \notin \mathbb{L}_{\text{REG}}.$$

§2.4 Nondeterministic FSM

Our goal is to extend the expressional power of our state diagrams by formalizing nondeterminism. To be less vague: we wish to occupy multiple states of a DFSM *simultaneously* allowing ambiguous moves like



This simply requires us to 'upgrade' the transition (and goto) function:

$$\begin{aligned} \delta : Q\Sigma &\to \mathscr{P}(Q) & \vdash : Q\Sigma &\to \mathscr{P}(Q) \\ qa &\mapsto \left\{q_0', \ldots, q_n'\right\} & \{q_0, \ldots, q_n\}\, a &\mapsto \delta(q_0, a) \cup \cdots \cup \delta(q_n, a). \end{aligned}$$

Now δ takes a single state to *many* states, and $\vdash$ defines a more robust configuration sequence where a *group of states* is mapped to a *group of states.*

Definition 61 (NDFSM)**.** A NONDETERMINISTIC FINITE STATE MACHINE (NDFSM) is a 5-tuple $(\Sigma, Q, s_0, \delta, F)$ where

Σ	input alphabet,
Q	set of states
$s_0 \in \mathscr{P}(Q)$	initial state,
$\delta : Q \times \Sigma \to \mathscr{P}(Q)$	'super'-state transition function, and
$F \in \mathscr{P}(Q)$	set of final states.

Exercise 15. It would have been simpler to let

$$\vdash : \mathscr{P}(Q) \to \mathscr{P}(Q)$$
$$\{q_0, \ldots, q_n\}\, a \mapsto \{q'_0, \ldots, q'_m\}\,.$$

Why did we not?

Notation ($\underline{x}$)**.** Let $\underline{x}$ denote $xu\,:\,x \in \Sigma \wedge u \in \Sigma^*$. (I.e. a word with first letter $x \in \Sigma$.)

To convince ourselves this is necessary.

Example 61. Draw a state transition diagram for reading

$$\mathbb{L} = \{u\underline{a}\,:\,u \in \Sigma^* \wedge |\underline{a}| = 3\}\,,$$

the language of words (over $\{a, b, c\}$) with third last letter a.

There is a recipe for this:

1. create states for the eight length three suffixes,
2. draw rules for transitioning between them,
3. label accepting states (those suffixes beginning with 'a'),
4. chose a viable initial state.

See Figure **??**.

Continuing like this causes our state diagrams to become prohibitively large.

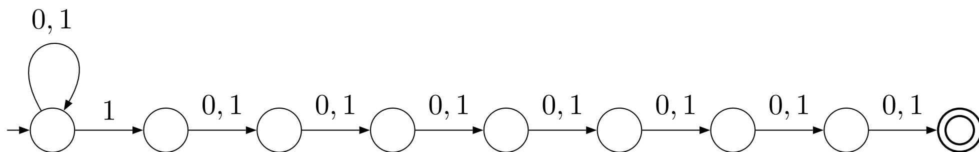
Example 62. $L(M) = \{u\underline{1}\,:\,u \in \{0,1\}^* \wedge |\underline{1}| = 8\}$. (Requires $2^8 = 256$ states and, for a complete graph,

$$2^{256} = 72\,057\,594\,037\,927\,936$$

transition rules.)

Using non-determinism, these machines are drawn much more compactly.

Example 63. A NDFSM for $\mathbb{L} = \{u\underline{1}\,:\,u \in \{0,1\}^* \wedge |\underline{1}| = 8\}$.



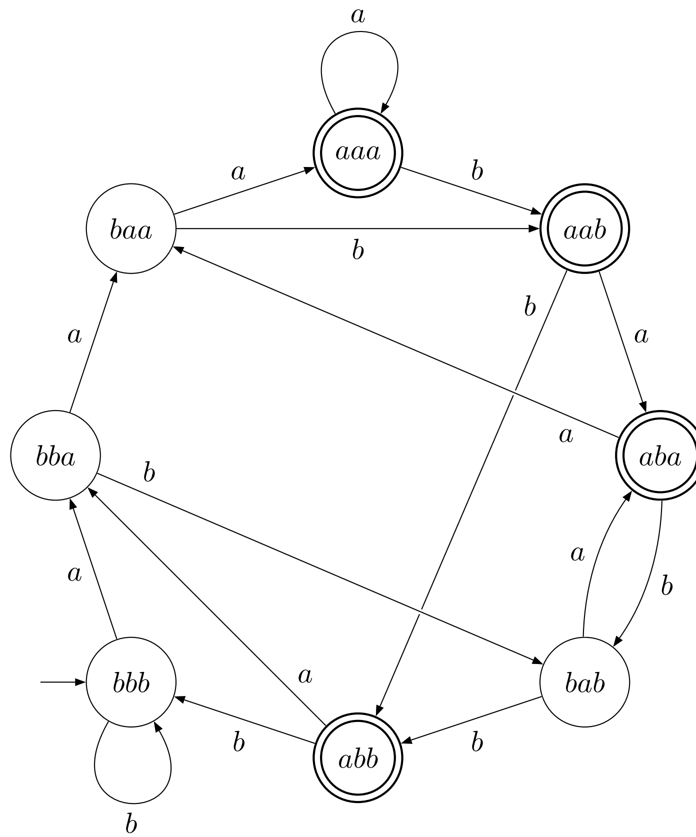
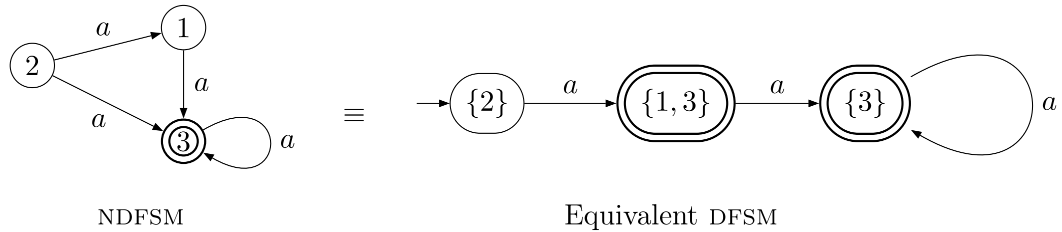


Figure 2.2: The machine for Example **??** accepting $\mathbb{L} = \{w\underline{a} \,:\, w \in [a,b]^* \wedge |\underline{a}| = 3\}$

Intuitively there is a surjection from states of aNDFSM into the powerset of those states. This surjection (which we will soon make explicit), essentially converts aNDFSM into a (truly) complete DFSM.

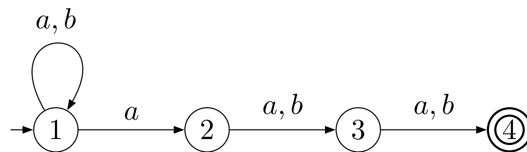
For instance



NDFSM Equivalent DFSM

With this new tool let us redo Example 61.

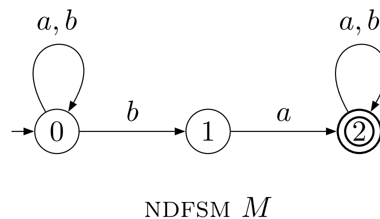
Example 64. A NDFSM machine for $\mathbb{L} = \{u\underline{a} \, : \, u \in \Sigma^* \, \wedge \, |\underline{a}| = 3\}$



The corresponding DFSM for this NDFSM is Figure 2.1. (The astute reader will recognize this as isomorphic to the machine of Example 61. It is this correspondence we wish to make precise.)

§NDFSM → DFSM CONVERSION

Expectantly (yet no less remarkably so) NDFSMs are no more powerful than the DFSMs they are derived from. A method for converting one to the other is sufficient to establish equivalence. We give two such methods and, for comparison, execute each on:



NDFSM M

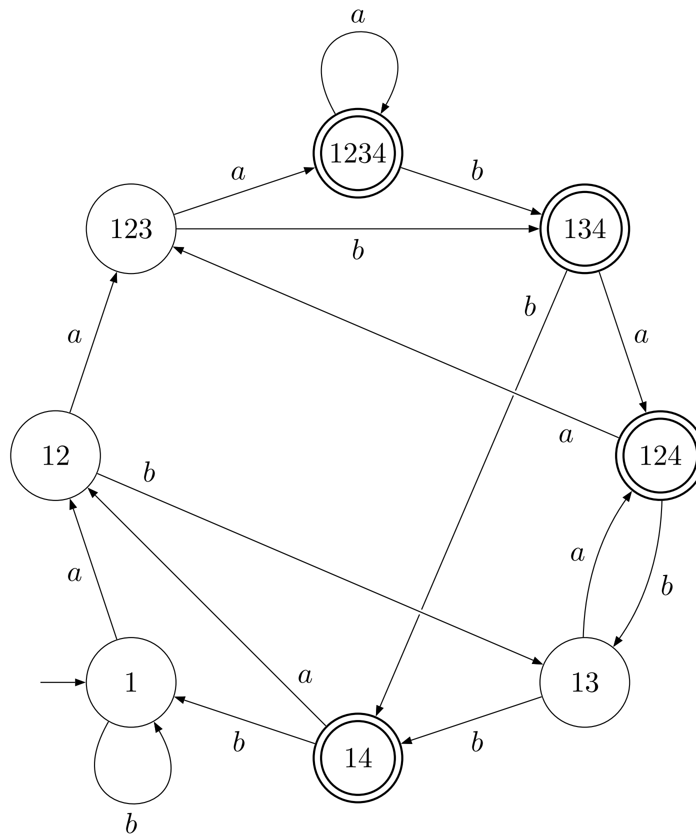


Figure 2.3: The corresponding DFSM for the NDFSM machine of Example ??: For clarity (and also to avoid huge states) we substitute 1 for $\{1\}$, 12 for $\{1, 2\}$, and so on.

$\mathscr{P}(Q)$	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0\}$	$\{0,1\}$
$\{1\}$	$\{2\}$	$\varnothing$
$\{2\}$	$\{2\}$	$\{2\}$
$\{0,1\}$	$\{0,2\}$	$\{0,1\}$
$\{0,2\}$	$\{0,2\}$	$\{0,1,2\}$
$\{1,2\}$	$\{2\}$	$\{2\}$
$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$

Table 2.1: The DFSM transition table for M' as generated by Subset Construction.

SUBSET CONSTRUCTION

Intuitively: draw the appropriate transition rules among *all* possible subsets of Q.

Input: NDFSM $M = (Q, \Sigma, \delta, s, F)$
Output: DFSM $M' = (Q', \Sigma, \delta', s', F') \,:\, L(M) = L(M')$
$F' \leftarrow \{q \,:\, q \in Q' \wedge q \cap F \neq \varnothing\}$;
for $x \in \mathscr{P}(Q)$ **do**
  for $a \in \Sigma$ **do**
    $\delta'(x, a) \leftarrow \{q' \,:\, \delta(q, a) = q' \wedge q \in x\}$
return $(\mathscr{P}(Q), \Sigma, \delta', \{s\}, F')$;

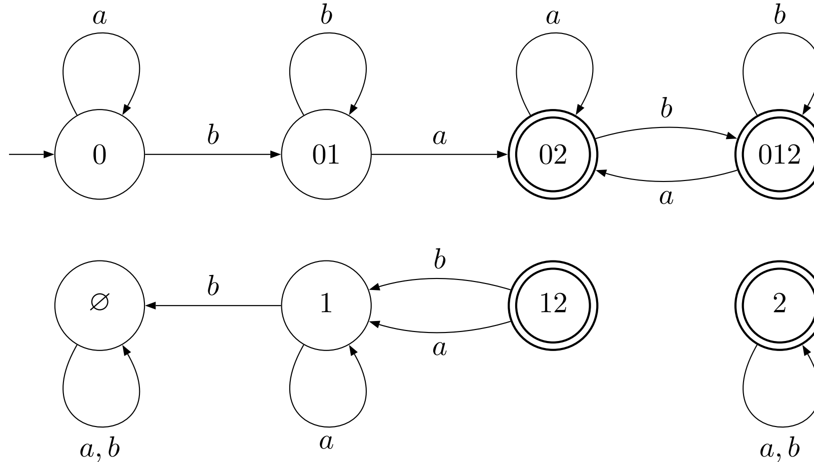
Algorithm 1: The Subset Construction

Remark 2 (alternative constructions)**.**

$$F' \leftarrow \{\{q_0, \ldots, q_n\} \,:\, \{q_0, \ldots, q_n\} \subset Q \wedge \exists q_i \in F\}$$
$$F' \leftarrow \{\text{all subsets of } Q \text{ containing (at least) one final state}\}$$
$$\delta(x = \{x_0, \ldots, x_n\}, a) \leftarrow \left\{q \,:\, (q) \xrightarrow{a} (x_0)\right\} \cup \cdots \cup \left\{q \,:\, (q) \xrightarrow{a} (x_n)\right\}$$
$$\delta'(x, a) \leftarrow \{\delta(q, a) \,:\, q \in x\} = \texttt{map}\,(q \mapsto \delta(q, a),\ \{x_0, \ldots, x_n\})$$

Applying subset construction to M generates transition Table 2.4.1. No-

tice we wasted computation by calculating transitions for unreachable states:



ITERATIVE SUBSET CONSTRUCTION

The ITERATIVE SUBSET CONSTRUCTION is an improvement of the subset construction; only transitions for reachable states are calculated.

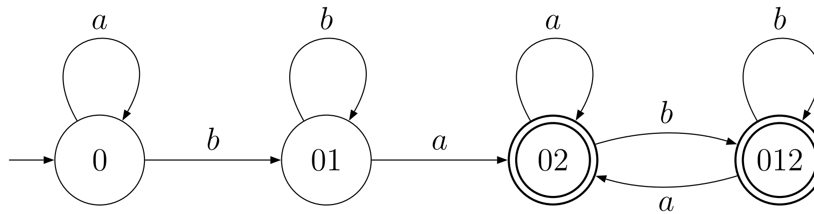
Input: NDFSM $M = (Q, \Sigma, \delta, s, F)$
Output: DFSM $M' = (Q', \Sigma, \delta', s', F') \,:\, L(M) = L(M')$
$Q' \leftarrow \{\{s\}\}$;
for $x \in Q'$ **do**
  for $a \in \Sigma$ **do**
    $\delta'(x, a) \leftarrow \bigcup_{q \in x} \delta(q, a)$;
    $Q' \leftarrow Q' \cup \{\delta'(x, a)\}$;
return $(Q', \Sigma, \delta', \{s\}, \{x \in Q' \,:\, x \cap F \neq \varnothing\})$;

Algorithm 2: The Iterative Subset Construction

Applying iterative subset construction to M generates transition Table 2.4.1. Notice only reachable states were considered.

$\mathscr{P}(Q)$	a	b
$\{0\}$	$\{0\}$	$\{0,1\}$
$\{0,1\}$	$\{0,2\}$	$\{0,1\}$
$\{0,2\}$	$\{0,2\}$	$\{0,1,2\}$
$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$

Table 2.2: The DFSM transition table for M' as generated by Subset Construction.



§NDFSM/DFSM EQUIVALENCE

Theorem 7 (NDFSM/DFSM equivalence)**.** For any NDFSM there is an equivalent DFSM.

$$\forall M \in \text{NDFSM} \; ; \exists M' \in \text{DFSM} \;\; : L(M) = L(M')$$

Proof. Proof by construction (aka, proof by algorithm). □

§2.5 PROPERTIES OF NDFSM LANGUAGES

We would like to establish

$$\exists M \in \text{FSM} \;\; : L(M) = \mathbb{L} \iff \mathbb{L} \in \mathbb{L}_{\text{REG}}.$$

That is, in addition to FSMs generating regular languages (trivially true by Definition **??**) we want to ensure each regular language is generated by some FSM. (The difference, although subtle, is important.)

Towards this goal notice a machine is just some 'combination' of

submachines and similarly a language is just some 'combination' of sublanguages. In particular, any regular language is constructible via:

1. $\varnothing \in \mathbb{L}_{\text{REG}}$,
2. $\forall a \in \Sigma; \{a\} \in \mathbb{L}_{\text{REG}}$,
3. $\mathbb{L}_1, \mathbb{L}_2 \in \mathbb{L}_{\text{REG}} \implies \mathbb{L}_1 \cup \mathbb{L}_2 \in \mathbb{L}_{\text{REG}}$,
4. $\mathbb{L}_1, \mathbb{L}_2 \in \mathbb{L}_{\text{REG}} \implies \mathbb{L}_1\mathbb{L}_2 \in \mathbb{L}_{\text{REG}}$, and
5. $\mathbb{L} \in \mathbb{L}_{\text{REG}} \implies \mathbb{L}^* \in \mathbb{L}_{\text{REG}}$.

(An idea we formalize and prove in the coming sections.)

Additionally a FSM can be built according to these rules as well (proving any regular language has a corresponding FSM which generates it).

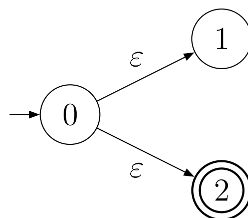
§ε-NDFSM

A consequence of nondeterminism is FREE MOVES: transitions on the empty word ε.

> In the machine analogy an ε-transition can be viewed as keeping the read head stationary while changing states.

These new transitions have a variety of uses. For instance, using ε-transitions we can enable multiple starting states.

Example 65. A machine with multiple starting states.



It is impossible to occupy only 0 since this state automatically sends us to 1 and 2 as well.

Exercise 16. What does the machine from Example 65 accept?

Machines with these transitions are called ε-NDFSMs.

Definition 62 (ε-NDFSM)**.** An EPSILON NONDETERMINISTIC FINITE STATE MACHINE is given by a 5-tuple $(Q, \Sigma, s_0, \delta, F)$ where

Σ	input alphabet,
Q	set of states,
$s_0 \in \mathscr{P}(Q)$	initial states,
$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$	transition function,
$F \in \mathscr{P}(Q)$	set of final states.

(The only change is the addition the ε-move to δ.)

The goto function is adjusted as follows:

$$p\varepsilon \vdash q \iff p \xrightarrow{\varepsilon} q \iff (p, \varepsilon, q) \in \delta.$$

Since

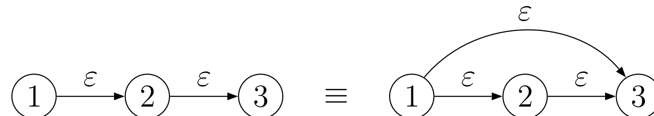
$$\forall w \in \Sigma^*; \forall n \in \mathbb{N}; \varepsilon^n w = w$$

this generalizes as expected.

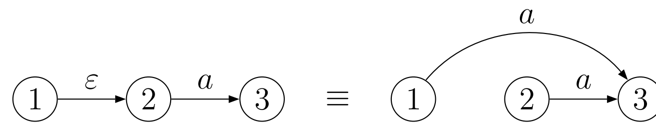
ε-NDFSM TO NDFSM CONVERSION

ε-NDFSMs are no more powerful than NDFSMs (which are in turn no more powerful than DFSMs).By providing an algorithm that converts the former into the later we establish equivalence as we did with ND/DFSMs.

The algorithm is based on the following observations:



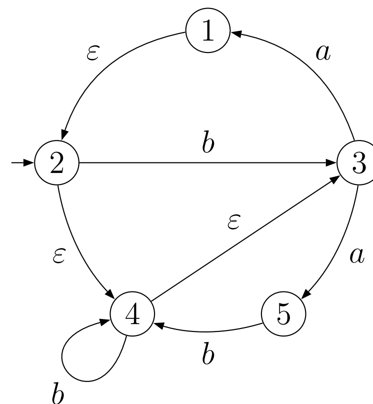
(ε-completion) and



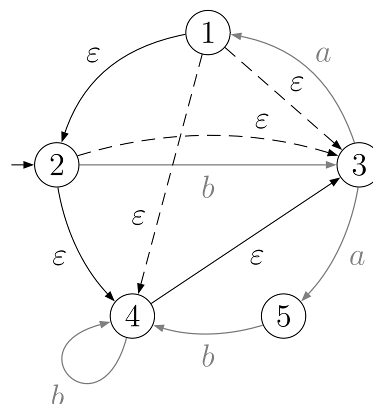
(ε-removal).

Doing an ε-completion is similar to finding the transitive closure but all non ε-transitions are ignored. Once this ε-CLOSURE is established the ε-transitions can be replaced by 'normal' Σ-based transitions.

