# SELF-REFERENCING, UNCOUNTABILITY, AND UNCOMPUTABILITY IN CPSC 421

JOEL FRIEDMAN

Contents





**Disclaimer:** The material may sketchy and/or contain errors, which I will elaborate upon and/or correct in class. For those not in CPSC 421/501: use this material at your own risk...

The reference [Sip] is to the course textbook, *Introduction to the Theory of Computation* by Michael Sipser, 3rd Edition. In CPSC 421 we assume you are familiar with the material in Chapter 0. In addition, we assume that you are you have seen some analysis of algorithms, including big-Oh and little-oh notation (e.g., $n \log_2 n + 3n + 5 = n \log_2 n + O(n)$).

In this article we introduce some fundamental concepts regarding self-referencing and uncomputability that we will cover in CPSC 421. This material is typical of the level of difficulty encountered in this course.

## 0. Decision Problems, Alphabets, Strings, and Languages

In this section we explain the connection between algorithms, decision problems, and some of the definitions in Chapter 0 of [Sip]. We also discuss *descriptions*, needed starting in Chapter 3 of [Sip].

Research supported in part by an NSERC grant.

0.1. **Decision Problems and Languages.** The term *decision problem* refers to the following type of problems:

(1) Given a natural number, $n \in \mathbb{N}$, give an algorithm to decide if $n$ is a prime.
(2) Given a natural number, $n \in \mathbb{N}$, give an algorithm to decide if $n$ is a perfect square.
(3) Given a natural number, $n \in \mathbb{N}$, give an algorithm to decide if $n$ can be written as the sum of two prime numbers.
(4) Given sequence of DNA bases, i.e., a string over the alphabet $\{C, G, A, T\}$, decide if it contains the string "ACT" as a substring.
(5) Given an ASCII string, i.e., a finite sequence of ASCII characters[1], decide if it contains the string "CPSC 421" as a substring.
(6) Given an ASCII string, decide if it contains the string "vacation" as a substring.
(7) Given an ASCII string, decide if it is a valid C program.

Roughly speaking, such problems take an *input* and say "yes" or "no"; the term *decision problem* suggests that you are looking for an *algorithm*[2] to correctly say "yes" or "no" in a finite amount of time.

To make the term *decision problem* precise, we use the following definitions.

(1) An *alphabet* is a finite set, and we refer to its elements as *symbols*.
(2) If $\mathcal{A}$ is an alphabet, a *string over* $\mathcal{A}$ is a finite sequence of elements of $\mathcal{A}$; we use $\mathcal{A}^*$ to denote the set of all finite strings over $\mathcal{A}$.
(3) If $\mathcal{A}$ is an alphabet, a *language over* $\mathcal{A}$ is a subset of $\mathcal{A}^*$.

(People often use *letter* instead of symbol, and *word* instead of string.) For example, with $\mathcal{D} = \{0, 1, \ldots, 9\}$, we use

$$\text{PRIMES} = \{s \in \mathcal{D}^* \mid s \text{ represents a prime number}\}$$

and

$$\text{SQUARES} = \{s \in \mathcal{D}^* \mid s \text{ represents a perfect square}\}$$

Here are examples of elements of PRIMES:

$$421, 3, 7, 31, 127, 8191, 131071, 524287, 2147483647$$

where we use the common shorthand for strings:

$$127 \text{ for } (1,2,7), \quad 131071 \text{ for } \quad (1,3,1,0,7,1), \quad \text{etc.}$$

So PRIMES is a language over the alphabet $\mathcal{D}$; when we say "the decision problem PRIMES" we refer to this language, but the connotation is that we are looking for some sort of algorithm to decide whether or not a number is prime. Here are some examples of strings over $\mathcal{D}$ that are not elements of the set PRIMES:

$$221, 320, 420, 2019.$$

[1] ASCII this is an alphabet of 256 letters that includes letters, digits, and common punctuation.

[2] The term *algorithm* means different things depending on the context; in CPSC 421 we will study examples of this (e.g., a DFA, NFA, deterministic Turing machine, a deterministic Turing machine with an orale $A$, etc.

0.2. **Descriptions of Natural Numbers.** From our discussion of PRIMES above, it is **not clear** if we consider 0127 to be element of PRIMES; we need to make this **more precise**. It is reasonable to interpret 0127 as the integer 127 and to specify that $0127 \in \text{PRIMES}$. However, in [Sip] we will be careful to distinguish a natural number $n \in \mathbb{N}$ and

$$\langle n \rangle \quad \text{meaning the "description" of } n,$$

i.e., the string that represents $n$ (uniquely, according to some specified convention), so the natural number 127 has a unique description as the string $(1,2,7)$, and the string $(0,1,2,7)$ is not the description of 127. With this convention, $0127 \notin \text{PRIMES}$; this is also reasonable.