Example 66. Consider the ε-NDFSM:



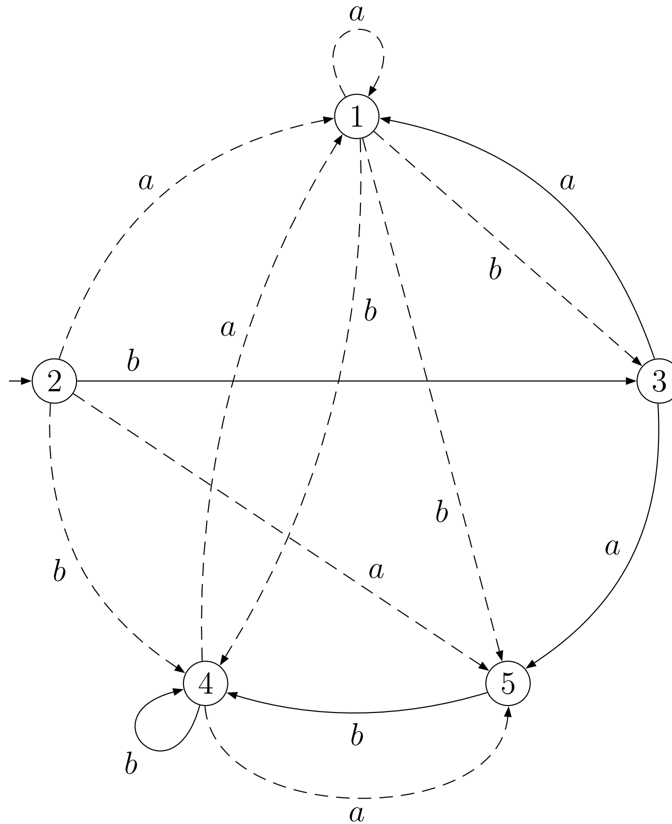
Its ε-closure is (dashed transitions are new, irrelevant transitions are greyed):



and the ε-removal

This is easier once you realize you only have to consider 'direct' ε-transitions.

corresponds to



Example 66 has no final states. In fact, the addition of final states complicates the process somewhat. However, observe

In general any state freely connected to any final state should be made final as well. (In Algorithm **??** this is done during ε-removal but can be correctly done as a preprocess, i.e. first.)

Example 67. If 3 was final state in Example 66 all but state 4 would be made final as well. (See Figure **??**.)

In general, the transformation from an ε-NDFSM toNDFSM is done

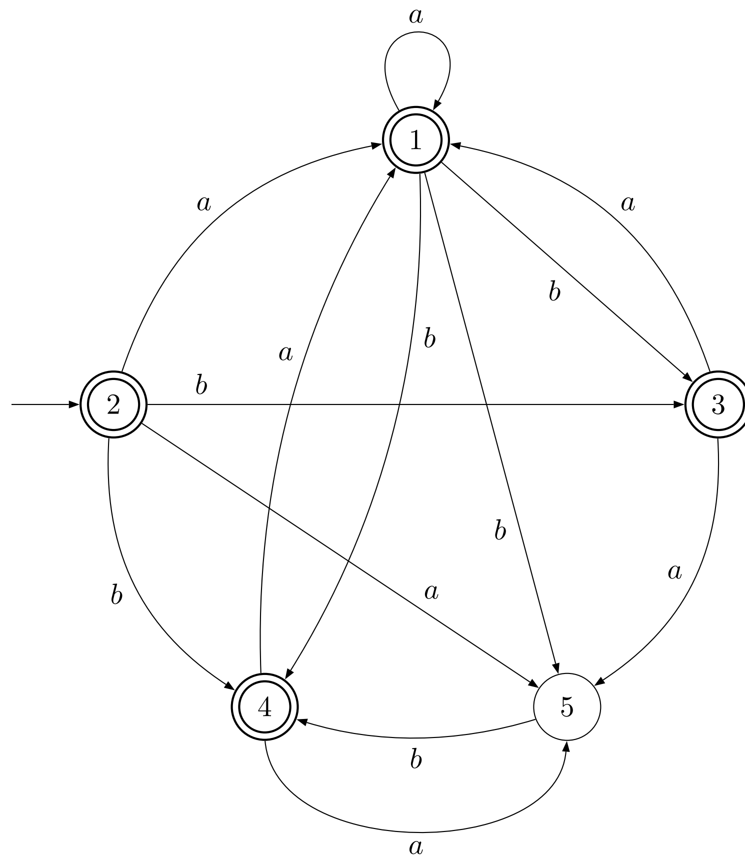


Figure 2.4: Example 66 with $3 \in F$.

by invoking the following algorithms one after each other.

```
Input: δ from ε-NDFSM M = (Q, Σ, δ, s, F)
Output: ε-completion of δ
δ' ← δ;
for (p, x, p') ∈ δ do
    for (q, y, q') ∈ δ do
        if p' = q ∧ x = ε = y then
            δ' ← δ' ∪ {(p, ε, q')};
if δ' = δ then
    return δ';
else
    return THISPROC (δ');
```

Algorithm 3: ε-completion

```
Input: ε-completed δ
Output: ε-free δ
δ' ← δ;
for (p, x, p') ∈ δ do
    for (q, a, q') ∈ δ do
        if p' = q ∧ x = ε ∧ y ≠ ε then
            δ' ← δ' ∪ {(p, a, q')};
return FILTER ( (p, x, p') → x ≠ ε, δ' );
```

Algorithm 4: ε-removal

THISPROC (or some variant) is available in many programming languages. It should be used when defining recursive functions so the function name can be changed without negative side-effect.

FILTER (or some variant) is available in many programming languages. It is used to remove members of a list based on some predicate. For us, we are removing 3-tuples which have ε-transitions.

Theorem 8. For any ε-NDFSM there is an equivalent NDFSM.

$$\forall\ \varepsilon\text{-NDFSM } M\ \exists \text{ NDFSM } M' \ :\ L(M) = L(M')$$

Sketch. Let $M = (Q,\ \Sigma,\ s_0,\ \delta,\ F)$. Given δ, Algorithm 3 and 4 produces

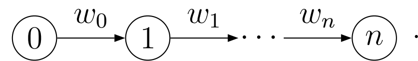
$$\delta' = \left(\delta \cup \left\{ (p) \xrightarrow{a} (r)\ :\ (q) \xrightarrow{\varepsilon} (q) \xrightarrow{a} (r) \in \delta \right\}\right) \setminus \left\{ (p) \xrightarrow{\varepsilon} (r)\ :\ p, q \in Q \right\}$$
$$= (\delta \cup \{(p, a, r)\ :\ (p, \varepsilon, q), (q, a, r) \in \delta\}) \setminus ((p, \varepsilon, q)\ :\ p, q \in Q)\,.$$

This is the δ' ofNDFSM $M' = (\Sigma, Q', s', \delta', F')$ and $L(M) = L(M')$. □

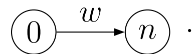
Exercise 17. Using the sketch as a guide, prove Theorem 8.

§Closure Properties

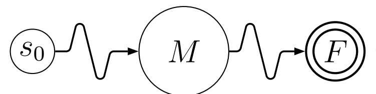
Notation. Let the machine accepting $w = w_0 \cdots w_n$



be denoted by



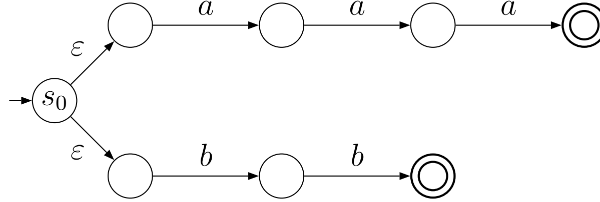
Notation (submachine)**.** The transition state diagram for $M = (\Sigma,\ Q,\ s_0,\ \delta,\ F)$ is abbreviated to



Union of Regular Languages

'Connect' two (or finitely many) machines using ε-transitions.

Example 68. A machine accepting $\{aaa\} \cup \{bb\}$



.

Theorem 9 (union)**.** The UNION of two regular languages is regular:

$$\mathbb{L}_1 = L(M_1) \wedge \mathbb{L}_2 = L(M_2) \implies \exists M_3 : \mathbb{L}_1 \cup \mathbb{L}_2 = L(M_3)$$

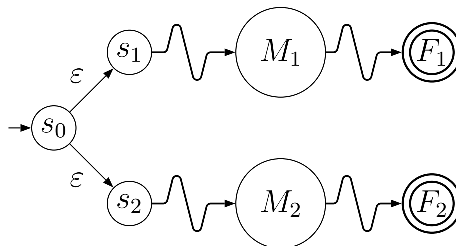
Proof. Given $\mathbb{L}_1 = L(M_1)$ and $\mathbb{L}_2 = L(M_2)$ and FSMs

$$M_1 = (\Sigma_1, Q_1, s_1, \delta_1, F_1), \text{ and}$$
$$M_2 = (\Sigma_2, Q_2, s_2, \delta_2, F_2)$$

construct an ε-FSM $M = (Q, \Sigma, s_0, \delta, F)$ with

$$Q = Q_1 \cup Q_2 \cup \{s_0\}$$
$$\Sigma = \Sigma_1 \cup \Sigma_2$$
$$F = F_1 \cup F_2$$
$$\delta = \delta_1 \cup \delta_2 \cup \left\{ s_0 \xrightarrow{\varepsilon} s_1 \;,\; s_0 \xrightarrow{\varepsilon} s_2 \right\}$$

Which builds a machine M with this shape:



.

Lemma 1. $L(M) = L_1 \cup L_2$.

A LEMMA is a sub-result towards the proof of the main result.

Let $w \in L(M)$ be arbitrary and take $f \in F_1 \cup F_2$. Either

$$s_0 \varepsilon w \vdash_M s_1 w \vdash^*_{M_1} f \implies w \in L(M_1),$$

or

$$s_0 \varepsilon w \vdash_M s_2 w \vdash^*_{M_2} f \implies w \in L(M_2).$$

Thus $L(M) \subseteq L(M_1) \cup L(M_2)$.

Conversely,

$$\begin{aligned} w \in L(M_1) \cup L(M_2) &\implies w \in L(M_1) \lor w \in L(M_2) \\ &\implies s_1 w \vdash^*_{M_1} f_1 \in F_1 \lor s_2 w \vdash^*_{M_2} f_2 \in F_2 \\ &\implies \{s_0 \varepsilon\}\, w \vdash \{s_1, s_2\}\, w \vdash^* f \subseteq F_1 \cup F_2 \\ &\implies w \in L(M) \end{aligned}$$

Thus $L(M_1) \cup L(M_2) \subseteq L(M)$ and moreover

$$L(M) = L(M_1) \cup L(M_2).$$

So there is some ε-FSM (orNDFSM or DFSM) M generating $L(M_1) \cup L(M_2)$. Thus by Definition **??** $L(M_1) \cup L(M_2)$ is regular. □

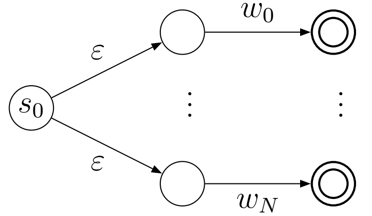
Theorem 10. Any finite language is regular:

$$\mathbb{L} \subset \Sigma^* \land |\mathbb{L}| < \infty \implies \mathbb{L} \in \mathbb{L}_{\text{REG}}.$$

Informally, any finite language

$$\mathbb{L} = \{w_0, \ldots, w_N\}$$

is accepted by



Provided $N < \infty$ this machine has a finite number of states (and thus is a FSM).

Proof. Let M_w be aFSM given by

$$\begin{aligned}\Sigma &= \{w_0, \ldots, w_n\}\\ Q &= \{q_0, \ldots, q_n\}\\ s_0 &= q_0\\ \delta(q_i, w_i) &= w_{i+1}\\ F &= \{q_n\}\end{aligned}$$

which is constructed to accept $w = w_0 \cdots w_n$.

By Theorem 10 $L(M_{w_i}) \cup L(M_{w_j})$ is regular, and thus a simple induction proves

$$L(M_{w_0}) \cup \cdots \cup L(M_{w_N})$$

for any $N < \infty$. □

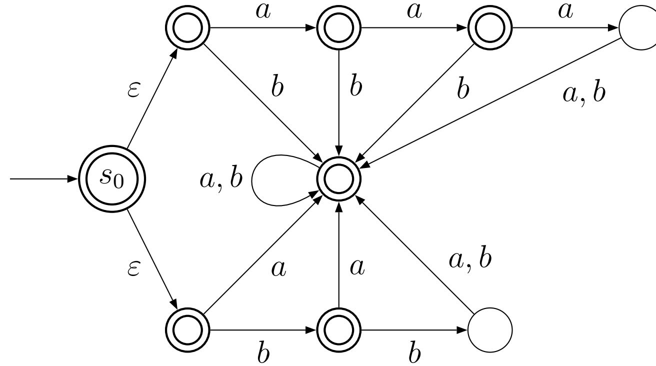
Exercise 18 (Midterm question).

Complementation of Regular Languages

'Swap' final and non-final states of a *complete* machine.

Example 69. The complement of Example 68: a machine accepting

$$\overline{\{a^3\} \cup \{b^2\}} = \left\{a^i \,:\, i \neq 3\right\} \cup \left\{b^i \,:\, i \neq 2\right\} \subseteq \{a, b\}^*$$



Notice the appearance of the previously unwritten sink state.

Exercise 19. Give a state transition diagram for M such that

$$L(M) = \overline{\{a^3\} \cup \{b^2\}} \subseteq \{a, b, c\}^* .$$

Theorem 11 (complement)**.** The COMPLEMENT of a regular language over a finite alphabet is regular:

$$\exists M : \mathbb{L} = L(M) \implies \exists \overline{M} : \overline{\mathbb{L}} = L(\overline{M}).$$

Proof. Let $\mathbb{L} = L(M)$ be generated by $M = (Q, \Sigma, s_0, \delta, F)$, a *complete* FSM. Define $\overline{M} = (\Sigma, Q, s_0, \delta, Q - F)$.

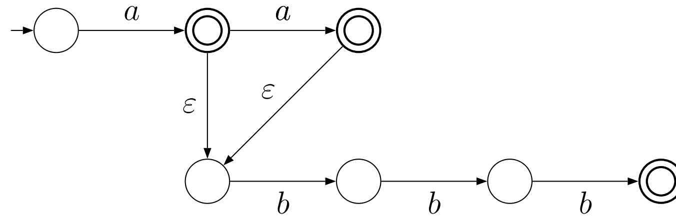
Lemma 2. $L(\overline{M}) = \overline{L(M)}$

Exercise. □

Concatenation of Regular Langauges

We connect the final state*s* of one machine with the starting state of the other.

Example 70. Let $\mathbb{L}_1 = \{aa, a\}$ and $\mathbb{L}_2 = \{bbb\}$ then



is a transition state diagram for $\mathbb{L}_1\mathbb{L}_2$.

Theorem 12 (concatenation)**.** The CONCATENATION of two regular is regular:

$$\mathbb{L}_1 = L(M_1) \wedge \mathbb{L}_2 = L(M_2) \implies \exists M : \mathbb{L}_1\mathbb{L}_2 = L(M)$$

Proof. Given $M_1 = (\Sigma_1, Q_1, s_1, \delta_1, F_1)$ and $M_2 = (\Sigma_2, Q_2, s_2, \delta_2, F_2)$ define $M = (Q, \Sigma, s_0, \delta, F)$ with

$$\begin{aligned} Q &= Q_1 \cup Q_2 \\ \Sigma &= \Sigma_1 \cup \Sigma_2 \\ \delta &= \delta_1 \cup \delta_2 \cup \left\{ (f) \xrightarrow{\varepsilon} (s_2) \;:\; f \in F_1 \right\} \\ s &= s_1 \\ F &= F_2 \end{aligned}$$

Lemma 3. $L(M_1) = L(M_2)$.

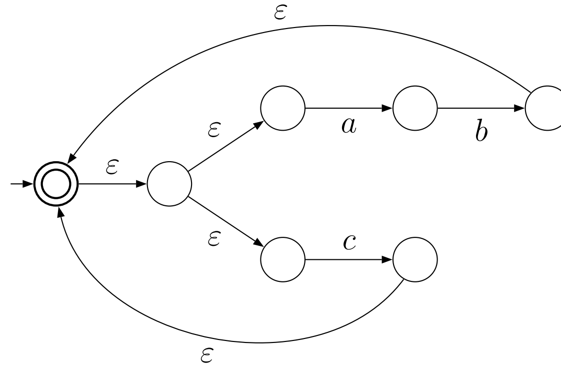
□

Kline star of Regular Languages

To allow as many consecutive words to be accepted at once[x] we construct paths from all final states to the initial state.

[x] Don't forget $\varepsilon \in \mathbb{L}^*$.

Example 71. A machine accepting $(\{ab\} \cup \{c\})^*$



Exercise 20. Why is the initial state accepting?

Theorem 13 (Kline)**.** The KLINE STAR of a regular language is a regular:

$$\exists M \,:\, \mathbb{L} = L(M) \implies \exists M' \,:\, \mathbb{L}^* = L(M')$$

Proof. Given $M = (Q, \Sigma, s_0, \delta, F)$ define $M' = (\Sigma', Q', s', \delta', F')$ with

$$\Sigma' = \Sigma$$
$$Q' = Q \cup \{s'\}$$
$$\delta' = \delta \cup \left\{ (s') \xrightarrow{\varepsilon} (s) \right\} \cup \left\{ (f) \xrightarrow{\varepsilon} (s') \;:\; f \in F \right\}$$
$$F' = \{s'\}$$

Lemma 4. $L(M) = L(M')$.

Exercise. □

§2.6 End of Chapter Exercises

Exercise 21. Work out everything in §2.2 taking $\vdash$ as a function. Hint: $\vdash (\vdash (\vdash (\cdots \vdash (qw))))$ can be written

$$\begin{aligned} S_0 &= qw \\ S_1 &= \vdash (S_0) \\ &\vdots \\ S_\ell &= \vdash (S_{\ell-1}). \end{aligned}$$

Thus taking $\circ$ to be function composition means $\vdash^*$ can be viewed as an iterative process modelling, for example:

$$\begin{aligned} \vdash^3 (qw_0w_1w_2) &= \vdash (\vdash (\vdash (qw_0w_1w_2))) \\ &= \vdash (\vdash (q_1w_1w_2)) \\ &= \vdash (q_2w_2) \\ &= \ q'. \end{aligned}$$

Exercise 22. Compete the proof of pumping lemma by proving Equation (2.3). Namely,

$$\forall k \in \mathbb{N};\ q_j v^k \vdash^* q_j \tag{2.3}$$

when v is given as in Theorem 6

Chapter 3

Other Regular Constructs

"Love has its reasons, whereof reason knows nothing."
– Blaise Pascal, *Mathematician*

Regular expressions, Kleene's theorem, FSM /regular expression conversion, state elimination techniques, linear grammar machines, regular grammars.

§3.1 Regular Expressions

A regular expression is a compact way of expressing a regular language. Regular expressions are aptly named as any regular language corresponds to a regular expression and vice-versa. The utility of these expressions are their '2D' or 'plaintext', representations which can be inputted via keyboard.

Example 72. The regular expression $a(a+b)^*b$ generates all words starting with a and ending with b:

$$L\left(a(a+b)^*b\right) = \{a\}\{a,b\}^*\{b\}$$

In Chapter **??** we proved any regular language is constructible from $\cup$, $\cdot$, and $*$. In the same way any regular *expression* is constructible from $+$[i], $\cdot$, and $*$.

Definition 63 ($\mathscr{R}_\Sigma$)**.** The set of regular expressions over Σ is denoted $\mathscr{R}_\Sigma$ and induced (i.e. generated recursively) by:

[i] $+$ because $\cup$ isn't on the keyboard.

1. $\varnothing$, $\varepsilon \in \mathscr{R}_\Sigma$, and $\Sigma \subset \mathscr{R}_\Sigma$ (i.e. $\forall a \in \Sigma; a \in \mathscr{R}_\Sigma$),
2. if E_1, $E_2 \in \mathscr{R}_\Sigma$ then
   (a) $(E_1 + E_2) \in \mathscr{R}_\Sigma$,
   (b) $(E_1 E_2) \in \mathscr{R}_\Sigma$,
   (c) $E_1^* \in \mathscr{R}_\Sigma$.

Definition 64 (Regular Expression)**.**

$$E_i \in \mathscr{R}_\Sigma \iff E_i \text{ is a regular expression.}$$

Definition 65 (Language of a regular expression)**.** $\forall E \in \mathscr{R}_\Sigma$ the LANGUAGE OF E, denoted $L(E)$ is given by

1. $L(\varnothing) = \varnothing$,
2. $L(\varepsilon) = \{\varepsilon\}$,
3. $\forall a \in \Sigma;\ L(a) = \{a\}$,
4. if E_1, $E_2 \in \mathscr{R}_\Sigma$ then
   (a) $L(E_1 + E_2) = L(E_1) \cup L(E_2)$,
   (b) $L(E_1 E_2) = L(E_1) L(E_2)$,
   (c) $L(E_1^*) = L(E_1)^*$

Example 73. Some regular expressions and the regular languages they generate.

E	$L(E)$	without $*$	δ
10^*	$\{1\}\{0\}^*$	$\{10^n : n \in \mathbb{N}\}$	Figure ??
$0+1$	$\{0\} \cup \{1\}$	$\{0, 1\}$	Figure ??
$(10)^*$	$[\{1\}\{0\}]^*$	$\{(10)^n : n \in \mathbb{N}\}$	Figure ??
$0\,(0+1)^*$	$\{0\}\{0,1\}^*$	—	Figure ??.
$\{0^* 1\}^*$	$[\{0\}^*\{1\}]^*$	—	Figure ??.

Notice the missing entries here. Sometimes it is not possible to give a truly explicit definition of a regular language. Regular expressions were created to rectify this gap.

Exercise 23. What is the regular expression giving the universal language over $\Sigma = \{w_0, \ldots, w_n\}$?

Theorem 14. The set of regular expressions is countably infinite. That is, there is a *injective* and *onto* mapping from $\mathbb{N}$ into $\mathscr{R}_\Sigma$.

Proof. Exercise. □

§Kleene's Theorem

A major result of formal language theory is Kleene's theorem which asserts arbitrary regular expressions over Σ are generated by finite state machines. The significant consequence of this theorem is that regular expressions are in fact 'regular' (in the sense they cannot generate irregular languages).

Theorem 15 (Kleene's Theorem)**.**

$$\forall E \in \mathscr{R}_\Sigma;\ \exists M \in \textsc{fsm} \,:\, L(E) = L(M).$$

Proof. An immediate consequence of (the yet to be introduced) Proposition 23 and 24. □

Converting Expressions into Machines

We need to demonstrate regular expressions have no more power than FSMs by proving

$$\{L(E) \,:\, E \in \mathscr{R}_\Sigma\} \subseteq \mathbb{L}_{\text{REG}}.$$

Our strategy is a familiar one[ii]: show all regular expressions correspond to a FSM and conversely that every finite state machine generates a regular expression.

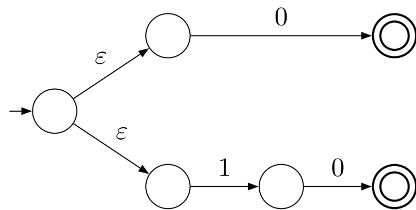
[ii] See proof of Theorem **??**

Towards this goal recall Definition 63 which shows how to build *any* regular expression. Let us demonstrate (and then formally establish) how a FSM can be built in parallel.

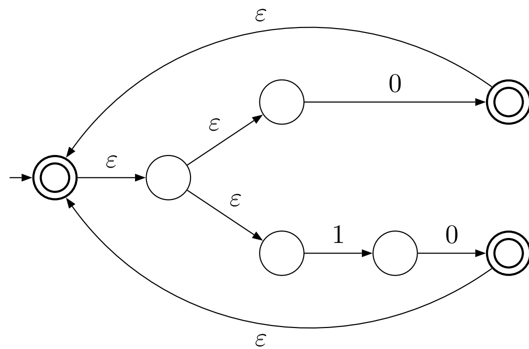
Example 74. Building an ε-NDFSM M such that

$$L(M) = L\left((0+10)^*1+00\right).$$

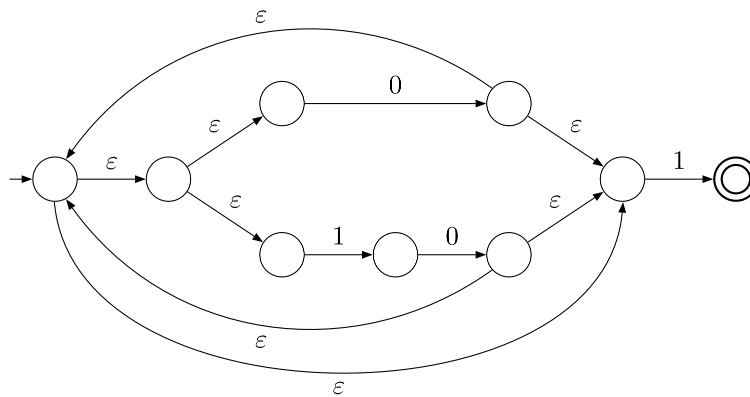
The machine is constructed using the strategies developed to prove the closure properties of regular languages.



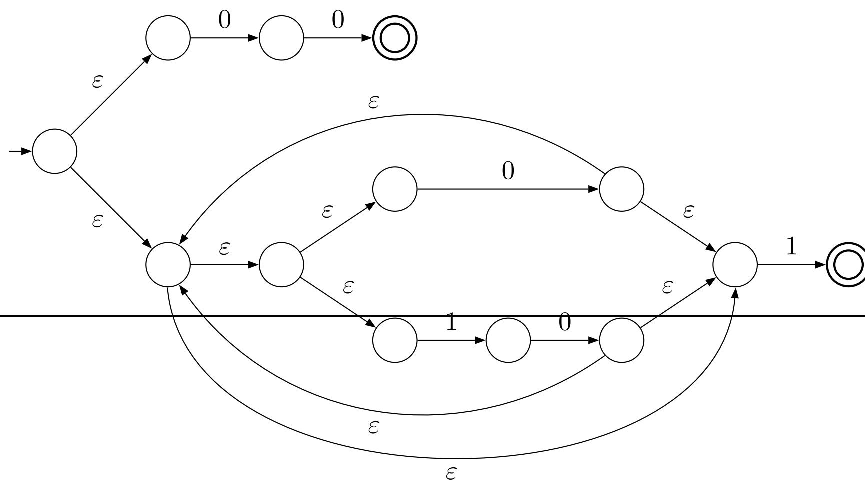
$(0+10)$



$(0+10)^*$



$(0+10)^*1$



$(0+10)^*1+00$

We formalize the construction of Example 74 in Proposition 23

Proposition 23 ($L(E) \subset L(M)$)**.**

$$\forall E \in \mathscr{R}_\Sigma;\ \exists M \in \text{FSM} \ :\ L(E) \subseteq L(M)$$

Induce over the number of 'operations' $(+, \cdot, *)$ in E (i.e. the number of recursive calls). For instance

$$\operatorname{Op}(a + ab^*) = \operatorname{Op}(a + a \cdot b^*) = 3.$$

Proof. Let $\operatorname{Op}(E) = |\{\, o \in E \ :\ o \in \{+, \cdot, *\}\,\}|$.

Take as an induction hypothesis

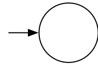
$$E \in \mathscr{R}_\Sigma \ :\ 0 \le \operatorname{Op}(E) < k \implies \exists M \in \text{FSM} \ :\ L(E) = L(M).$$

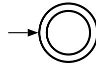
for arbitrary $n \in \mathbb{N}$.

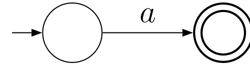
When $n = 0$ (the base case)

$$\operatorname{Op}(E) = 0 \implies E \in \{\varnothing, \varepsilon, a \in \Sigma\}.$$

In any of these selections for E a machine can be constructed as follows:







$M \ :\ L(M) = \varnothing$ $\qquad M \ :\ L(M) = \varepsilon$ $\qquad M \ :\ L(M) = \{a\}$

When $n > 0$ we are guaranteed $\operatorname{Op}(E) > 0 \implies \operatorname{Op}(E) \ge 1$ and consequently E contains a $+$, $*$, or $\cdot$.

Suppose $\operatorname{Op}(E) = n + 1$:

Case '$+$'. Here $+ \in E \implies E = E_1 + E_2$.

$$\begin{aligned} E = E_1 + E_2 &\implies \operatorname{Op}(E) = \operatorname{Op}(E_1 + E_2) \\ &\implies n + 1 = \operatorname{Op}(E_1) + \operatorname{Op}(+) + \operatorname{Op}(E_2) \\ &\implies n = \operatorname{Op}(E_1) + \operatorname{Op}(E_2) \end{aligned}$$

By IH

$$\exists M_1, M_2 \,:\, L(M_1) = L(E_1) \,\wedge\, L(M_2) = L(E_2),$$

and because regular languages are closed over union (Theorem **??**)

$$\exists M' \in \text{FSM} \,:\, L(M') = L(M_1) \cup L(M_2) = L(E_1) \cup L(E_2).$$

It follows

$$\exists M \in \text{FSM} \,:\, L(M') = L(E_1) \cup L(E_2) \overset{\text{def}}{=} L(E_1) + L(E_2)$$

Case '$\cdot$'. (Same argument as above.)

Case '$*$'. Here $* \in E \implies E = E_1^*$.

$$\begin{aligned} E = E_1^* &\implies \text{Op}(E) = \text{Op}(E_1^*) \\ &\implies n + 1 = \text{Op}(E_1) + \text{Op}(*) \\ &\implies n = \text{Op}(E_1) \end{aligned}$$

Thus, by the IH,

$$\exists M \in \text{FSM} \,:\, L(M) = L(E_1)$$

and consequently, because regular languages are closed over $*$ (Theorem **??**),

$$\exists M' \in \text{FSM} \,:\, L(M') = L(M)^* = L(E_1)^* \overset{\text{def}}{=} L(E_1^*).$$

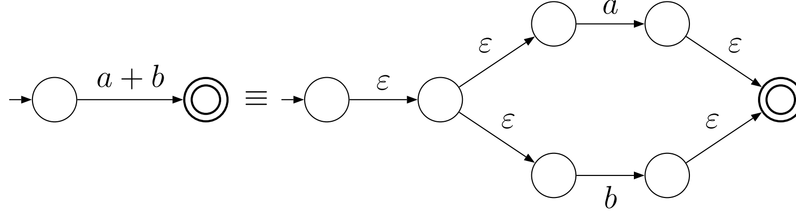
□

Converting Machines into Expressions

Converting regular expressions into finite state machines requires us to 'extend' our transition function to allow for moves on regular expressions.

Example 75. A machine with transitions from $\mathscr{R}_\Sigma$ and equivalent

ε-FSM.



The second machine is called an EXTEND FINITE STATE MACHINE.

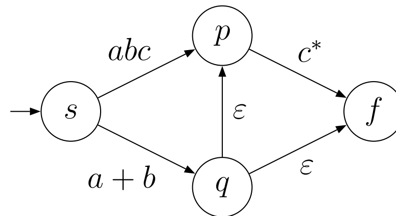
Definition 66 (EFSM)**.** An EXTENDED FINITE STATE MACHINE is a 5-tuple $(Q, \Sigma, s_0, \delta, f)$:

Q	set of states
Σ	input alphabet,
$s_0 \in \mathscr{P}(Q)$	initial state,
$\delta : Q \times \Sigma \to \mathscr{R}_\Sigma$	'extended'-state transition function, and
$f \in Q \setminus \{s_0\}$	*single* final states.

There are only two differences between an EFSM and a FSM:

1. there is only a *single* finite state distinct from s_0, and
2. $\delta(q_1, q_2)$ 'returns' a regular expression instead of a letter.

Example 76. Let δ be given by



then the values of δ are

$$\delta(s,p) = abc \qquad \delta(p,s) = \varnothing$$
$$\delta(s,q) = a+b \qquad \delta(q,s) = \varnothing$$
$$\delta(p,f) = c^* \qquad \delta(f,p) = \varnothing$$
$$\delta(q,p) = \varepsilon \qquad \delta(p,q) = \varnothing$$

$$\delta(q, f) = \varepsilon \qquad\qquad \delta(f, q) = \varnothing$$

(all remaining values—like $\delta(s, f)$—are all $\varnothing$ as well.)

Our goto function must be changed to handle moves on $\mathscr{R}_\Sigma$. In particular, we need

$$px \vdash q \iff \textcircled{p} \xrightarrow{E} \textcircled{q} \wedge x \in L(E)$$

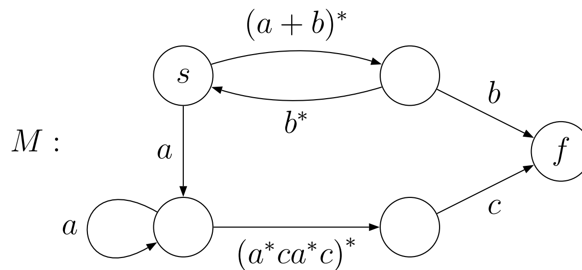
or, more generally,

$$px \vdash qy \iff \textcircled{p} \xrightarrow{E} \textcircled{q} \wedge \exists w \trianglelefteq x \,:\, x = wy \wedge w \in L(E)$$

(p goes to q on x when some prefix of x is in $L(E)$). As this is routine to establish once modifying δ we leave it to the reader to develop.

Although these new machines allow for a more compact drawing of state transition diagrams, what is drawn is not as 'user friendly' as a regular FSM.

Example 77. The language of the following machine is not immediately obvious[iii].



Some word of M are:

$$bbabab \in L(M) \qquad\qquad aabbc \in L(M)$$
$$accc \in L(M) \qquad\qquad aaaaac \in L(M)$$

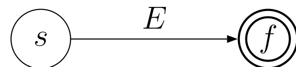
but the general form is far more elusive then it would be for a DFSM.

[iii] At least it is not obvious to the author.

Generally, it is easier to determine if a word is accepted by a DFSM than it is for a EFSM—DFSMs move on *letters* which are inherently more simple than regular expressions. It is thus beneficial to convert these EFSMs into DFSMs if only for the practical purpose of language detection. As we have already formalized a $\mathscr{R}_\Sigma$ to FSM conversion it only remains to show any EFSM can be converted into a regular expression.

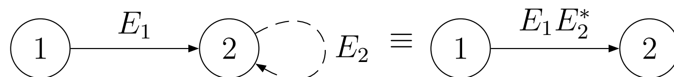
State Elimination Technique

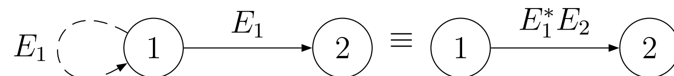
By removing states of a machine one by one, and combining the regular expressions of the edges, we reduce any EFSM to a machine with only a single transition:

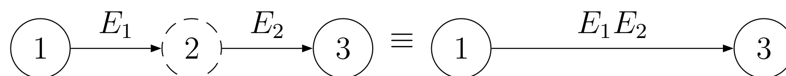


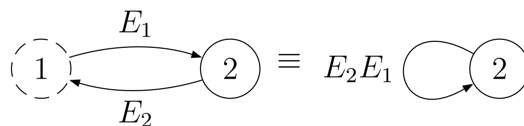
where E is a 'single' regular expression. (This is the reason EFSM are limited to a single final state distinct from the starting state.)

This process, called STATE ELIMINATION is motivated by the following observations. ($E_i \in \mathscr{R}_\Sigma$ and dashed states/edges are being removed.)



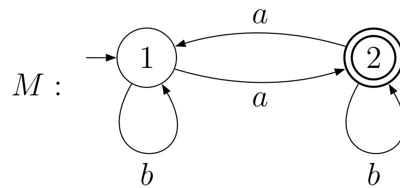




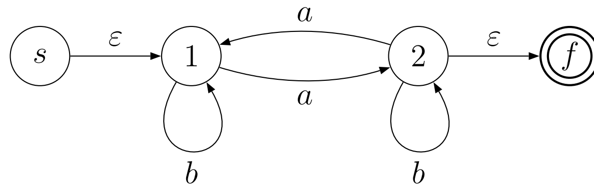


The process is fundamentally one of removing states and resolving edge routes.

Example 78. Consider the NDFSM given by

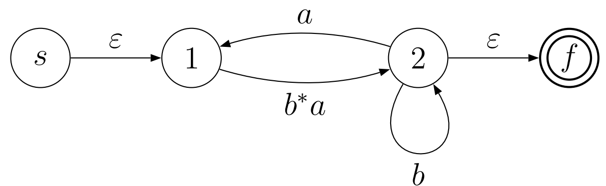


After (easily) converting M into a EFSM

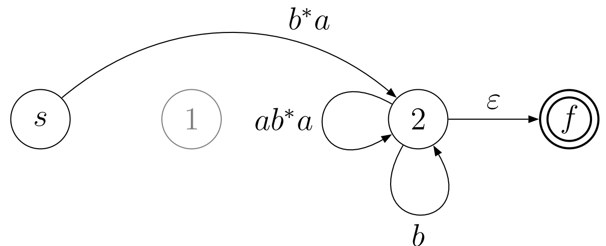


the state elimination can commence. (As a matter of convention we take $Q \subset \mathbb{N}$ and eliminate these states by the 'natural' order.)

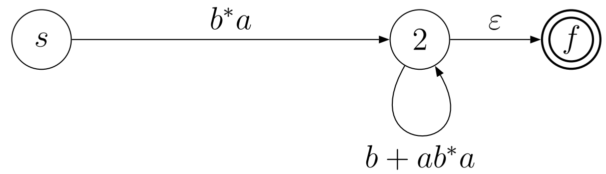
First, let us remove the loop on state 1,



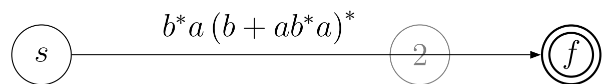
Now, states s and 2 can bypass state 1,



The two loops on state 2 are then combined,



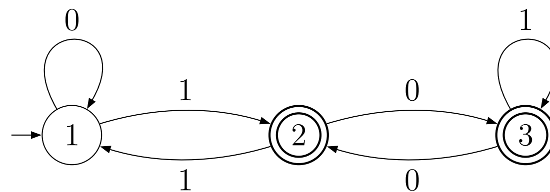
And, finally, removing state 2 is just a matter of accounting for the loss by introducing a Kleene star:



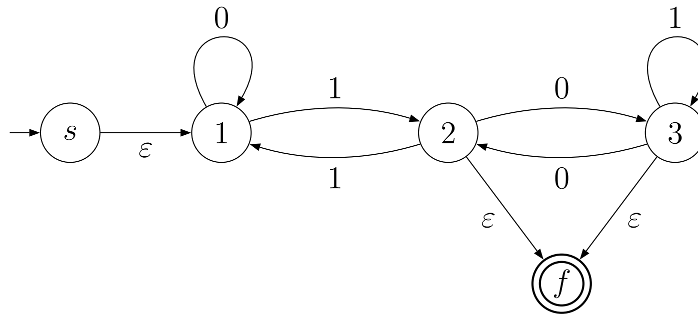
Thus $L(b^*a\,(b + ab^*a)^*) = L(M)$.

> The final regular expression depends on how the states are labelled and subsequently removed. Although these regular expressions are superficially different they are mathematically equivalent.

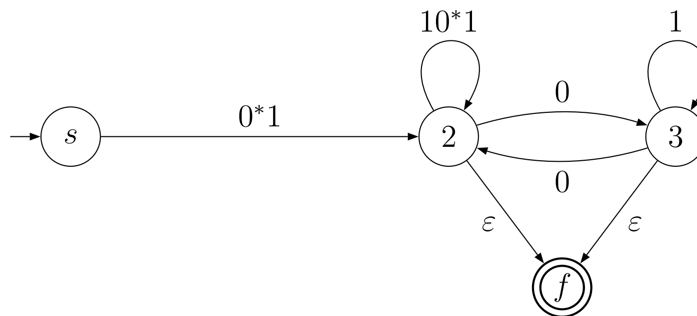
Example 79. Consider the (more complicated) NDFSM given by



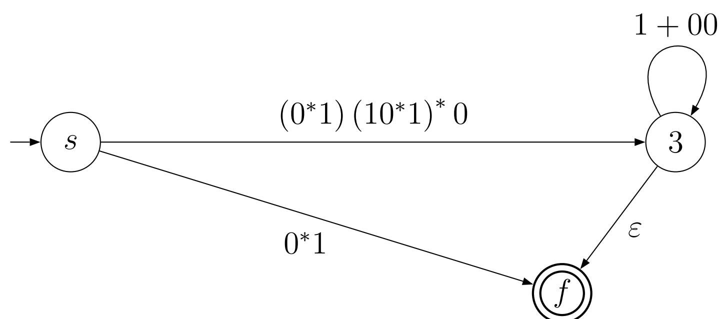
The corresponding EFSM is



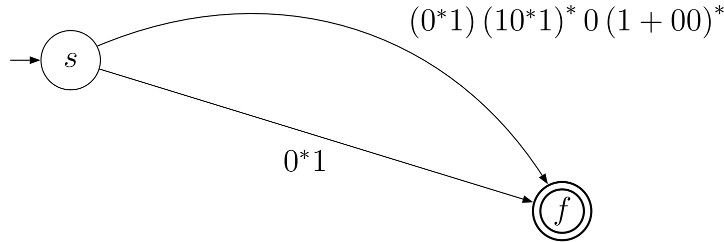
Proceeding with state elimination let us eliminate state 1,



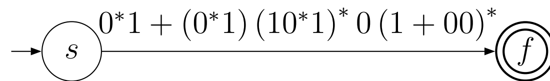
state 2,



and state 3.



Consolidating all edges into



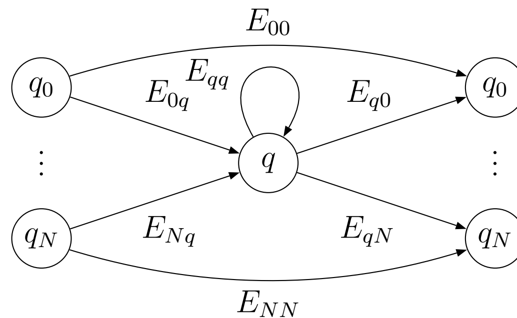
reveals the regular expression $0^*1 + (0^*1)\,(10^*1)^*\,0\,(1+00)^*$.

We conclude this section by formalizing the state elimination technique which simultaneously concludes the proof of Theorem 15 (Kleene's theorem).

Proposition 24 ($L(M) \subseteq L(E)$)**.**

$$\forall M \in \text{FSM};\ \exists E \in \mathscr{R}_\Sigma \,:\, L(E) \subseteq L(M).$$

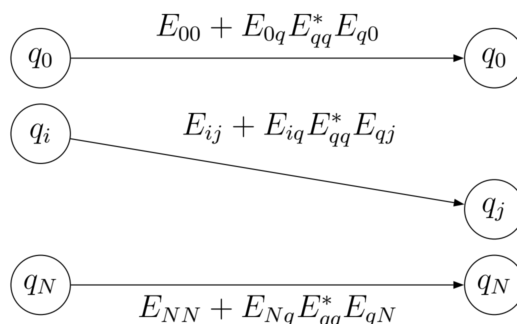
Sketch of proof. Let EFSM $M = (Q, \Sigma, s, \delta, f)$ be given and consider this drawing of state $q \in Q \setminus \{s, f\}$:



where the $E_{ii}, E_{qq} \in \mathscr{R}_\Sigma$ and $N = |Q|$.