[Later in the course we will speak of "the description of a graph" (when studying graph algorithms), "the description of a Boolean formula" (when studying SAT, 3SAT), "the description of a Turing machine," etc. In these situtations it will be clear why the input to an algorithm should be a description of something (as a string over some fixed alphabet) rather than the thing itself.]

If $n = \mathbb{Z}$ with $n = 127$, the symbol $\langle n \rangle$, meaning the "description of $n$" can refer to

(1) "1111111," when $\langle n \rangle = \langle n \rangle_2$ means the "binary representation of $n$" (a unique string over the alphabet $\{0,1\}$);
(2) "11201," when $\langle n \rangle = \langle n \rangle_3$ means the "base 3 representation of $n$" (a unique string over the alphabet $\{0,1,2\}$);
(3) "one hundred and twenty-seven," when $\langle n \rangle = \langle n \rangle_{\text{English}}$ means the "English representation of $n$" (a unique string over the ASCII alphabet, or at least an alphabet containing the English letters, a comma, a dash, and a space);
(4) "cent vingt-sept," similarly for French, $\langle n \rangle = \langle n \rangle_{\text{French}}$
(5) "wa'vatlh wejmaH Soch," similarly for Klingon[3], $\langle n \rangle = \langle n \rangle_{\text{Klingon}}$;
(6) and good old "127," when $\langle n \rangle = \langle n \rangle_{10}$ means the "decimal representation of $n$."

Note that haven't yet specified whether or not $\epsilon$, the empty string, is considered to be an element of PRIMES.

0.3. **More on Strings.** Chapter 0 of [Sip] uses the following notion:

(1) if $\mathcal{A}$ is an alphabet and $k \in \mathbb{Z}_{\geq 0} = \{0, 1, 2 \ldots\}$, a *string of length $k$ over $\mathcal{A}$* is a sequence of $k$ elements of $\mathcal{A}$;
(2) we use $\mathcal{A}^k$ to denote the set of all strings of length $k$ over $\mathcal{A}$;
(3) equivalently, a string of length $k$ over $\mathcal{A}$ is a map $[k] \to \mathcal{A}$ where $[k] = \{1, \ldots, k\}$;
(4) by consequence (or convention) $\mathcal{A}^0 = \{\epsilon\}$ where $\epsilon$, called the *empty string*, is the unique map $\emptyset \to \mathcal{A}$;
(5) a *string over $\mathcal{A}$* is a string over $\mathcal{A}$ of some length $k \in \mathbb{Z}_{\geq 0}$;
(6) therefore $\mathcal{A}^*$ is given as
$$\mathcal{A}^* = \bigcup_{k \in \mathbb{Z}_{\geq 0}} \mathcal{A}^k = \mathcal{A}^0 \cup \mathcal{A}^1 \cup \mathcal{A}^2 \cup \cdots$$
(7) strings are sometimes called *words* in other literature;
(8) a *letter* or *symbol* of an alphabet, $\mathcal{A}$, is an element of $\mathcal{A}$.

[3] Source: https://en.wikibooks.org/wiki/Klingon/Numbers.

## 1. Some Self-Referencing "Paradoxes" and Theorems

We have seen in Subsection 0.2 that terminology and definitions that seem reasonable, such as our definition of PRIMES, can turn out to be ambiguous and imprecise. Below we give a number of sentences or phrases that are "paradoxes" and seemingly lead to logical contradictions. In each of these sentences or phrases, either:

(1) the "paradox" arises from imprecise or ambiguous language, and the paradox goes away once we make things precise;
(2) these statements prove "theorems," saying that if you can construct such a phrase (or sentence, or algorithm, etc.), then you obtain a contradiction;
(3) both of the above.

All of the sentences or phrases theorems below involve "self-referencing" combined with a logical "negation."

(1) I am lying. [Is this statement true or false?]
(2) This statement is a lie. [Is this statement true or false?]
(3) "the smallest positive integer not defined by an English phrase wth fewer than fifty words" [This is called the "Berry Paradox"; we will discuss this in class.]
(4) Leslie writes about (and only about) all those who don't write about themselves. [Does Leslie write about Leslie?]
(5) "the set of all sets that do not contain themselves" [This is Russell's most famous (and serious) paradox: if $S$ is this set, does $S$ contain itself?]
(6) This is a statement that does not have a proof that it is true. [If you are working in a (consistent and complete) logical system that can build such a statement, then this statement is true but not provably true within your system[4].]
(7) Consider a C program, $P$, that (1) takes as input a string, $i$, (2) figures out if $i$ is the description of a C program that halts on input $i$, and (3) (i) if so, $P$ enters an infinite loop, and (ii) otherwise $P$ stops running (i.e., halts). [What happens when this program is given input $j$ where $j$ is the string representing $P$ ?] [This contradiction shows that $P$ cannot exist; but if the Halting Problem could be decided, then you have a subroutine to implement step (2), and then you could easily write such a C program, $P$.]