The notation is chosen purposefully, E_{ij} is meant to denote the regular expression taking state q_i to state q_j. Also, there are *many* undrawn edges.

Removing q from this gives



from which the state transition function for a new machine without q is constructed.

Remember there is a direct route

$$\delta(q_i, q_j) = E_{ij}$$

between q_i and q_j; this is the reason for the '+'.

In particular $M' = \{Q \setminus \{q\}, \Sigma, s, \delta', f\}$ has

$$\delta'(q_i, q_j) = \delta(q_i, q_j) + \delta(q_i, q)(\delta(q, q))^* \delta(q, q_N).$$

(The routine presentation of showing $L(M') = L(M)$ is left as an exercise.)

Applying the above until only state s and f are left produces

$$\delta(s, f) = E$$

where E is a regular expression such that $L(E) = L(M)$. □

§3.2 Regular Grammars

The model of computation developed so far would be easy to simulate. A lamp, for instance, is a FSM and generally *any* system capable of processing input (e.g. turning a lamp off and on) is a FSM . Intuitively, we view FSMs as language *detectors*: machines which determine language membership in order to test equality among regular languages.

Let us now build a machine which *writes* or *produces* a language instead of reading it.

§Linear Grammar Machines

A Linear Grammar Machine prints letters—linearly—into slots:

H	e	l	l	o	W	o	r	l	d	¶

We use ¶ (end of line symbol) to indicated the number of slots available are infinite.

Notation (End of line)**.**

¶ = ☐ ☐ ☐ ⋯ .

What the LGM writes is governed by RE-WRITE or PRODUCTION rules. For instance, the production rule (or simply 'production')

$$P : S \to aS \mid b$$

expresses S can be *rewritten* as aS or b. Consequently, a machine with this production can write 'aab' as follows.

To start the machine we provide it with a *single* input: a REWRITE SYMBOL. (Unlike FSMs which we gave potentially infinite many inputs.)

Bold box indicates the location(s) of the read/write head.

S						¶

The LGM is now free to apply (any of) its production rules

S						¶	$\underline{S} \to aS$
a	S					¶	$S \to \underline{aS}$

(we combine the writing of 'aS' into one step,
even though it really requires two).

a	S					¶	$\underline{S} \to aS$
a	a	S				¶	$S \to \underline{aS}$
a	a	S				¶	$\underline{S} \to b$
a	a	b				¶	$S \to \underline{b}$

Since b does not correspond to a rewrite rule the machine terminates.

This entire process is more compactly written as

> Note, $\Longrightarrow$ (implies) is a longer arrow.

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aab$$

(much in the same way $\vdash$ indicates how we 'move' through a FSM).

Formal Definition of a LGM

So we may investigate the languages produced by LGMs let us formalize the ideas of the previous section.

Definition 67 (LGM)**.** A Linear Grammar Machine (LGM) is a 4-tuple (N, Σ, P, S)

N	nonterminals (rewrite symbols)
$\Sigma \,:\, \Sigma \cap N = \varnothing$	terminals (output alphabet)

$P \subseteq N \times \Sigma^* N$	set production rules
$S \in N$	single input

Notice the description of P in Definition 67 enforces that nonterminals are *always* written *last*.

Example 80. A LGM Γ producing $\{a^n b : n \in \mathbb{N}\}$:

$$\Gamma = (\{S\}, \{a, b\}, \{(S, aS), (S, b)\}, S),$$

is more typically written as

$$\Gamma = (\{S\}, \{a, b\}, P, S)$$
$$P : S \to aS \mid b.$$

Notation ($\to$)**.** When T, $T'_i \in N$ and $w_i \in \Sigma^*$

$$T \to w_0 T'_0 \mid \cdots \mid w_n T'_n$$

denotes $(T,\, w_0 T'_0), \ldots, (T,\, w_n T'_n)$. Thus,

$$T \to wT' \in P \iff (T, wT') \in P.$$

To model productions like $S \Rightarrow aS \Rightarrow aaS \Rightarrow aab$ we need to construct a function[iv] which, in this instance, takes 'aaS' and rewrites it as 'aaB'. Namely: $\Rightarrow (aaS) = aaB$.

Definition 68 ($\Rightarrow$)**.** A *function* called the PRODUCER which—when $T \to vT' \in P$ and $u, v \in \Sigma^*$—is given by:

$$\Rightarrow : \Sigma^* N \to \Sigma^* N$$
$$uT \mapsto uvT'.$$

[iv] Technically speaking this is a mapping, not a function.

Eventually produces ($\Rightarrow^*$)

Let us briefly confirm our definitions are sensible by checking $uT \Rightarrow^* vT'$ means (as we want)

$$uT \text{ eventually produces } vT' \,.$$

Taking $\Rightarrow$ as a relation, namely

$$vT \Rightarrow vT' \iff (uT, vT') \in \text{`} \Rightarrow_{\mathrm{R}} \text{'}$$

implies the meaning of $uT \Rightarrow^* u^{(n)}T^{(n)}$ [v] is that there is some transitive chain

$$uT \Rightarrow u'T' \Rightarrow \cdots \Rightarrow u^{(n)}T^{(n)}.$$

The meaning of $\Rightarrow^0$ is analogous to $\vdash^0$. We interpreted the later as not moving within a FSM. The interpretation of $\Rightarrow^0$ is *not producing.* Thus, every machine has the ability to write ε in this way.

> We need to be careful in proofs and use $\Rightarrow^+$ when $\Rightarrow^*$ is not appropriate.

It follows the transitive reflexive closure of the producer is consistent with what we are trying to express.

§The language of a LGM

Definition 69. The LANGUAGE OF $\Gamma = (N, \Sigma, P, S)$ is given by

$$L(\Gamma) = \{w \,:\, S \Rightarrow^* w\} \,.$$

(Any word that S produces.)

Definition 70 (Regular Grammar)**.**

$$\mathbb{L}_{\mathrm{LGM}} = \{L(\Gamma) \,:\, \Gamma \in \mathrm{LGM}\}$$

The natural question to raise here is:

---

[v] $u^{(1)} = u'$, $u^{(2)} = u''$, $\ldots$, $u^{(n)} = u'^{\cdots\prime}$

Can a LGM produce something a FSM cannot read?

Unfortunately, no—this is why members of $\mathbb{L}_{\text{LGM}}$ are called the *regular* grammars.

Theorem 16. LGMs can only produce regular languages.

$$\mathbb{L}_{\text{LGM}} = \mathbb{L}_{\text{REG}}.$$

Proof. Proof by construction. □

It is not difficult to construct a FSM M given a LGM Γ so that $L(M) = L(\Gamma)$.

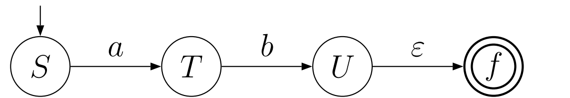
Example 81. Consider $\Gamma = ((S, T, U), \{a, b\}, P, S)$ with

$$S \to aT$$
$$T \to bU$$
$$U \to \varepsilon$$

A FSM accepting $L(\Gamma)$ is given by

$$M = (\{S, T, U, f\}, \{a, b\}, S, \delta, \{f\})$$

with



Notice M has $T \xrightarrow{x} T'$ whenever $T \to xT' \in P$ and the final state, f, is new.

Lemma 5 ($\mathbb{L}_{\text{LGM}} \subseteq \mathbb{L}_{\text{REG}}$)**.**

$$\forall \text{ LGM } \Gamma;\ \exists \text{FSM } M \ :\ L(\Gamma) = L(M)$$

Proof. Let LGM $\Gamma = (N, \Sigma, P, S)$ be given and define a FSM

$$M = (N, \Sigma, \delta, S, \{f\})$$

where δ is given by

$$\delta = \left\{ \textcircled{q} \xrightarrow{x} \textcircled{q'} \; : \; q \to xq' \in P \right\} \cup \left\{ \textcircled{q} \xrightarrow{x} \textcircled{f} \; : \; q \to x \in P \right\}$$

$$= \{qx \vdash q' \; : \; q \to xq' \in P\} \cup \{qx \vdash f \; : \; q \to x \in P\}$$

Proceeding with the PMI assume when $n \in \mathbb{N}^+$ that any word produced in n-steps by Γ is accepted by M:

$$S \Rightarrow^n w \implies w \in L(M) \tag{IH}$$

Clearly, when $n = 1$ (the base)

$$S \Rightarrow^1 w \implies S \to w \in P \overset{\text{ass}}{\implies} qw \vdash f \overset{\text{def}}{\implies} w \in L(M).$$

Supposing $S \Rightarrow^{n+1} w$ we deduce

$$S \Rightarrow^{n+1} w' \iff S \Rightarrow^n wT \Rightarrow^1 w'.$$

□

Lemma 6 ($\mathbb{L}_{\text{REG}} \subseteq \mathbb{L}_{\text{LGM}}$).

Chapter 4

Context Free Grammars

"I do face facts, they're lots easier to face than people."
– Meg Wallace, *A Wrinkle in Time*

Context free grammars.

§4.1 Introduction

The productions of a LGM $\Gamma = (N, \Sigma, P, S)$ are limited to

$$T \to w_0 \,:\, w_0 \in \Sigma$$
$$T \to wT' \,:\, w \in \Sigma^* \,\wedge\, T \in N$$

and subsequently LGMs can only produce regular languages (or equivalently, regular grammars).

Observe these productions can never introduce multiple nonterminals, that is, our producer can only write a symbol from N once every production. To finally 'escape' the regular languages we imbue our producers with the ability to write *multiple* nonterminals.

Example 82. Let $G = (\{S, T\}, \{a, b\}, P, S)$ have $P : S \to aSb \mid \varepsilon$ and consider the production

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow \cdots \Rightarrow a^n S b^n \Rightarrow^* a^n b^n \Rightarrow a^n b^n.$$

We can informally deduce $L(G) = \{a^n b^n \,:\, n \in \mathbb{N}\}$.

In order to *prove* G produces $\{a^n b^n \,:\, n \in \mathbb{N}\}$ it helps to develop some intuition about these machines first. Let us try to intuit what

some simple grammar machines produce by tracking *all* the productions.

> We will do left most derivations *only* although there is nothing preventing us from doing right derivations or even mixing them.

Example 83. Let $G = (N, \Sigma, P, S)$ with $P : S \to aSb \mid \varepsilon$.

$$
\begin{array}{ccccccccccc}
S & \Rightarrow & aSb & \Rightarrow & aaSbb & \Rightarrow & \cdots & \Rightarrow & a^nSb^n & \Rightarrow & \cdots \\
 & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\
 & & a\varepsilon b & & aa\varepsilon bb & & & & a^n\varepsilon b^n & & \\
 & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\
 & & ab & & aabb & & & & a^nb^n & &
\end{array}
$$

We informally deduce G produces $\left\{a^ba^n \,:\, n \in \mathbb{N}\right\}$.

Example 84. Let $G = (N, \Sigma, P, S)$ with

$$
\begin{aligned}
P : T &\to SS \\
S &\to aSb \mid \varepsilon
\end{aligned}
$$

T producse SS where S produces a^nb^n by Example 83. It follows G should generate $\{vw \,:\, v, w \in \{a^nb^n \,:\, n \in N\}\}$. Let us verify this guess:

$$
\begin{array}{ccccccccccccc}
T & \Rightarrow & SS & \Rightarrow & aSbS & \Rightarrow & aaSbbS & \Rightarrow & \cdots & \Rightarrow & a^mSb^mS & \Rightarrow & \cdots \\
 & & \Downarrow & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\
 & & \varepsilon S & & a\varepsilon bS & & aa\varepsilon bbS & & & & a^m\varepsilon b^mS & & \\
 & & \Downarrow_* & & \Downarrow_* & & \Downarrow_* & & & & \Downarrow_* & & \\
 & & a^nb^n & & aba^nb^n & & aabbS & & & & a^mb^ma^nb^n & &
\end{array}
$$

We informally deduce G produces

$$\{a^mb^ma^nb^n \,:\, n, m \in \mathbb{N}\} = \{vw \,:\, v, w \in \{a^nb^n \,:\, n \in N\}\}.$$

Having developed intuition about these grammar machines, let us commence with our formal derivation of what it means for a grammar to produce a language.

§4.2 Context Free Grammars

As a context free grammar machine has the same 'parts' as a linear grammar machine, we need only modify the production rules slightly. Namely, instead of taking $P \subseteq N \times \Sigma^* N$ allowing productions like $T \to wT'$ (nonterminals at the end *only*) we take $P \subset N \times (\Sigma + N)^*$ enabling, for instance, $T \to aTbT$.

Definition 71 (CFG)**.** A CONTEXT FREE GRAMMAR G is the 4-tuple (quadruple) $\{N, \Sigma, P, S\}$ with

N	nonterminals (rewrite symbols)
$\Sigma : \Sigma \cap N = \varnothing$	terminals (output alphabet)
$P \subseteq N \times (\Sigma + N)^*$	set production rules
$S \in N$	single input.

Definition 72 (write space)**.** The set

$$(\Sigma + N)^*$$

is called the WRITE SPACE (because it is what the CFG can 'write'). As convention we use α, β, ..., to represent members of $(\Sigma + N)^*$.

Our producer function requires 'upgrading' to account for the new production rules.

Definition 73 ($\Rightarrow$)**.** The PRODUCER is a mapping given by

> We use the set of production rules P as a function that accepts a nonterminal and returns $\alpha \in (\Sigma + N)^*$.

$$\Rightarrow : (\Sigma + N)^* \to (\Sigma + N)^*$$

$$wT\beta \mapsto wP(T)\beta = w\alpha\beta.$$

(Note $wT\beta$ where $w \in \Sigma^*$ means T is the leftmost nonterminal. In other words, we are rewriting the *first* nonterminal.)

As a function the producer satisfies

$$\Rightarrow_{\mathrm{F}} (wT\beta) = w\alpha\beta \iff wT\beta \Rightarrow w\alpha$$

which yields the relation

$$\begin{aligned} \Rightarrow_{\mathrm{R}} &= \{(\beta, \Rightarrow_{\mathrm{F}} (\beta)) \,:\, \beta \in (\Sigma + N)^*\} \\ &= \{(\beta, \beta') \,:\, \beta \Rightarrow \beta' \wedge \beta, \beta' \in (\Sigma + N)^*\}. \end{aligned}$$

This relation satisfies

$$(\beta, \beta'), (\beta', \beta'') \in \Rightarrow_{\mathrm{R}} \iff (\beta, \beta'') \in \Rightarrow_{\mathrm{R}}^2$$

which is consistent with our intended meaning of

$$\beta \Rightarrow \beta' \Rightarrow \beta'' \iff \beta \Rightarrow^2 \beta''$$

or more generally that

$$\beta \Rightarrow^* \beta'$$

means β *eventually produces* β'.

Definition 74 (language of a CFG)**.** The LANGUAGE OF A CFG $G = (N, \Sigma, P, S)$, denoted $L(G)$, is given by

$$L(G) = \{w \in \Sigma^* \,:\, S \Rightarrow^* w\}.$$

(The language of a CFG is any word that S can produce.)

Definition 75 (Context Free Languages)**.**

$$\mathbb{L}_{\mathrm{CF}} = \{\mathbb{L} \,:\, \exists G \in cfg; \mathbb{L} = L(G)\}.$$

Proposition 25. A CFG can produce an irregular language

$$\mathbb{L}_{\mathrm{CF}} \not\subseteq \mathbb{L}_{\mathrm{REG}}.$$

We just need to demonstrate

$$\exists \mathbb{L} \in \mathbb{L}_{\text{CF}} : \mathbb{L} \notin \mathbb{L}_{\text{REG}}$$

by *proving* $\{a^n b^n : n \in \mathbb{N}\} \in \mathbb{L}_{\text{CF}}$.

§4.3 CFG Language Proofs

We have shown *many* times $\{a^n b^n : n \in \mathbb{N}\} \notin \mathbb{L}_{\text{REG}}$, let us now prove this elusive language can be produced by a CFG.

Proposition 26. Let $G = (\{S\}, \{a, b\}, P, S) \in$ CFG with $P : S \to \varepsilon \mid aSb$; then

$$L(G) = \{a^n b^n : n \in \mathbb{N}\}.$$

Proof.

Lemma 7. $L(G) \subseteq \mathbb{L}$.

Take as an induction hypothesis

$$\forall k \leq n;\ S \Rightarrow^k w \in \Sigma^* \implies w \in \mathbb{L} \tag{IH}$$

which asserts anything produced by G in n or less steps is in the language.

Base. When $n = 1$ we have $S \Rightarrow^1 aSb \notin \Sigma^*$ (vacuous) and $S \Rightarrow^1 \varepsilon$ where $\varepsilon \in \Sigma^*$.

Induction.

$$S \Rightarrow^{n+1} w' \implies S \Rightarrow aSb \Rightarrow^n w' \overset{\text{IH}}{\implies} w' = a\left(a^k b^k\right) b \in \mathbb{L}$$

and thus $w \in L(G) \implies w \in \mathbb{L}$.

Lemma 8. $L(G) \supseteq \mathbb{L}$.

Take as an induction hypothesis

$$\forall k \leq n;\ S \Rightarrow^* a^k b^k \tag{IH}$$

which asserts G can only produce words of the form $a^k b^k$.

Base. $S \to \varepsilon \in P \implies S \Rightarrow^* \varepsilon = a^0 b^0$.

Induction. By IH $S \Rightarrow^* a^n b^n$

$$S \to aSb \in P \wedge S \Rightarrow^* a^n b^n \implies S \Rightarrow^* a(a^n b^n)b = a^{n+1} b^{n+1}.$$

Thus if $w \in \mathbb{L} \implies w = a^n b^n \wedge n \in \mathbb{N} \implies S \Rightarrow^* w \overset{\text{def.}}{\implies} w \in L(G)$

The result follows from the two lemmas. □

An immediate consequence of Proposition 26 is Proposition 25. However, we can show something stronger.

Theorem 17. $\mathbb{L}_{\text{REG}} \subset \mathbb{L}_{\text{CF}}$.

(We have already shown that every regular language can be produced by a LGM. As a LGM is just a 'weaker' CFG — but a context free grammar none-the-less — the result is immediate.)

Proof. By Theorem **??** $\mathbb{L}_{\text{REG}} \subseteq \mathbb{L}_{\text{CF}}$. Proposition 26 precludes the possibility of equality. □

We conclude this section with the less trivial language proof for Example 84.

Proposition 27. CFG $G = (\{S\}, \{a, b\}, P, S)$ with

$$P : S \to \varepsilon \mid aSb \mid bSa \mid SS$$

has $L(G) = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$.

Proof.

Lemma 9. $L(G) \subseteq \mathbb{L}$.

Take as induction hypothesis

$$\forall k \leq n; \left[S \Rightarrow^k w \wedge w \in \{a, b\}^*\right] \implies w \in \mathbb{L}. \tag{IH}$$

(The '$\wedge\ w \in \{a, b\}^*$' just means we ignore productions with nonterminals in them.)

Base. When $n = 1$, $S \to \varepsilon \in P \implies S \Rightarrow^1 \varepsilon$ and $\varepsilon \in \mathbb{L}$. All other productions, like $S \Rightarrow aSb$, are vacuous.

Induction. We need to show $S \Rightarrow^{n+1} w' \implies w\prime \in \mathbb{L}$, for this break into cases corresponding to the productions of P:

Case $S \to \varepsilon$. $S \to \varepsilon \implies S \Rightarrow \varepsilon$ where $\varepsilon \in \mathbb{L}$.

Case $S \to aSb$. $S \Rightarrow aSb \Rightarrow^n awb \stackrel{\text{IH}}{:} w \in \mathbb{L}$. Thus

$$|awb|_a = |a|_a + |w|_a + |b|_a = 1 + |w|_a + 0,$$
$$|awb|_b = |a|_b + |w|_b + |b|_b = 0 + |w|_b + 1, \text{ and}$$

$w \in \mathbb{L} \implies |w|_a = |w|_b \implies |awb|_a = |awb|_b \implies awb \in \mathbb{L}$.

Case $S \to bSa$. $S \Rightarrow bSa \Rightarrow^n bwa \stackrel{\text{IH}}{:} w \in \mathbb{L}$ and $bwa \in \mathbb{L}$ by similar argument[i].

Case $S \Rightarrow SS$. $S \Rightarrow SS \Rightarrow^n w_1w_2 \stackrel{\text{IH}}{:} w_1w_2 \in \mathbb{L}$. It is trivial to show $w_1, w_2 \in \mathbb{L} \implies w_1w_2 \in \mathbb{L}$.

Lemma 10. $L(G) \supseteq \mathbb{L}$.

Take as induction hypothesis

$$\forall k \leq n;\ |w| = 2k \implies S \Rightarrow^* w. \tag{IH}$$

Base. $n = 0 \implies |w| = 0 \implies w = \varepsilon$ and $S \to \varepsilon \in P \implies S \Rightarrow^* \varepsilon$.

Induction. $|w| = 2\,(n+1) = 2n + 2 \implies$

$$w \in \{aw'b,\ bw'a,\ aw'a,\ bw'b\}\,.$$

Case $w = aw'b$. $|aw'b| = 2n + 2 \implies |w'| = 2n \stackrel{\text{IH}}{\implies} S \Rightarrow^* aw'b$. and $S \Rightarrow aSb \Rightarrow^* aw'$.

Case $w = bw'a$. Same argument as above.

Case $w = aw'a$. It suffices to prove there are consecutive bs in w so that it can be produced via

$$S \Rightarrow SS \Rightarrow^2 aSbbSa \Rightarrow^* awa.$$

[i] Ensure when you use this that it is indeed appropriate.

Lemma 11. There are consecutive bs in $w = aw'a$,

$$aw_2 \cdots w_{2n-1}a \in \mathbb{L} \implies \exists i : w_i = w_{i+1} = b.$$

TAC suppose there are consecutive bs in w. As $|w| = 2n$ this means we are able to place n many bs among the $\frac{2n-2}{2}$ many letters ($w_2, \ldots, w_{2n-1}$) so that the bs alternate (which is the only way they can be nonconsecutive).

There are $\frac{2n-2}{2} = n - 1$ many nonconsecutive places (the even or odd positions) in which we place n many bs. ↯

The main result follows.

□

§4.4 CFG Simplificitation

There are bad production rules. For instance

$$S \to aS \mid a$$
$$T \to b$$

can never 'reach' the nonterminal T and

$$S \to S \mid SS$$

can never 'terminate' by writing a terminal. For this reason (and others) it is desirable to SIMPLIFY a set of production rules so that some of these redundancies can be eliminated.

§Terminating Symbols

Implicit to the statement

$$S \Rightarrow^* w$$

is that at some point the machine will print a word absent of rewrite symbols and stop. This is called TERMINATION and in general any $\alpha T \beta \in (\Sigma + N)^*$ which produces a word from Σ^* is called a TERMI-

NATING SYMBOL.

Definition 76 (terminating symbol)**.** $x \in \Sigma \cup N$ is a TERMINATING SYMBOL when

1. $x \in \Sigma$, or
2. $(x \to \alpha_0 \cdots \alpha_n) \in P \wedge \alpha_i$ is terminating.

Notice we say $\alpha T \beta \Rightarrow^* w \in \Sigma^*$ and not $S \Rightarrow^* \alpha T \beta \Rightarrow^* w \in \Sigma^*$ because we do not care if the symbol is 'reachable' (see Definition 77).

Notation. The set of TERMINATING SYMBOLS of a CFG G is denoted $\mathrm{TS}(G)$ and given by

$$\mathrm{TS}(G) = \{\alpha : \alpha \text{ is a terminating symbol}\}$$

The set of NONTERMINATING SYMBOLS is $\overline{\mathrm{TS}(G)}$.

> Production rules like
> $$\alpha T \beta \to \gamma$$
> define rules in CONTEXT SENSITIVE languages. So, for instance, you could have a rule which takes T to aTb only when T is adjacent to a
> $$aT \to aTB.$$
> As with english, words can mean entirely different things when taken out of context.

Example 85. Let $G = (\{S, T, U\}, \{a\}, P, S) \in$ CFG have

$$\begin{aligned} P : S &\to T \mid SS \\ T &\to a \mid UU \\ U &\to U \end{aligned}$$

The terminating symbols can be deduced by iterative process:

$$\Sigma = \{a\} \subseteq \{a, T\} \subseteq \{a, T, S\} = \mathrm{TS}(G)$$

and consequently $\overline{\mathrm{TS}(G)} = \{U\}$.

§Reachable Symbols

A reachable symbol is any symbol from $\Sigma \cup N$ the machine can write (even if the symbol is surrounded by other symbols).

Definition 77 (reachable symbol)**.** $x \in \Sigma \cup N$ is REACHABLE when

$$S \Rightarrow^* \alpha x \beta.$$

Consequently, $S \in N$ is *always* reachable. (Note, unlike terminating symbols, $a \in \Sigma$ is not necessarily reachable.)

Notation. The set of REACHABLE SYMBOLS of a CFG G is denoted $\mathrm{RS}(G)$ and given by

$$\mathrm{RS}(G) = \{\alpha : \alpha \text{ is a reachable symbol}\}.$$

The set of UNREACHABLE SYMBOLS is $\overline{\mathrm{RS}(G)}$.

Example 86. Let $G = (\{S, T, U\}, \{a\}, P, S) \in$ CFG have

$$\begin{aligned} P : S &\to a \mid S \mid T \\ T &\to b \\ U &\to UU \mid b \end{aligned}$$

The reachable symbols can be determined by iterative process:

$$\{S\} \subseteq \{a, S, T\} \subseteq \{a, S, T, b\} = \mathrm{RS}(G)$$

and consequently $\overline{\mathrm{RS}(G)} = \{U\}$.

§Empty productions

Finding reachable symbols can be obstructed by the presence of ε. For instance, consider a CFG with productions

$$\begin{aligned} P : S &\to ASB \mid BSA \mid SS \mid aS \mid \varepsilon \\ A &\to AB \mid B \end{aligned}$$

$$B \to BA$$

Seemingly the only terminating symbols are $\{a\}$ (remember $\varepsilon \notin \Sigma \cup N$). However,

$$S \Rightarrow aS \Rightarrow \varepsilon$$

so clearly S is terminating as well.

Definition 78 (Empty production)**.** An EMPTY PRODUCTION is any production with form $T \to \varepsilon$.

Empty productions can be dealt with by introducing new production rules which are the result of applying the empty production. For instance,

$$S \to ST \mid \varepsilon$$

can be re-written as

$$S \to ST \mid \varepsilon T \mid \varepsilon.$$

Doing this to the production set at the beginning of this section reveals the hidden terminating symbols

$$\begin{aligned} P : S &\to \varepsilon \\ S &\to ASB \mid A\varepsilon B \\ S &\to BSA \mid B\varepsilon A \\ S &\to SS \mid \varepsilon S \mid S\varepsilon \\ S &\to aS \mid a\varepsilon \end{aligned}$$

so $\mathrm{TS}(G) = \{a, S\}$.

More generally we want to eliminate *any* nonterminal which can produce ε.

Definition 79 (ε-nonterminal)**.** $T \in N$ is a ε-NONTERMINAL when

> Using $\Rightarrow^*$ here would mean *everything* is a ε-nonterminal.

$$T \Rightarrow^+ \varepsilon.$$

Notation. The set of ε-NONTERMINALS of a CFG G is denoted RS (G) and given by

$$\mathrm{N}_\varepsilon(G) = \{\alpha : \alpha \text{ is a } \varepsilon\text{-nonterminal}\}.$$

Definition 80 (ε-free CFG)**.** $G \in$ CFG is ε-free when

$$\mathrm{N}_\varepsilon(G) = \varnothing.$$

Example 87. Let $G = (\{S, A, B, C\}, \{a, b\}, P, S)$

$$\begin{aligned} P : S &\to aSas \mid SS \mid bA \\ A &\to BC \\ B &\to \varepsilon \\ C &\to BB \mid bb \mid aC \mid aCbA \end{aligned}$$

The ε-nonterminals can be determined by iterative process:

$$\{B\} \subseteq \{B, C\} \subseteq \{A, B, C\} = \mathrm{N}_\varepsilon(G).$$

Removing ε-productions will revoke the ability of the machine to 'write nothing'—this is not a big deal. To address this in the theory we weaken our notion of language equality.

Definition 81 (ε-equivalence)**.** We say two language $\mathbb{L}_1$ and $\mathbb{L}_2$ are ε-EQUIVALENT when

$$\mathbb{L}_1 \setminus \{\varepsilon\} = \mathbb{L}_2 \setminus \{\varepsilon\}.$$

Notation.

$$\mathbb{L}_1 \stackrel{\varepsilon}{=} \mathbb{L}_2 \iff \mathbb{L}_1 \setminus \{\varepsilon\} = \mathbb{L}_2 \setminus \{\varepsilon\}.$$

§REDUCTION

A reduced context free grammar has no unreachable or nonterminating symbols.

Definition 82 (Reduced)**.** A CFG G is REDUCED if

$$\overline{\mathrm{RS}\,(G)} \cup \overline{\mathrm{TS}\,(G)} = \varnothing.$$

The process of removing the unreachable and nonterminating symbols from a CFG G is called REDUCTION. A CFG which is reduced is called a REDUCED CONTEXT FREE GRAMMAR (RCFG).

Theorem 18.

$$\forall G \in \mathrm{G};\ \exists G' \in \text{CFG} \,:\, L(G) = L(G') \,\wedge\, G' \text{ reduced.}$$

Proof. Let $G = (N, \Sigma, P, S)$ be given and consider $G' = (N', \Sigma', P', S)$. Determine $\mathrm{RS}\,(G)$ and $\mathrm{TS}\,(G)$ by marking algorithm and let

$$\begin{aligned} N' &= \mathrm{RS}\,(G) \cap \mathrm{TS}\,(G) \cap N \\ \Sigma' &= \mathrm{RS}\,(G) \cap \Sigma \\ P' &= \left\{T \to \alpha \,:\, \alpha \in (N' + \Sigma')^* \,\wedge\, T \to \alpha \in P\right\} \\ S &= S \end{aligned}$$

$w_0 \in \Sigma \not\Longrightarrow w_0 \in \mathrm{RS}\,(G)$

Showing $L(G) \stackrel{\varepsilon}{=} L(G')$ is left as an exercise. ☐

Example 88. Reduce $G = (\{S, A, B, C\}, \{0, 1\}, P, S)$ where

$$\begin{aligned} P : S &\to ASB \mid BSA \mid SS \mid 0S \mid \varepsilon \\ A &\to AB \mid B \\ B &\to BA \mid A \\ C &\to 1 \end{aligned}$$

$$\mathrm{TS}\,(G) = \{0, 1, C, S\} \implies \overline{\mathrm{TS}\,(G)} = \{A, B\}\,.$$

Eliminate $\overline{\mathrm{TS}\,(G)} = \{A, B\}$ — that is, remove any productions involving A or B.

$$P : S \to SS \mid 0S \mid \varepsilon$$

$$C \to 1$$

$\mathrm{RS}(G) = \{S\} \implies \overline{\mathrm{RS}(G)} = \{C\}$.

Eliminate $\overline{\mathrm{RS}(G)} = \{C\}$

$$P : S \to SS \mid 0S \mid \varepsilon$$

The CFG is now reduced.

§ε-REMOVAL

The removal of $S \to \varepsilon$ in Example 88 is slightly more involved because $\mathrm{N}_\varepsilon(G) = \{S\}$ and *surely* we cannot remove S. Instead we apply $S \to \varepsilon$ to every production which yields:

$$P : S \to \varepsilon S \mid S\varepsilon \mid 0\varepsilon \mid SS \mid 0S$$

or equivalently $P : S \to S \mid 0 \mid SS \mid 0S$.

Note this removal means we can only ensure an ε-equivalent language is generated.

Example 89. Do an ε-removal on $G = (\{S, T, U, V\}, \{a, b\}, P, S)$ where

$$\begin{aligned} P : S &\to aSaS \mid SS \mid bT \\ T &\to UV \\ U &\to \varepsilon \\ V &\to UU \mid bb \mid aV \mid aVbT. \end{aligned}$$

First we reduce as this has the potential to simplify the production set (even though in this case it won't).

The terminating symbols are constructed by

$$\Sigma = \{a, b\} \subseteq \{a, b, U\} \{a, b, U, V\} \subseteq \{a, b, U, V, T, S\} = \mathrm{TS}(G).$$

As this means $\overline{\mathrm{TS}(G)} = \varnothing$ there are no nonterminating symbols to eliminate.

The reachable symbols are constructed by

$$\{S\} \subseteq \{a, b, S, T\} \{a, b, S, T, U, V\}$$

and again $\overline{\mathrm{RS}(G)} = \varnothing$ so there are no unreachable symbols to eliminate.

However, the set of ε-nonterminals is nonempty because because $U \Rightarrow^+ \varepsilon$, $V \Rightarrow UU \Rightarrow^2 \varepsilon$, and $T \Rightarrow UV \Rightarrow V \Rightarrow^3 \varepsilon$. Thus

$$\{U\} \subseteq \{U, V\} \subseteq \{U, V, T\} = \mathrm{N}_\varepsilon(G)$$

<u>Apply</u> $U \Rightarrow \varepsilon$, $V \Rightarrow^* \varepsilon$, and $T \Rightarrow^* \varepsilon$:

$$\begin{aligned} P : S &\to aSaS \mid SS \mid bT \\ T &\to \varepsilon V \mid U\varepsilon \mid \varepsilon \\ U &\to \varepsilon \\ V &\to \varepsilon U \mid U\varepsilon \mid \varepsilon\varepsilon \mid bb \mid aV \mid a\varepsilon \mid a\varepsilon bT \mid aVb\varepsilon \mid a\varepsilon b\varepsilon. \end{aligned}$$

<u>Reduce</u>

$$\begin{aligned} P : S &\to aSaS \mid SS \mid bT \\ T &\to V \mid U \mid \varepsilon \\ U &\to \varepsilon \\ V &\to U \mid \varepsilon \mid bb \mid aV \mid a \mid abT \mid aVb \mid ab. \end{aligned}$$

<u>Elimiate</u> the ε-productions $T \to \varepsilon$, $U \to \varepsilon$, and $V \to \varepsilon$.

$$\begin{aligned} P : S &\to aSaS \mid SS \mid bT \\ T &\to V \mid U \\ U &\to \varnothing \\ V &\to U \mid bb \mid aV \mid a \mid abT \mid aVb \mid ab. \end{aligned}$$

(Note $U \to \varnothing \implies U \in \overline{\mathrm{TS}(G)}$ and consequently will be removed during a subsequent reduction.)

Remark 3. A production rule set with

$$\begin{aligned} S &\to \alpha_0 \cdots \alpha_n \\ \alpha_0 &\to \varepsilon \\ &\vdots \\ \alpha_n &\to \varepsilon \end{aligned}$$

would require $|\mathscr{P}(\{\alpha_0, \cdots, \alpha_n\})| = 2^{n+1}$ new production rules during ε-removal.

§4.5 Chompsky Normal Form

The 'ultimate' form of simplification is called the Chompsky normal form.

Definition 83. $G \in$ CFG is in Chompsky Normal Form (CNF) provided its productions are limited to

> 'Limited to' is misleading as there will not be a CFG which cannot be expressed in this form.

$$\begin{aligned} T &\to a \;:\; a \in \Sigma, \\ T &\to UV \;:\; U, V \in N. \end{aligned}$$

(Note: $T \to aV$ where $aV \in \Sigma N$ is *not* allowed.)

There are only two ways $T \to \alpha$ with $|\alpha| \neq 2$:

Definition 84 (Unit)**.** If $T \to U \;:\; T, U \in N$ then U is called a unit.

Example 90. The CFG G with

$$\begin{aligned} P : S &\to aT \mid U \\ T &\to b \\ U &\to SS \mid TT \end{aligned}$$

has the unit production $T \to U$. It can be removed by simply replacing U in $T \to U$ with everything U can write:

$$\begin{aligned} P : S &\to aT \mid SS \mid TT \\ T &\to b \end{aligned}$$

(U was eliminated because it became unreachable).

Definition 85 (long production)**.** $T \to \alpha \in P$ is a LONG PRODUCTION when

$$T \to \alpha \,\wedge\, |\alpha| \geq 2.$$

Example 91. The CFG G with

$$P : S \to TUV$$

has the long production $S \to TUV$. This long production can be converted to

$$\begin{aligned} S &\to TU' \\ U' &\to UV. \end{aligned}$$

One can appreciate how this procedure can be iterated and made into an algorithm.

§Unit Production Removal

As we demonstrated in Example 90 it is straightforward to remove unit productions.

Theorem 19. $\forall$ RCFG G; $\exists$ CFG G' :

1. G' is in CNF, and
2. $L(G) \stackrel{\varepsilon}{=} L(G')$.

Proof.

Input: ε-free and RCFG $G = (N, \Sigma, P, S)$.
Output: CFG $G' = (N, \Sigma, P', S)$: P' has no unit productions.
$P' = \varnothing$;
for $T \to \alpha \in P$ **do**
  if $\alpha \in N$ **then**
    $P' \leftarrow P' \cup \{T \to \beta : \alpha \to \beta \in P\}$;
  else
    $P' \leftarrow P' \cup \{T \to \alpha\}$;
return (N, Σ, P', S);

Algorithm 5: Unit Production Removal

□

§Long Production Removal

To remove the long production $A \to \alpha_0 \cdots \alpha_n$ let $N \leftarrow N \cup \{\alpha_1', \ldots, \alpha_n'\}$ and replace this long production with

$$
\begin{aligned}
A &\to \alpha_0 \alpha_1' \\
\alpha' &\to \alpha_1 \alpha_2' \\
&\vdots \\
\alpha_{n-1}' &\to \alpha_{n-1} \alpha_n
\end{aligned}
$$

Example 92. The production $S \to TUVbWa$ can be written as

$$
\begin{aligned}
S &\to TU \\
U' &\to UV' \\
V' &\to Vb' \\
b' &\to bW' \\
W' &\to Wa
\end{aligned}
$$

Since this is very close to the CNF let us complete the transformation by applying a HACK which handles the disallowed $b' \to bW'$ and $W' \to$

Wa.

$$\begin{aligned} S &\to TU \\ U' &\to UV' \\ V' &\to Vb' \\ b' &\to \overline{b}W' \\ W' &\to W\overline{a} \\ \overline{b} &\to b \\ \overline{a} &\to a \end{aligned}$$

Theorem 20. For every RCFG with long productions, there is an ε-equivalent machine without long productions:$\forall G \in$ RCFG; $\exists G' \in$ CFG : $L(G) \overset{\varepsilon}{=} L(G') \wedge \max(|\alpha| : A \to \alpha \in P) \leq 2$.

Proof.

Input: ε-free and RCFG $G = (N, \Sigma, P, S)$.
Output: CFG $G' = (N, \Sigma, P', S)$: P' has no long productions.
if $P = \varnothing$ **then**
    return $(N, \Sigma, \varnothing, S)$;
remove some $T \to \alpha = \alpha_0 \cdots \alpha_N$ from P;
if $N < 2$ **then**
    $(N', \Sigma, P', S) \leftarrow$ **thisproc** (N, Σ, P, S);
    return $(N', \Sigma, P' \cup \{T \to \alpha\}, S)$;
else
    $(N', \Sigma, P', S) \leftarrow$
    thisproc $(N \cup \{\alpha'_1\}, \Sigma, P \cup \{\alpha'_1 \to \alpha_1 \cdots \alpha_N\}, S)$;
    return $(N', \Sigma, P' \cup \{T \to \alpha_0\alpha'_1\}, S)$;

Algorithm 6: Long Production Removal

□

§Converting to CNF

There is a recipe for converting to CNF.

1. Reduce, namely:
   - Remove $\mathrm{TS}(G)$
   - Remove $\mathrm{RS}(G)$
2. Remove ε-productions.
3. Reduce again (there *may* be nothing to reduce).
4. Remove unit-productions.
5. Remove long-productions.
6. Apply hack.

Example 93. Convert

$$\begin{aligned} S &\to aAB \mid aABD \\ A &\to aD \mid B \mid \varepsilon \\ D &\to AB \\ B &\to bA \mid \varepsilon \\ C &\to cC \\ E &\to cc \\ F &\to BBA \mid \varepsilon \end{aligned}$$

into an equivalent CFG in Chomsky normal form.

1. Reduction

1i. Reduction : remove Nonterminating $= \{C\}$

$$\begin{aligned} S &\to aAB \mid aABD \\ A &\to aD \mid B \mid \varepsilon \\ D &\to AB \\ B &\to bA \mid \varepsilon \\ \not{C} &\to cC \\ E &\to cc \end{aligned}$$

$$F \to BBA \mid \varepsilon$$

1ii. Reduction : remove Nonreachables = $\{E, F\}$

$$S \to aAB \mid aABD$$
$$A \to aD \mid B \mid \varepsilon$$
$$D \to AB$$
$$B \to bA \mid \varepsilon$$
$$\not E \to cc$$
$$\not F \to BBA \mid \varepsilon$$

2. Remove ε-productions

2i. Remove $B \to \varepsilon$ and reduce

$$S \to aAB \mid aA \mid aABD \mid aAD$$
$$A \to aD \mid B \mid \varepsilon \mid \not\varepsilon$$
$$D \to AB \mid A$$
$$B \to bA$$

2ii. Remove $A \to \varepsilon$

$$S \to aAB \mid aB \mid aA \mid a \mid aABD \mid aBD \mid aAD \mid aD$$
$$A \to aD \mid B$$
$$D \to AB \mid B \mid A \mid \varepsilon$$
$$B \to bA \mid b$$

2iii. Remove $D \to \varepsilon$ and reduce

$$S \to aAB \mid aB \mid aA \mid a \mid aABD \mid \cancel{aAB} \mid aBD \mid \cancel{aB} \mid aAD \mid \cancel{aA} \mid aD \mid \cancel{a}$$
$$A \to aD \mid a \mid B$$
$$D \to AB \mid B \mid A$$
$$B \to bA \mid b$$

3. Remove Units

Remove $A \to B$, $D \to B$

$$
\begin{aligned}
S &\to aAB \mid aB \mid aA \mid a \mid aABD \mid aBD \mid aAD \mid aD \\
A &\to aD \mid a \mid bA \mid b \\
D &\to AB \mid bA \mid b \mid A \\
B &\to bA \mid b
\end{aligned}
$$

Remove $D \to A$ and reduce

$$
\begin{aligned}
S &\to aAB \mid aB \mid aA \mid a \mid aABD \mid aBD \mid aAD \mid aD \\
A &\to aD \mid a \mid bA \mid b \\
D &\to AB \mid bA \mid b \mid aD \mid a \mid \cancel{bA} \mid \cancel{b} \\
B &\to bA \mid b
\end{aligned}
$$

3. Remove Long Productions

$$
\begin{aligned}
S &\to aT \mid aB \mid aA \mid a \mid aU \mid aV \mid aW \mid aD \\
T &\to AB \\
U &\to TD \\
V &\to BD \\
W &\to AD \\
A &\to aD \mid a \mid bA \mid b \\
D &\to AB \mid bA \mid b \mid aD \mid a \\
B &\to bA \mid b
\end{aligned}
$$

4. Apply Hack

$$
\begin{aligned}
S &\to \overline{a}T \mid \overline{a}B \mid \overline{a}A \mid a \mid \overline{a}U \mid \overline{a}V \mid \overline{a}W \mid \overline{a}D \\
T &\to AB \\
U &\to TD \\
V &\to BD
\end{aligned}
$$

$$W \to AD$$
$$A \to \overline{a}D \mid a \mid \overline{b}A \mid b$$
$$D \to AB \mid \overline{b}A \mid b \mid \overline{a}D \mid a$$
$$B \to \overline{b}A \mid b$$
$$\overline{a} \to a$$
$$\overline{b} \to b$$

Chapter 5

Pushdown Automata

"There is an art to flying, or rather a knack. Its knack lies in learning to throw yourself at the ground and miss. Clearly, it is this second part, the missing, that provides the difficulties."

– The Hitchhiker's Guide to the Galaxy

Stacks, parse tress, and context free pumping lemma.

§5.1 Preliminaries

A Push Down Automata is a fsm with access to "first in last out" memory.

§The Stack

A stack essentially models a pile (or stack) of books on a table:

The books cannot be accessed 'randomly'. That is, to ensure this pile of books does not topple over, we insist

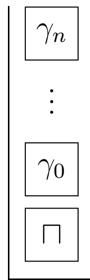
1. only the top book can be removed/accessed, and

2. new books are placed on top.

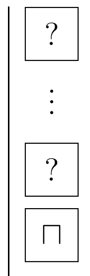
Although we have to remove *all the books* in order to access the bottom one, ultimately we can still access anything we want (albeit less efficiently).

Adding a book (*to* the top of the stack) is called PUSHING whereas removing a book (*from* the top of the stack) is called POPPING.

We illustrate a stack by



even though it is (perhaps) more appropriate to draw



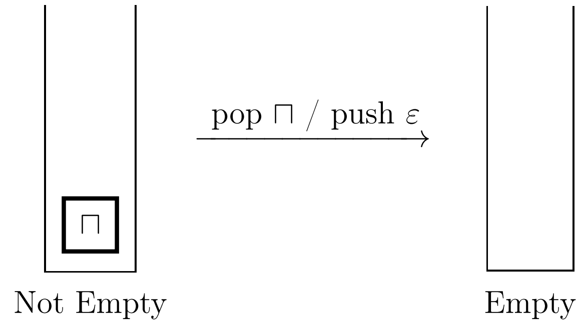
instead because elements of the stack are unknown until retrieved.

The purpose of the 'table' symbol '$\sqcap$' is to enable empty stack testing—popping the table means the stack is empty. As *there is no other way to detect the empty stack*, we should take care to put the table back if we pop it.[i]

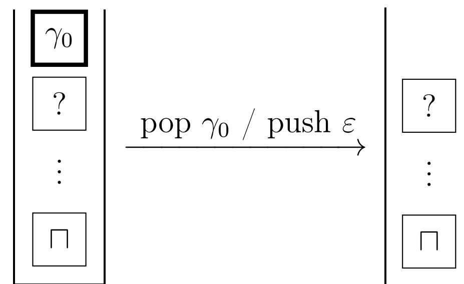
> One (perhaps silly) way of viewing $\sqcap$ is to imagine an empty stack containing 'lava' and the table being 'lava proof'. Elements not put on the table will be destroyed by lava.

[i] Otherwise we are reaching into lava.

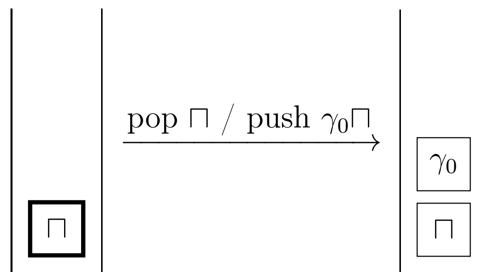
Example 94 (Emptying the stack)**.** A stack containing $\sqcap$ is *not* empty.



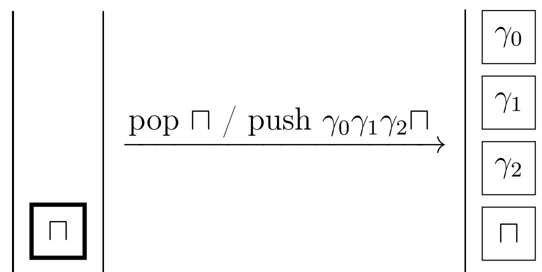
Example 95. Popping a symbol from the stack.



Example 96. Pushing a single stack symbol.



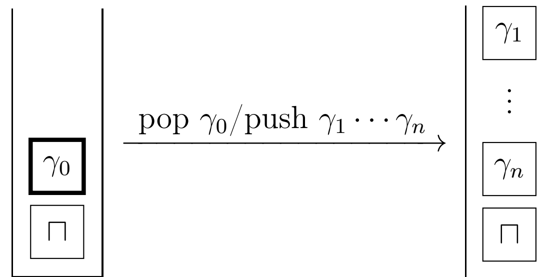
Example 97. Pushing multiple stack symbols.



Note $\sqcap$ is pushed *first*.

> In some sense, we really are 'pushing' $\gamma_0\gamma_1\gamma_2\sqcap$ into the stack by letting the symbols fall in from the back.

Example 98. Simultaneous pop and push.



Saying 'pop $\sqcap$' does *not* mean "remove the table from the top of the stack". Rather, it means, "*if* the table is removed from the top of the stack *then* push". Consequently, it is of no meaning to 'pop ε' as the stack at least has $\sqcap$.

§5.2 Push Down Automata

§Using A Stack

Our configurations sequences now need to encode the stack as well as the current state and input word. This new type of configuration is called an INSTANTANEOUS DESCRIPTION.

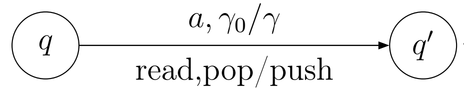
Definition 86 (ID)**.** An INSTANTANEOUS DESCRIPTION (ID) is an encoding of the current configuration of a PDA written

$$qw\gamma \in Q\Sigma^*\Gamma^*$$

where

1. $q \in Q$ (the current state),
2. $w \in \Sigma^*$ (the remainder of the word being processed), and
3. $\gamma \in \Gamma^*$ (the composition of the stack).

By strengthening the transition function δ to accommodate

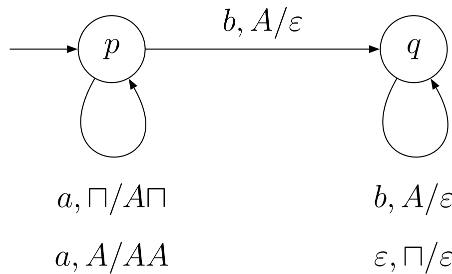


we will—unlike nondeterminism—extend our detection capabilities beyond the regular languages.

> Nondeterminism for FSMs is just a notational convenience allowing us to draw *exponentially* less states.

Example 99. A PDA for detecting $\{a^n b^n : n \in \mathbb{N}\} \notin \mathbb{L}_{\text{REG}}$ that works as follows:

1. put an A in the stack for every a, then
2. remove an A from the stack for every b, then
3. accept the word if (and only if) the stack is empty.



$a, \sqcap/A\sqcap$ $\quad$ $b, A/\varepsilon$

$a, A/AA$ $\quad$ $\varepsilon, \sqcap/\varepsilon$

Let us show (in Table 5.1) that $aabb$ is 'accepted by empty stack'. (As a matter of convention we assume the stack does not start empty, but rather contains $\sqcap$.)

§Formalizing PDAs

We need to extend the definition of a FSM to make the ID sequence

$$paabb\sqcap \vdash pabbA\sqcap \vdash pbbAA\sqcap \vdash qbA\sqcap \vdash q\sqcap \vdash q$$

Input	Pop	Transition	Push	ID Sequence
$aabb$	$\sqcap$	$\xrightarrow{a,\sqcap/A\sqcap}$	A, $\sqcap$	$paabb\sqcap \vdash pabbA\sqcap$
$\square abb$	A, $\sqcap$	$\xrightarrow{a,A/AA}$	A, A, $\sqcap$	$pabbA\sqcap \vdash pbbAA\sqcap$
$\square\square bb$	A, A, $\sqcap$	$\xrightarrow{b,A/\varepsilon}$	A, $\sqcap$	$pbbAA\sqcap \vdash qbA\sqcap$
$\square\square\square b$	A, $\sqcap$	$\xrightarrow{b,A/\varepsilon}$	$\sqcap$	$qbA\sqcap \vdash q\sqcap$
$\square\square\square\square$	$\sqcap$	$\xrightarrow{\varepsilon,\sqcap/\varepsilon}$		$q\sqcap \vdash q$

Table 5.1: Example 99. Accepting sequence for *aabb*.

meaningful (at least mathematically).

The new additions are simple to identify, we require:

1. a set of stack symbols, and
2. a transition function which, in addition to governing the state transitions, also handles stack management.

Definition 87 (PDA)**.** A PUSH DOWN AUTOMATA M is a sex-tuple $(Q, \Sigma, \Gamma, \delta, s_0)$

Σ	input alphabet,
$\Gamma : \sqcap \in \Gamma$	stack symbols containing $\sqcap$,
Q	finite set of states,
$s_0 \in Q$	initial state,
$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$	state transition function, and
$F \subseteq Q$	set of final states.

To see why this new definition of δ is reasonable, recall we would like δ to satisfy

$$q \xrightarrow{a, \gamma_0/\gamma} q' \iff \delta(q, a, \gamma_0) = (q', \gamma).$$

As δ is the mapping

$$\begin{aligned} \delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma &\to Q \times \Gamma^* \\ (q, a, \gamma_0) &\mapsto (q', \gamma), \end{aligned}$$

we see this is indeed the case.

Now, using this new δ, let us make the appropriate 'upgrades' to our goto function.

Definition 88 (goto)**.** The goto function maps IDs to IDs.

$$\vdash : Q \times \Sigma^* \times \Gamma^* \to Q \times \Sigma^* \times \Gamma^*$$

$$qw_0w\gamma_0\gamma \mapsto q'w\gamma'\gamma \,:\, \delta(qw_0\gamma_0) = q'\gamma'.$$

The routine demonstration of showing 'eventually goes to' means

$$qw\gamma \vdash^* q'w'\gamma' \iff \exists N \in \mathbb{N} \,:\, (qw\gamma, q'w'\gamma') \in \vdash^N$$

is left to the reader as an exercise.

§Acceptance by PDA

Asides from acceptance by empty stack, we also have the (usual) notion of acceptance by final state, as well as acceptance by *both* final and empty stack.

For the following denotations, let $M = (\Sigma, \Gamma, Q, q_0, \delta, F) \in$ PDA, $q \in Q$, $q_f \in F$, and $\gamma \in \Gamma^*$.

Notation. Acceptance by empty stack.

$$L_{\sqcup}(M) = \{w \,:\, q_0w\sqcap \vdash^* q\}$$

(q is an ID encoding an arbitrary state and empty stack.)

Notation. Acceptance by final state.

$$L_{\mathrm{F}}(M) = \{w \,:\, q_0w\sqcap \vdash^* q_f\gamma\}$$

($q_f\gamma$ is an ID encoding a final state and arbritrary stack.)

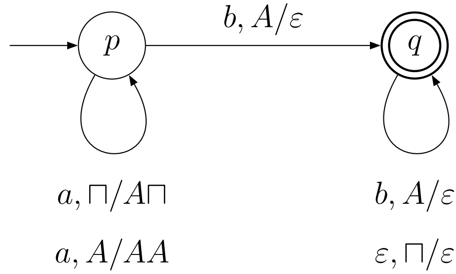
Notation. Acceptance by both final and empty stack.

$$L_{\sqcup+\mathrm{F}}(M) = \{w \,:\, q_0w\sqcap \vdash^* q_f\}$$

(q_f is an ID encoding a final state *and* empty stack.)

Example 100. Let $M = (\{a, b\}, \{A, B\}, \{p, q\}, p, \delta, \{q\}) \in$ PDA

where δ is given by



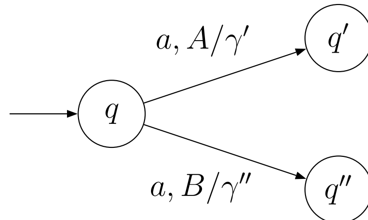
$$L_{\sqcup}(M) = \{a^n b^n \,:\, n \in \mathbb{N}\}$$
$$L_{\mathrm{F}}(M) = \left\{a^i b^j \,:\, i, j \in \mathbb{N} \wedge i > j\right\}$$
$$L_{\sqcup+\mathrm{F}}(M) = L_{\sqcup}(M) \cap L_{\mathrm{F}}(M) = \{a^n b^n \,:\, n \in \mathbb{N}\}$$

§5.3 Deterministic PDAs

Nondeterministic PDAs are more powerful than deterministic ones and thereby nondeterminsim for PDAs is a nontrivial issue.

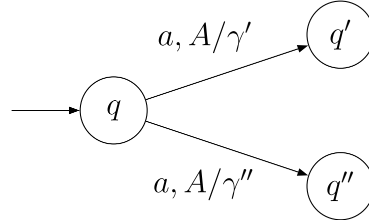
Let us first consider some examples.

Example 101. A deterministic PDA is given by



because, although q has *two* outgoing transitions for a, 'popping' eliminates the ambiguity over which state to move to.

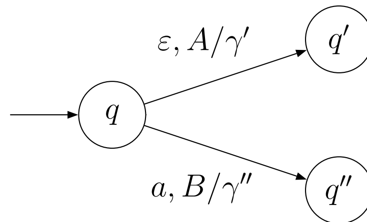
Example 102. A nondeterministic PDA is given by



because, contrary to Example 101, 'popping' does *not* eliminate the ambiguity—p goes to q' *and* q'' on a/A.

Additionally, the dual move conditions enable *determinisitc* ε-transitions.

Example 103. A PDA with deterministic ε-move.



Here, the ε move triggers *only* when A is popped. As no other outgoing transition relies on A, no ambiguity is introduced. (This *would* be nondeterministic if $A = B$. See Exercise **??**.)

> An additional consideration for nondeterministic PDAs is stack management. Since determinism is independent of what is being *pushed*, each time a nondeterministic move is encountered copies of the stack must be spawned. This is not an insignificant theoretical consideration and a certain obstacle for any practical implementation.

Definition 89. $M \in$ PDA is a DETERMINISTIC PUSH DOWN AUTOMATA when

1. $(q,\, w_0,\, \gamma_0,\, q',\, __)\,,(q,\, w_0,\, \gamma_0,\, q'',\, __) \in \delta \implies q' = q''$, and
2. $(q,\, \varepsilon,\, \gamma_0,\, __,\, __) \in \delta \implies (q,\, w_0,\, \gamma_0,\, __,\, __) \notin \delta$

(A '__' is used to indicate the value is irrelevant in addition to being arbitrary.)[ii]

[ii] This is actually valid syntax in Haskel and SML. Same semantics as well!

Example 104. The PDA of Example 102 violates Definition 89-1 because

$$(q,\, a,\, A,\, q',\, \gamma') \in \delta \,\wedge\, (q,\, a,\, A,\, q'',\, \gamma'') \in \delta \,\wedge\, q' \neq q''.$$

Example 105. The ε-move of Example 103 is deterministic because there is no other transition rule involving A. That is,

$$\neg\exists w_0 \in \Sigma \,:\, (q, w_0, A, __, __) \in \delta.$$

Notation. Denote the class of DETERMINISTIC CONTEXT FREE LANGUAGES by $\mathbb{L}_{\text{CF}}^{\text{D}}$:

$$\mathbb{L}_{\text{CF}}^{\text{D}} = \{L_F(M) \,:\, M \in \text{DPDA}\}\,.$$

Proposition 28. Deterministic PDAs are weaker than nondeterministic ones.

$$\mathbb{L}_{\text{CF}}^{\text{D}} \subset \mathbb{L}_{\text{CF}}.$$

Proof. □

§Closure Properties of DPDA

Deterministic PDAs have a more particular set of closure properties than generic (nondeterministic) ones. For instance, (only) deterministic PDA languages are closed over intersection with regular languages.

Proposition 29. $\mathbb{L}_{\text{CF}}^{\text{D}}$ is closed under intersection with regular languages

$$\mathbb{L}_1 \in \mathbb{L}_{\text{PDA}}^{\text{D}} \wedge \mathbb{L}_2 \in \mathbb{L}_{\text{REG}} \implies \mathbb{L}_1 \cap \mathbb{L}_2 \in \mathbb{L}_{\text{PDA}}^{\text{D}}.$$

(The idea here is to create a hybrid machine "$M_1 \times M_2$")

Proof. Suppose $\mathbb{L}_1 \in \mathbb{L}_{\text{CF}}^{\text{D}}$ and $\mathbb{L}_2 \in \mathbb{L}_{\text{REG}}$. By definition

$$\exists M_1 = (Q_1,\, \Sigma,\, \Gamma_1,\, s_1,\, \delta_1,\, F_1) \in \text{DPDA} \,:\, L_F(M_1) = \mathbb{L}_1, \text{ and}$$
$$\exists M_2 = (Q_2,\, \Sigma,\, s_2,\, \delta_2,\, F_2) \in \text{FSM} \,:\, L(M_2) = \mathbb{L}_2.$$

Consider $M \in$ PDA with

$$Q = Q_1 \times Q_2$$
$$\Gamma = \Gamma_1$$
$$s = (s_0,\ s_1)$$
$$F = F_1 \times F_2$$

and

$$\delta = \Big\{ (q,p) \xrightarrow{a,\beta/\gamma} (q',p') \ : \ q \xrightarrow{a,\beta/\gamma} q' \in \delta_1 \wedge p \xrightarrow{a} p' \in \delta_2 \Big\}.$$

Lemma 12. M is deterministic.

Proof of this. □

Finally, let $(f_0,\ f_1) \in F_1 \times F_2$ and $\gamma \in \Gamma^*$ be arbitrary. Deduce,

$$\begin{aligned} w \in L(M) &\iff (s_0,\ s_1)\, w\sqcap \vdash^* (f_0,\ f_1)\, \varepsilon\gamma \\ &\iff \underbrace{s_0 w\sqcap \vdash^* f_0\varepsilon\gamma}_{\text{in the PDA}} \wedge \underbrace{s_1 w \vdash^* f_2}_{\text{in the FSM}} \\ &\iff w \in L(M_1) \wedge w \in L(M_2) \\ &\iff w \in L(M_1) \cap L(M_2). \end{aligned}$$

□

Proposition 30. $\mathbb{L}^{\mathrm{D}}_{\mathrm{CF}}$ is closed under complementation.

$$\mathbb{L} \in \mathbb{L}^{\mathrm{D}}_{\mathrm{CF}} \implies \overline{\mathbb{L}} \in \mathbb{L}^{\mathrm{D}}_{\mathrm{CF}}.$$

Proof. □

§5.4 Context Free Pumping Lemma

In order to use counter-examples to prove (general) CFGs/PDAs are *not* closed under union and intersection, we need to develop a CF-PUMPING

LEMMA. Unfortunately, a SHORTCUT here is much more ellusive.

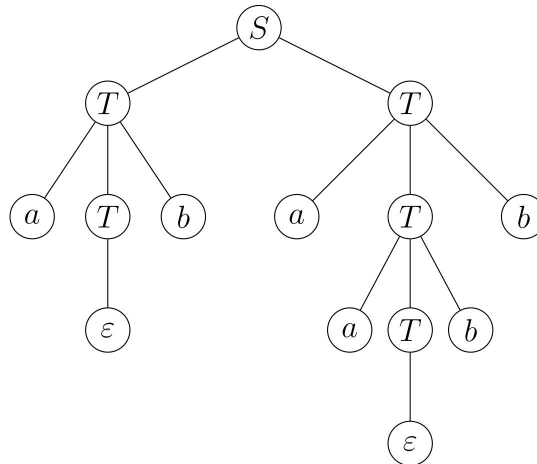
Recall (Example ??) where $M \in$ PDA had

$$P : S \to TT$$
$$T \to aTb \mid \varepsilon$$

producing $\mathbb{L} = \{a^n b^n a^m b^m : n, m \in \mathbb{N}\}$. The production

$$S \Rightarrow TT \Rightarrow aTbT \Rightarrow abT \Rightarrow abaTb \Rightarrow abaaTbb \Rightarrow abaabb$$

can be drawn as the graph



(because the graph's 'yield' is

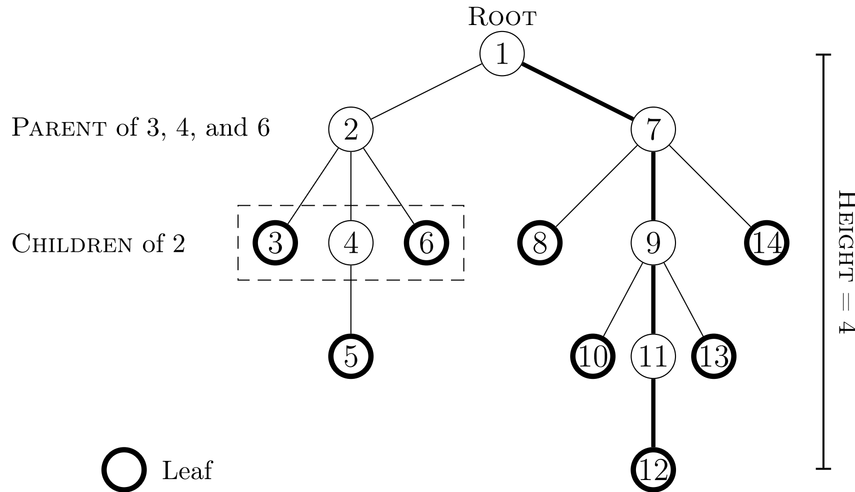
corresponding to $abaabb \in L(M)$.)

These type of loop-less graphs (more properly: undirected acyclic graphs) are called TREES. When a tree is used in this fashion to represent the construction of an expression, it is called a SYNTAX TREE.

As no cycles are allowed in trees it is overkill to have directed edges. (Any 'upward' pointing edge *automatically* introduces a cycle and thus all edges must be 'downward' pointing.) We will none-the-less refer to edges as INCOMING and OUTGOING despite drawing undirected edges.

Definition 90 (Tree)**.** An undirected acyclic graph is called a TREE.

Example 106. A tree and its various parts.



For our purposes, the crucial property of a tree is its YIELD.

Definition 91 (Yield)**.** The YIELD of a tree is the sequence of its leaf nodes ordered by DEPTH-FIRST-SEARCH.

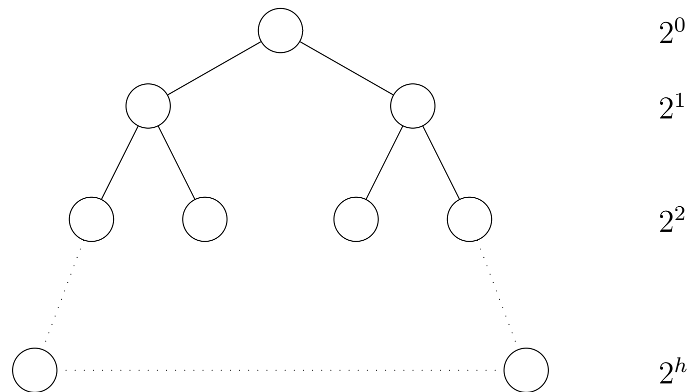
> In a *depth first* traversal the algorithm chooses the leftmost not-yet-visited child and backtracks when this node is a leaf or when there are no more nodes left to visit. The nodes of Example 106 are numbered in this manner.

§Maximum Yields

Our interest is the tree's canopy, that is, how 'wide' a tree can get at its leaves. This is more typically referred to as the MAXIMUM YIELD and it is fairly straightforward to give an *upper* bound for its size.

Example 107. A BINARY TREE (a tree where each node has two children or is a leaf) of arbitrary height h has 2^h many leaf nodes (i.e. a

yield of size 2^h).



These syntax trees will typically *not* consist of nodes with an equal number of children. To address this we use the *largest* number of children among *all* the nodes (a value called the FANOUT) which is an upper bound for the size of the yield.

Example 108. Unbalanced tree.

Definition 92 (Fanout).

Chapter 6

Turing Machines

> *"No, I'm not interested in developing a powerful brain. All I'm after is just a mediocre brain, something like the President of the American Telephone and Telegraph Company."*
>
> – Alan Turing

Turing machines, primitive and μ recursive functions, recursive and recursively enumerable languages, decidable and semidecideable problems, computability and uncomputability.

§6.1 Preliminaries

We have extended finite state machines with nondeterminism, ε-transitions, and stacks. We also constructed entirely different machinery (CFGs) which wrote languages instead of detecting them. Each attempt proved futile as

$$\mathbb{L} = \{a^n b^n c^n \, : \, n \in \mathbb{N}\}$$

was shown unreadable/unproducable in each case.

Our final extension: sequential memory, will be 'maximal'. That is, all attempts at extending from here will fail and thus we will have achieved the strongest (theoretical) model for computation.

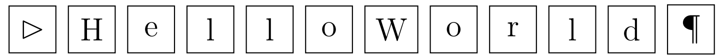
A Turing Machine (TM) is an automaton with read/write access to an infinitely long array called the TAPE.

Definition 93 (tape)**.** The TAPE is an infinite sequence of cells. The contents of the tapes first position is *always* '$\rhd$' (start of line).

Notation. Let ¶ denote infinitely many empty cells

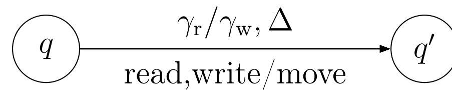
$$¶ = \square\,\square\,\square\,\square\ \cdots .$$

Example 109. A tape with 'HelloWorld' written on it. Note the infinitely long tailing sequence of '□'s (empty cells).



> Alan Turing. (Placeholder).

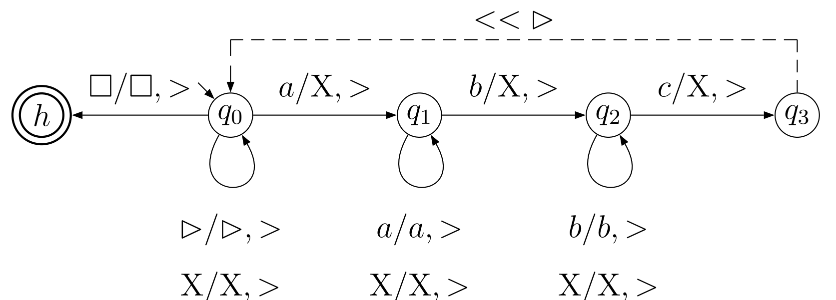
As with PDAs, we assume some interface with the tape exists and extend δ to include



where $\gamma_{\mathrm{r}}, \gamma_{\mathrm{w}} \in \Gamma$ are tape symbols and $\Delta \in \{<, >\}$ denotes one cell move left $(<)$ or one cell move right $(>)$.

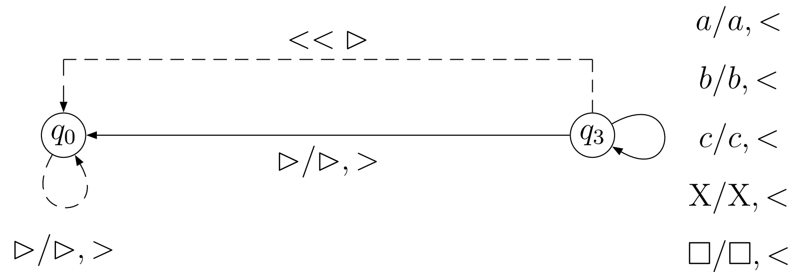
Example 110. A TM accepting $\mathbb{L} = \{a^n b^n c^n \ : \ n \in \mathbb{N}\}$.

The strategy here is to repeatedly cross out an a then b then c. If the process eventually produces a tape of all Xs (only possible for words of the form $a^n b^n c^n$) then we accept. Table 6.1 demonstrates this method for $aabbcc$.



The dashed edge is a compact way to invoke the frequently used instruction 'scan left for ▷'. The exact correspondence is given below,

where the dashed and solid edges may be freely interchanged,



.

§6.2 Formalizing Turing Machines

Definition 94. A Turing Machine (tm) is a sextuple $(Q, \Sigma, \Gamma, \delta, s, h)$ where

Q	set of states,
Σ	input alphabet,
$\Gamma : \{\triangleright, \Box\} \subseteq \Gamma$	tape symbols,
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{<, >\}$	transition function,
$s \in Q$	initial state,
$h \in Q$	final/halting state.

Investigating Definition **??** we see it is quite similar to that of a pda. The significant exception is the more robust δ which must also return the direction to shift. For instance, in Example **??** we implicitly applied

> Remember the state diagram is merely a notation for δ.

$$\delta(q_0, a) = (q_1, X, >)$$

when in state q_0 we read 'a', wrote 'X' and shifted right. This is perhaps

State	Tape	Read/Write	Move
q_0	▷ **[a]** a b b c c ¶	a/X	$>$
q_1	▷ X **[a]** b b c c ¶	a/a	$>$
q_1	▷ X a **[b]** b c c ¶	b/X	$>$
q_2	▷ X a X **[b]** c c ¶	b/b	$>$
q_3	▷ X a X b **[c]** c ¶	c/X	$<< \rhd$
q_0	**[▷]** X a X b X c ¶	$\rhd/\rhd$	$>$
q_0	▷ **[X]** a X b X c ¶	X/X	$>$
q_0	▷ X **[a]** X b X c ¶	a/X	$>$
q_1	▷ X X **[X]** b X c ¶	X/X	$>$
⋮	⋮	⋮	⋮
q_3	▷ X X X X X X **[¶]**	$\square/\square$	$<< \rhd$
q_0	**[▷]** X X X X X X ¶	$\rhd/\rhd$	$>$
⋮	⋮	⋮	⋮
q_0	▷ X X X X X X **[¶]**	$\square/\square$	$>$
h	Accepted.		

Table 6.1: Example 110. Accepting sequence for *aabbcc*. Note: bold boxes denote the *location of the read head*.

more succinctly conveyed by

$$q \xrightarrow{\gamma_r/\gamma_w,\,\Delta} q' \iff \delta(q,\gamma_r) = (q',\gamma_w,\Delta)\,.$$

§Configurations

A CONFIGURATION encodes the current state of the tape as well as the current state.

Example 111. The word

$$q \rhd a\underline{b}c$$

encodes a configuration of a TM in state q with $\rhd abc$ on its tape and read head at b.

Notation (Read Head)**.** Using _ to denote the position of the read head is notation writing tuples:

$$\rhd\gamma_0\cdots\underline{\gamma_i}\cdots\gamma_N = (\rhd\gamma_0\cdots\gamma_i,\ \gamma_{i+1}\cdots\gamma_N)\,.$$

Example 112.

$$\rhd a\underline{a}ab = (\rhd aa, ab)\,.$$

Definition 95. A CONFIGURATION of a TM is some element of

$$Q\Gamma^* \times \Gamma^*$$

but more precisely, $Q\left\{\rhd\left\{\Gamma\setminus\{\rhd\}\right\}^* \times \left\{\Gamma\setminus\{\rhd\}\right\}^* \left\{\Gamma\setminus\{\rhd,\square\}\right\}\right\}$.

For practical purposes we will say

$$q\rhd\underline{\gamma}\in\Gamma^* \iff \exists i\in\mathbb{N} : q\rhd\gamma_0\cdots\underline{\gamma_i}\cdots\gamma_N$$

and let $\underline{\Gamma}^* = \left\{\underline{\gamma} : \gamma\in\Gamma^*\right\}$.

§Moving The Read Head

It is convenient to view $<$ and $>$ as the functions

$$> \left(\rhd\gamma_0 \cdots \underline{\gamma_i} \cdots \gamma_n\right) = \rhd\gamma_0 \cdots \gamma_i\underline{\gamma_{i+1}} \cdots \gamma_n$$
$$< \left(\rhd\gamma_0 \cdots \underline{\gamma_i} \cdots \gamma_n\right) = \rhd\gamma_0 \cdots \underline{\gamma_{i-1}}\gamma_i \cdots \gamma_n$$

so $\delta\left(\rhd\underline{\gamma}\right) = \rhd\underline{\gamma'}$ expresses an unknown move on some arbitrary tape.

§GOTO

Our $\vdash$ function, which takes configurations to configurations, needs upgrading.

Definition 96 (goto)**.**

$$\vdash : Q\underline{\Gamma}^* \to Q\underline{\Gamma}^*$$
$$q \rhd \gamma_0 \cdots \underline{\gamma_i} \cdots \gamma_n \mapsto q' \Delta \left(\rhd\gamma_0 \cdots \underline{X} \cdots \gamma_n\right)$$

when $q \xrightarrow{\gamma_i/X,\Delta} q'$.

As a relation

$$\vdash = \left\{\left(q \rhd \gamma_0 \cdots \underline{\gamma_i} \cdots \gamma_n,\ q'\Delta\left(\rhd\gamma_0 \cdots \underline{X} \cdots \gamma_n\right)\right) \ : \ q \xrightarrow{\gamma_i/X,\Delta} q' \right\}.$$

For example,

$$q \rhd \underline{a}b \vdash^2 q''XX\underline{\square} \iff q \rhd \underline{a}b \vdash q' \rhd X\underline{b} \vdash q''.$$

§6.3 Recursive and Recursively Enumerable Languages

When a Turing machine ACCEPTS or REJECTS some word in both cases the machines terminates or HALTS. However, it is easy to build a machine which never halts.

Notation. Let Mw denote an INITIAL CONFIGURATION for $M = (Q, \Sigma, \Gamma, \delta, q_s, H) \in$ TM, namely

$$Mw = q_s\underline{w}\P.$$

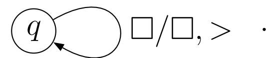
Notation. Let $M \in$ TM be given. Denote a

1. HALTING CONFIGURATION by

$$\mathscr{H} = q_f \rhd \underline{\gamma'},$$

2. ACCEPTING HALTING CONFIGURATION by $\mathscr{H}^\top$, and a
3. REJECTING HALTING CONFIGURATION by $\mathscr{H}^\perp$.

Example 113. A 'nonhalting' $M = (\{q\}, \varnothing, \{\Box, \rhd\}, \delta, q, \varnothing) \in$ TM which moves right an infinite number of times:



Although we can *intuit* $L(M) = \varnothing$, this is insufficient because our goal is to replace intuition with algorithms. More precisely, we are endeavouring to build machines which DECIDE language membership *for us* and, until now, termination has not been an issue: Finite state machines *always* halt because the necessarily finite input can only trigger finitely many moves; and, although context free grammars with NON-TERMINATING symbols exist, we developed an algorithm for detecting these loops in *finitely* many steps.

Soon we will prove an algorithm for detecting termination in Turing machines is impossible (See Section **??**) and therefore must accept that, instead of the two cases we desire (i.e. accept and reject) we have *three* cases when testing the language membership: The Turing machine

1. halts in an accepting state,
2. halts in a rejecting state, or

3. never halts.

Thus we (must) divide the languages Turing machines detect into two classes.

Definition 97 (Language of $M \in \text{TM}$)**.** The language generated by a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_s, H)$ is denoted $L(M)$ and given

$$L(M) = \left\{w \in \Sigma^* \,:\, Mw \vdash^* \mathscr{H}^\top\right\}.$$

Definition 98. $\mathbb{L} = L(M)$ is a DECIDABLE LANGUAGE when

1. $w \in \mathbb{L} \iff Mw \vdash^* \mathscr{H}^\top$, and
2. $w \notin \mathbb{L} \iff Mw \vdash^* \mathscr{H}^\perp$.

(M halts on all inputs.)

Definition 99. $\mathbb{L} = L(M)$ is a SEMIDECIDABLE LANGUAGE when

$$w \in \mathbb{L} \iff Mw \vdash^* \mathscr{H}^\top.$$

(M halts *exclusively* on accepting input.)

Definition 100. The class of RECURSIVE LANGUAGES is denoted $\mathbb{L}_{\text{REC}}$ and given by

$$\mathbb{L}_{\text{REC}} = \{L(M) \,:\, L(M) \text{ is decidabale}\}.$$

Definition 101. The class of RECURSIVELY ENUMERABLE LANGUAGES is denoted $\mathbb{L}_{\text{REC}}$ and given by

$$\mathbb{L}_{\text{REC}}^{\text{EN}} = \{L(M) \,:\, L(M) \text{ is semi-decidable}\}.$$

Exercise 24. Give an example of a recursive language.

§6.4 Computing with Turing Machines

Throughout this text we have been seemingly preoccupied with building machines for language detection/production. Although this is a

fascinating mental exercise the utility of constructing machines which move symbols about a tape is not immediately apparent. What we would *really* like to do is COMPUTE something, that is, to have a Turing machine take INPUT and write OUTPUT.

Let us start with an example of a computation and build a TM for addition over the BINARY DIGITS

$$\mathbb{Z}_2 = \{0, 1\}$$

or more generally over the BINARY NUMBERS

$$\begin{aligned} 0 + 1\,\{0,1\}^* &= \{0, 1, 10, 11, 100, \ldots\}_2 \\ &= \{0, 1, 2, 3, 4, \ldots\}_{10} \end{aligned}$$

Such a machine can be modelled on the simple 'carrying' technique taught to grade school students.

Example 114. $572 + 38 = 610$ calculated with carrying.

$$\begin{array}{rr} & \overset{1}{5}\overset{1}{7}2 \\ + & 38 \\ \hline & 610 \end{array}$$

where $\overset{1}{7} = 8$, $\overset{1}{5} = 6$, and so on.

The same can be done in base 2.

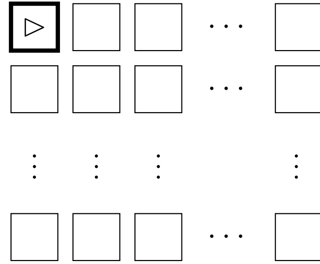
Example 115. $6 + 2 = 8$ calculated (in base 2) with carrying.

$$\begin{array}{rr} & \overset{1}{1}10 \\ + & 10 \\ \hline & 1000 \end{array}$$

where $1 + 1 = \overset{1}{0}$ and $\overset{1}{1} + 1 = 1$.

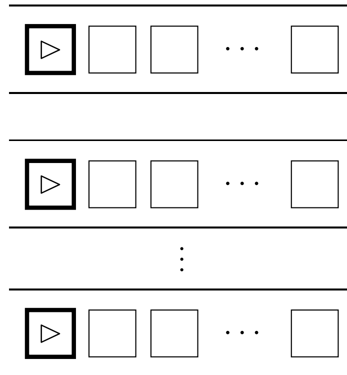
To implement this it would be reasonable to show single tape machines are equivalent to

2D-TAPES with a 'better' $\Delta = \{<, >, \wedge, \vee\}$ acting on



(a simple matter of index arithmetic, see Appendix **??**), or

MULTI-TAPE machines, with many read/write heads acting on



(also a simple matter, see Appendix **??**). However, neither are necessary here. We merely extend Γ to include the tape symbols

$$\left\{ \substack{0\\0}, \substack{1\\1}, \substack{1\\0}, \substack{0\\1}, \substack{1\\\square}, \substack{0\\\square}, \substack{\square\\\square} \right\},$$

which effectively *simulates* a multi-tape machine. (This is actually more-or-less how multi-tape machines are shown to be equivalent to single tape machines.)

Example 116. A TM for binary addition.

See Figure 6.1.

Let us verify this machine can compute $6 + 2 = 8$:

$$\overline{c} \rhd \underline{\substack{0\\0}}\, \substack{1\\1}\, \substack{1\\\square}\, \substack{\square\\\square} \vdash \overline{c} \rhd 0\underline{\substack{1\\1}}\, \substack{1\\\square}\, \substack{\square\\\square} \vdash c \rhd 00\underline{\substack{1\\\square}}\, \substack{\square\\\square} \vdash c \rhd 000\underline{\substack{\square\\\square}} \vdash \overline{c} \rhd 0001.$$

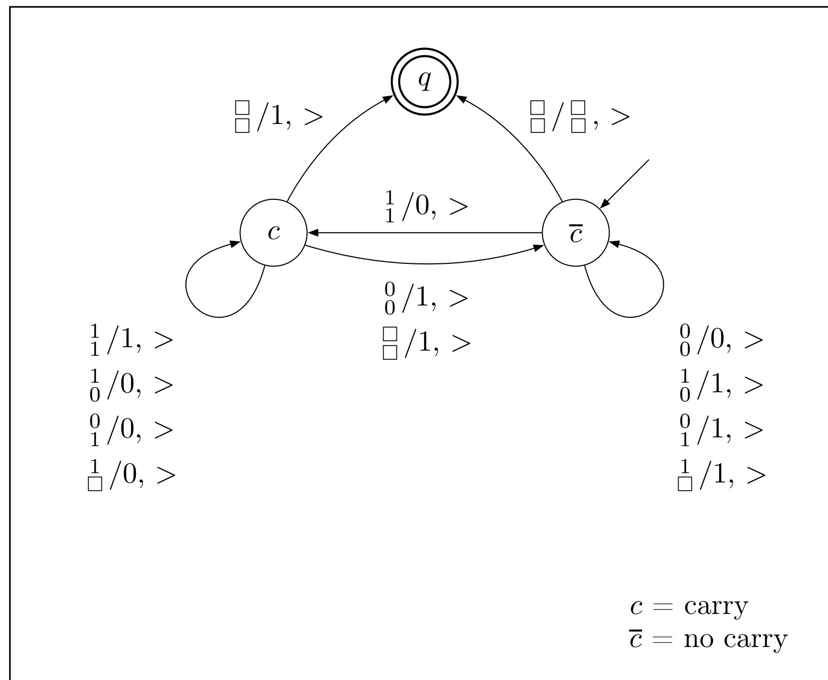


Figure 6.1: A TM for binary addition.

Exercise 25. Design a Turing machine in this way which can subtract numbers provided the difference is positive. That is, a TM for $x - y \, : \, x > y$.

Exercise 26. Design a Turing machine assuming the unary representation which can subtract numbers provided the difference is positive. That is, a TM for $x - y \, : \, x > y$.

§6.5 Recursive Functions

As computers were originally conceived to solve mathematical problems[i] we would like to show TMs can be used to solve numerical problems (thus justifying all our hard work). Thankfully, as much of mathematics can be constructed recursively we need only show how this recursion can be applied to Turing Machines as well.

[i] At least there is no known historical record showing Turing was motivated by a desire to enumerate all cat videos.

Definition 102. A RECURSIVE TURING MACHINE (RTM) halts on all input:

$$M \in \text{RTM} \iff \forall w \in \Sigma^*;\ Mw \vdash^* \mathscr{H}.$$

So we can invoke Turing machines in a more familiar fashion let us adopt the following notation.

Notation. For $M = (Q, \Sigma, \Gamma, \delta, q_s, H) \in \text{RTM}$ we say

$$M(w_0, \ldots, w_k) = w' \overset{\text{def}}{\iff} q_s \rhd \underline{\overset{w_k}{\underset{w_0}{\vdots}}} ¶ \vdash^* q_f \rhd \underline{w'}¶$$

where $q_f \in H$. (Here M simulates a multi-tape Turing machine with $k+1$ tapes for the $k+1$ inputs.)

Example 117. When M is given as in Example 116 (binary addition) we write $M(110, 10) = 1000$.

Definition 103. A function $g : \mathbb{N}^{k+1} \to \mathbb{N}$ is RECURSIVE when

$$\exists M \in \text{RTM} : g(n_0, \ldots, n_k) = M(n_0, \ldots, n_k)$$

$\forall (n_0, \ldots, n_k) \in \mathbb{N}^{k+1}$.

§Representing $\mathbb{N}$

There are a myriad of ways to represent the natural numbers. See, for instance, Table **??**. However, independent of representation, the natural numbers can be constructed recursively from 'zero' and a '+1' (successor) function. For this purpose it is best to take the unary representation because it is easy to build a TM taking

$$X \cdots X¶ \mapsto X \cdots XX¶.$$

(The machine merely scans for the first blank and writes X there.)

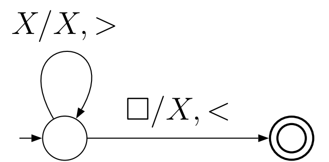
Proposition 31 (successor)**.**

$$\exists \texttt{succ} \in \text{RTM} : \forall n \in \mathbb{N}_X;\ \texttt{succ}\,(n) = n + 1.$$

Base 10	Binary	Unary	Peano
$\mathbb{N}_{10}$	$\mathbb{N}_2$	$\mathbb{N}_X$	$\mathbb{N}_\varnothing$
0	0	X	$\varnothing$
1	1	XX	$\varnothing \cup \{\varnothing\}$
2	10	XXX	$\varnothing \cup \{\varnothing \cup \{\varnothing\}\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	$X \cdots X$	n
$\{n+1\}_{10}$	$\{n+1\}_2$	$X \cdots XX$	$\varnothing \cup \{n\}$

Table 6.2: Different encodings of the natural numbers.

Proof.



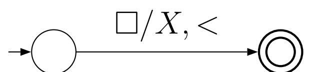
□

As well as a machine for producing 'zero' on a blank tape.

Proposition 32 (zero)**.**

$$\texttt{zero} \in \text{RTM} \ : \ \forall n \in \mathbb{N}_X;\ \texttt{zero}\,(n) = X.$$

Proof.



□

Definition 104 (The Natural Numbers)**.** The natural numbers are induced by

1. $0 \in \mathbb{N}$, and
2. $n \in \mathbb{N} \implies \texttt{succ}\,(n) \in \mathbb{N}$.

For the sake of clarity we adopt the notation

$$\begin{aligned} 1 &= \texttt{succ}(0) \\ 2 &= \texttt{succ}(1) = \texttt{succ}(\texttt{succ}(0)) \\ &\vdots \\ n &= \texttt{succ}(n-1) = \texttt{succ}(\texttt{succ}(\cdots \texttt{succ}(0))) . \end{aligned}$$

Proposition 33. The set of natural numbers is a recursive language.

$$\mathbb{N} \in \mathbb{L}_{\text{REC}}.$$

Sketch. There is a Turing machine for constructing any $n \in \mathbb{N}_{\text{X}} \approx \mathbb{N}$. Thus $\mathbb{N}$ is recursive by definition.

> $A \approx B$ means there is some 1-1 mapping from A to B. Effectively this means they can be taken to be mathematically equivalent.

□

§Primitive Recursive Functions

A Primitive Recursive Function is any function constructible (in a particular way) from other (primitive) recursive functions.

For example, addition can be constructed using the recursive functions `succ` and `zero`.

Example 118 (Addition)**.**

$$\texttt{add} \in \text{RTM} : \forall n, m \in \mathbb{N}_{\text{X}};\ \texttt{add}(n, m) = n + m.$$

$$\begin{aligned} \texttt{add}(n, 0) &= n \\ \texttt{add}(n, \texttt{succ}(m)) &= \texttt{succ}(\texttt{add}(n, m)) \end{aligned}$$

Which enables us to prove an (undeservedly) famous proposition.

Proposition 34 (One plus one equals two)**.**

$$\texttt{add}(1, 1) = 2.$$

Proof. By definition,

$$\begin{aligned}\texttt{add}(1,1) &= \texttt{add}(\texttt{succ}(0), \texttt{succ}(0)) \\ &= \texttt{succ}(\texttt{add}(\texttt{succ}(0), 0)) \\ &= \texttt{succ}(\texttt{succ}(0)) \\ &= 2.\end{aligned}$$

□

Something more difficult is $(a+b)+c = a+(b+c)$.

Proposition 35. $\forall \ell, n, m \in \mathbb{N}$,

$$\texttt{add}(\ell, \texttt{add}(m,n)) = \texttt{add}(\texttt{add}(\ell,m), n)$$

(i.e. addition is an associative operation.)

Proof. Requires *two* inductions (one embedded in the other). See Appendix **??**. □

In general, functions constructed in this way are called PRIMITIVE RECURSIVE FUNCTIONS and—importantly—each one corresponds to a recursive Turing machine.

Definition 105 (PRF)**.** $\forall k, \ell \in \mathbb{N}$ let $n, n_0, \ldots, n_\ell \in \mathbb{N}$ and

$$\begin{aligned} g &: \mathbb{N}^{k+1} \to \mathbb{N}, \\ h &: \mathbb{N}^{\ell+1} \to \mathbb{N}, \\ h_0 &: \mathbb{N}^{\ell+1} \to \mathbb{N}, \\ &\vdots \\ h_k &: \mathbb{N}^{\ell+1} \to \mathbb{N}. \end{aligned}$$

The PRIMITIVE RECURSIVE FUNCTIONS are constructed from

THE BASIC PRFS

1. $\texttt{zero}(n_0, \ldots, n_k) = 0$,
2. $\texttt{ident}_j(n_0, \ldots, n_k) = n_j$, and

3. $\texttt{succ}(n) = n + 1$.

COMPOSITION

$$g \odot \langle h_0, \ldots, h_k \rangle (n_0, \ldots, n_\ell) \overset{\text{def.}}{=} g(h_0(n_0, \ldots, n_\ell), \ldots, h_k(n_0, \ldots, n_\ell))$$

and

PRIMITIVE RECURSION

$$f(n_1, \ldots, n_k, 0) = g(n_1, \ldots, n_k),$$
$$f(n_0, \ldots, n_k, \texttt{succ}(m)) = h(n_1, \ldots, n_k, m, f(n_1, \ldots, n_k, m)).$$

Composition is typically used to 'shed' unwanted input. For example, suppose we wanted to define

$$A(x, y, z) = B(x, z)$$

this is really

$$A(x, y, z) = B(\texttt{ident}_1(x, y, z), \texttt{ident}_3(x, y, z))$$

or even more generally

$$A(x, y, z) = B \odot \langle \texttt{ident}_1, \texttt{ident}_3 \rangle.$$

Proving a function is a PRF requires we build a recursive description of that function using *only* the rules of Definition 105.

Proposition 36. Addition over $\mathbb{N}$ is primitive recursive.

Proof. We have

$$h = \texttt{succ} \odot \langle \texttt{ident}_3 \rangle \in \text{PRF}$$

and so addition can be given in the following way

$$\texttt{add}(n, 0) = \texttt{ident}_1(n, 0) = n$$

$$\begin{aligned}\texttt{add}\,(n, \texttt{succ}\,(m)) &= h\,(n, m, \texttt{add}\,(n, m)) \\ &= \texttt{succ}\,(\texttt{ident}_3\,(n, m, \texttt{add}\,(n, m))) \\ &= \texttt{succ}\,(\texttt{add}\,(n, m)) \in \text{PRF}\end{aligned}$$

□

Proposition 37. Multiplication over $\mathbb{N}$ is primitive recursive.

(Note: $n \times (m+1) = n + n \times m$.)

Proof. We have

$$h = \texttt{mul} \odot \langle \texttt{ident}_1, \texttt{ident}_3 \rangle \in \text{PRF}$$

and so multiplication can be given in the following way

$$\texttt{mul}\,(n, 0) = \texttt{zero}\,(n, 0) = 0$$

$$\begin{aligned}\texttt{mul}\,(n, \texttt{succ}\,(m)) &= h\,(n, m, \texttt{mul}\,(n, m)) \\ &= \texttt{add}\,(\texttt{ident}_1\,(n, m, \texttt{mul}\,(n, m))\,, \texttt{ident}_3\,(n, m, \texttt{mul}\,(n, m))) \\ &= \texttt{add}\,(n, \texttt{mul}\,(n, m)) \in \text{PRF}\end{aligned}$$

□

Proposition 38. Power (e.g. n^m) over $\mathbb{N}$ is primitive recursive.

(Note: $n^{m+1} = n \times n^m$.)

Proof. We have

$$h = \texttt{mul} \odot \langle \texttt{ident}_1, \texttt{ident}_3 \rangle \in \text{PRF}$$

and so power can be given in the following way

$$\texttt{pow}\,(n, 0) = \texttt{succ} \odot \langle \texttt{zero} \rangle = \texttt{succ}\,(\texttt{zero}\,(n, 0)) = 1$$

$$\begin{aligned}\texttt{pow}\,(n, \texttt{succ}\,(m)) &= h\,(n, m, \texttt{pow}\,(n, m)) \\ &= \texttt{mul}\,(\texttt{ident}_1\,(n, m, \texttt{pow}\,(n, m))\,, \texttt{ident}_3\,(n, m, \texttt{pow}\,(n, m))) \\ &= \texttt{mul}\,(n, \texttt{pow}\,(n, m)) \in \text{PRF}\end{aligned}$$

□

Proposition 39. 'Greater than' (e.g. $n > m$) over $\mathbb{N}$ is primitive recursive provided it is defined as

$$n > m \iff >(n, M) = \begin{cases} 1 & \text{if } n - m > 0 \\ 0 & \text{otherwise} \end{cases}$$

Proof. □

§6.6 μ-RECURSION

It is easy to prove addition over the positive fractions is primitive recursive because it only requires addition and multiplication:

$$\frac{a}{c} + \frac{b}{d} = \frac{ad + cb}{cd}.$$

However, insisting the answer is *reduced* is not only more difficult, but currently impossible.

Suppose we wanted to calculate

$$\frac{1}{2} + \frac{1}{4}$$

by doing

$$\frac{6 \div 2}{8 \div 2} = \frac{3}{4}.$$

How can we calculate $6 \div 3$?

To Evoke another strategy from primary school we calculate $6 \div 3$ by asking

> "How many times does 3 go into 6?"

or equivalently

> "What is the *least* ℓ such that $3 \cdot (\ell + 1) > 6$?"

The answer to both questions is, of course, two.

The above describes a method for calculating the so-called QUOTIENT; a precise description is given in Algorithm **??**.

```
Input:  m, n ∈ ℕ
Output: quo (m, n)
ℓ ← 0;
while ℓ · n < m do
  │ ℓ ← ℓ + 1;
return ℓ;
```

Algorithm 7: The Quotient Algorithm

Questions of this sort (ones involving while loops) can be solved using MINIMIZATION. The problem (as with while loops) is that termination is not guaranteed.

Definition 106. The MINIMIZATION of the function $g : \mathbb{N}^{k+1} \to \mathbb{N}$ is *any* function μ_g satisfying

$$\begin{aligned} \mu_g &: \mathbb{N}^k \to \mathbb{N} \\ (n_0, \dots, n_k) &\mapsto \min\left(\ell \,:\, g\left(n_0, \dots, n_k, \ell\right) = 1\right). \end{aligned}$$

Definition 107 (μRF)**.** The μ-RECURSIVE FUNCTIONS are given by

1. PRF $\subset \mu$RF,
2. $f \in \mu\text{RF} \implies \mu_f \in \mu\text{RF}$, and
3. closure over composition and primitive recursion.

(Equivalent to appending μ-recursion to Definition 105.)

Now that we have μ-recursion we can continue developing mathematics provided we are okay with algorithms which may never terminate. This may seem severe, but consider anyone who has written

$$1/3 = 0.333\cdots$$

has already made this concession.

Proposition 40. The quotient function, as described in Algorithm 7, is μ-recursive.

$$\text{quo}\,(m, n) \in \mu\text{RF}.$$

Proof. It is left to the reader to show

$$P(m, n, \ell) = \ell \cdot (n+1) > m \in \text{PRF}.$$

Taking this for granted we can write

$$\text{quo}(m, n) = \mu_P(m, n).$$

□

§6.7 Undecidable Problems

Notice Definition 106 does not actually define μ_g. This may seem strange but remember we are modelling computation in general and not particular algorithms (this is the concern of complexity theory). Our only assumption is any minimization algorithm requires a possibly unbounded number of steps.

This divides what Turing machines can solve into two categories

1. Decidable problems which require only for loops (bounded iteration), and

2. Semi-Decidable problems which require while loops (*possibly* unbounded iteration).

We can now pose the central question of this topic:

> Is there a problem which is *not* decideable?

or equivalently

> Is there a language which a Turing machine cannot detect?

§The Halting Problem

The difficulties with Turing machines could be resolved by detecting and removing infinite loops as we did with context free grammars. So, let us try to build a Turing machine `halts`(M, w) which can detect if another Turing machine M halts on input w.

> This is analogous to a complier preprocess which could detect if while loops terminate. A pipe dream.

Definition 108. Let $M = (Q, \Sigma, \Gamma, \delta, q_s, H)$ be given. Define `halts` to be the function

$$\texttt{halts}(M, w) = \begin{cases} \top & \text{if } Mw \vdash^* \mathscr{H} \\ \bot & \text{otherwise} \end{cases}.$$

Proposition 41 (The HALTING PROBLEM)**.**

$$\texttt{halts} \notin \text{RTM}.$$

> We are going to do something weird here and let $w = M$ in $M(w)$. Consider any program is merely a long string and thus can be used as input for another program.

Proof. TAC suppose `halts` $\in$ TM and let

$$\texttt{foo}(M) \text{ semi-decide } \neg\, \texttt{halts}(M, M).$$

That is, `foo`(M) halts only when `halts`$(M, M) = \bot$.

Consider[ii]

$$\texttt{halts}\,(\texttt{foo}, \texttt{foo}) \iff \texttt{foo}\,(\texttt{foo}) \text{ halts} \iff \neg\, \texttt{halts}\,(\texttt{foo}, \texttt{foo}) \lightning$$

□

§Reduction to the Halting Problem

Problems like the halting problem which are beyond the decision capabilities of Turing machines are called UNDECIDABLE.

Definition 109. A proposition P is UNDECIDABLE when

$$\neg \exists M \in \text{TM} : M \text{ semi-decides } P.$$

[ii] Do not let the compactness of this expression undermine its importance!

Every decidable turing machine is trivially semi-decidable. Thus if there is no semi-decidable machine for a problem there is also no decidable one.

We have already shown any TM which semidecides the halting problem derives contradiction. Thus the halting problem is undecidable. Here are some more undecidable problems:

The following problems Turing machines are undecidable.

1. Does M halt on w? (halting problem)

$$Mw \vdash^* \mathscr{H}$$

2. Does M halt on the empty tape?

$$M\varepsilon \vdash^* \mathscr{H}$$

3. Is there input for which M halts?

$$\exists w \in \Sigma^*;\ Mw \vdash^* \mathscr{H}$$

4. Does M halt on every input?

$$\forall w \in \Sigma^*;\ Mw \vdash^* \mathscr{H}$$

5. Do M_1 and M_2 halt on the same inputs?

$$\forall w \in \Sigma^*;\ M_1 \vdash^* \mathscr{H} \iff M_2 \vdash^* \mathscr{H}$$

Instead of deriving contradictions for each problem individually it is best to develop a general method of REDUCING one problem to another. In other words, a formal way of saying "if this problem is decidable then so is the halting problem".

The converse is *not* useful. Nothing is accomplished by showing a problem is decidable if the halting problem is decidable—because it isn't!

Notation. Let $M_1, M_2 \in \textsc{tm}$ and denote by and M_1M_2 the Turing machine given by

$$(M_1M_2)(w) = M_1(M_2(w)).$$

(This is the usual notion of function composition.)

Proposition 42.

$$M_1, M_2 \in \textsc{rtm} \implies M_1M_2 \in \textsc{rtm}.$$

Proof. Exercise. One would just need to formalize what we mean by

$$M_1(M_2(w)).$$

□

Definition 110 (Reducible)**.** M_1 is reducible to M_2 when

$$\exists \tau \in \textsc{rtm} : \forall w \in \Sigma^*;\ M_1(w) = (M_2\tau)w.$$

Notation.

$$M_1 \hookrightarrow_\tau M_2 \overset{\text{def.}}{\iff} M_1 \text{ reduces to } M_2$$

Theorem 21.

$$[M_1 \notin \textsc{rtm} \land M_1 \hookrightarrow_\tau M_2] \implies M_2 \notin \textsc{rtm}.$$

Proof. TAC let

$$[M_1 \notin \textsc{rtm}] \land [M_1 \hookrightarrow_\tau M_2] \land [M_2 \in \textsc{rtm}].$$

Because $M_1 \hookrightarrow_\tau M_2$ we have

$$[\forall w \in \Sigma^*;\ M_1(w) = M_2(\tau(w))] \iff M_1 = M_2\tau$$

However $M_2, \tau \in \textsc{rtm}$ means $M_2\tau \in \textsc{rtm}$ and thus $M_1 \in \textsc{rtm}$. $\lightning$. □

Proposition 43. There is no $P \in \text{RTM}$ such that

$$P(M) \iff M\varepsilon \vdash^* \mathscr{H}$$

Proof. Fix a $w \in \Sigma^*$ and TAC suppose there is such a P as given in the problem statement. Let $\tau(v) = M(w)$ (a function which maps any input to w then runs M); this function is trivially recursive.

$$P(\tau) \iff \tau\varepsilon \vdash^* \mathscr{H} \iff Mw \vdash^* \mathscr{H}.$$

Thus we have solved the halting problem. $\lightning$. $\square$

A Brief Reflection

Let us pause to appreciate what has been accomplished. The purpose of our endeavour was to create a theoretical model for computation powerful enough to do mathematics. Our early attempts for this failed because no FSM, NDFSM, εFSM, CFG, or PDA could read

$$\{a^n b^n c^n \,:\, n \in \mathbb{N}\}$$

and any reasonable computational model *should* be able to do this.

We finally arrived at Turing machines after recognizing the crucial feature that all our former machines lacked is the ability to read *and* write. However, a side effect of this feature—machines which never halt—is hugely problematic as algorithms (by definition) must terminate.

Accepting this side effect we investigated how much of mathematics could be modelled with Turing machines which are guaranteed to terminate (PRFs) and found these insufficient (e.g. we would like to do simple things like divide). Extending our Turing machines to allow 'while' loops (μ-recursion) provided us sufficient computational power to do division and from there we can expect to get the remaining numerical operations.

Thus we have accomplished our goal of deriving the standard model of computation...

However, there is a crucial deficiency of our computational model. If we say a 'computer' is some realization of a Turing machine (something with a processor and memory) then:

> There are logical statements in mathematics that cannot be decided by computers.

Not current computers, not future computers, not *any* computer *ever*.

Bibliography

[1] Kenneth Rosen. *Discrete Mathematics and Its Applications.* McGraw-Hill Science/Engineering/Math, 2011.