These statements and algorithms all "refer to themselves," or can be "applied to themselves;" they also involve some type of "negation." Furthermore, the proof of Cantor's theorem in Section 2 looks very similar to Russell's famous paradox.

Consider the first statement, "I am lying," famously used, of course, by Captain Kirk and Mr. Spock[5] to destroy the leader of an army of robots. This leads to a paradox: if the speaker is telling the truth, then he is lying ("I am lying"), but if he is lying, then he is lying that he is lying, meaning he is telling the truth. Either way we get a contradition.

[4] You can build such statements in systems that, roughly speaking, can express multiplication over the natural numbers and have a set of axioms that is finite or can be generated by a Turing machine.

[5] Thanks to Benjamin Israel for pointing out an earlier inaccuracy.

All the other statements lead to “paradoxes” (of somewhat different types) or theorems; this will be discussed in class and the exercises. When you get a genuine paradox, there are a number of ways of dealing with them, such as:

(1) Ignore the paradox. Carry on regardless.
(2) Admit that the paradox is a valid problem in your foundations, but claim that it doesn't matter in what you are doing.
(3) Try to resolve the paradox.

We will discuss paradoxes (1)–(7) in class.

## 2. Counting, Power Sets, and Countability

In CPSC 421, for any model of computation that we study (e.g., finite automata, Turing machines, Python programs), we can easily show that there exist programs that cannot be solved. The reason is that there are “more” problems than algorithms.

[Unfortunately, this does not identify which problem(s) cannot be solved, it merely shows that unsolvable problems exist.]

More precisely, we will use Cantor's theorem to show that the set of languages over an alphabet is uncountable. This technique uses *diagonalization* (see Theorem 4.17 and Corollary 4.18 in [Sip]) in a way that looks similar to Russell's famous paradox.

### 2.1. **Injections, Surjections, Bijections, and the Size of a Set.**

Any finite set, $S$, has a size, which we denote by $|S|$. For example,

$$\bigl|\{a,b,c\}\bigr| = 3, \quad \bigl|\{X,Y,Z,W\}\bigr| = 4.$$

To say that $S$ is *a smaller set* than $T$, if both are finite sets, just means that $|S| < |T|$.

When working with infinite sets, the notion of one set being *smaller* than another is much more subtle. To compare the “size” of infinite sets one can use the following notions.

**Definition 2.1.** Let $f\colon S \to T$ be a map of sets. We say that $f$ is

(1) *injective* (or *one-to-one*) if for all $s_1, s_2 \in S$ with $s_1 \neq s_2$ we have $f(s_1) \neq f(s_2)$;
(2) *surjective* (or *onto*) if for all $t \in T$ there is an $s \in S$ with $f(s) = t$;
(3) *bijective* (or *a one-to-one correspondence*) if it is injective and surjective.

You should convince yourself that if $S, T$ are finite sets with $|S| < |T|$, then there is no surjective map $S \to T$. See the exercises for related notions.

**Definition 2.2.** Let $S, T$ be two sets. We say that $S$ *is smaller* than $T$ if there is no surjective map $f\colon S \to T$.

There is an alternate definition of “$S$ *is smaller* than $T$,” namely that no map $T \to S$ is injective (we will discuss this in class).

### 2.2. **Countable Sets.**

Chapter 0 of [Sip] uses $\mathcal{N}$ to denote the *natural numbers*

$$\mathbb{N} = \{1, 2, 3, \dots\}.$$

**Definition 2.3.** We say that a set $S$ is *countable* if there is a surjective map $\mathbb{N} \to S$. We say that $S$ is *countably infinite* if $S$ is countable and not a finite set.

The following sets are countably infinite:

(1) $\mathbb{N}$;
(2) the integers $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$;
(3) the positive rational numbers $\mathcal{Q}$ (Example 4.15 in [Sip]);
(4) the rational numbers $\mathbb{Q}$;
(5) the set of words over an alphabet $A$,

$$A^* = \bigcup_{i \geq 0} A^i$$

(an *alphabet* is any nonempty, finite set).

In class we explain why this is true.

**Definition 2.4.** A set is *uncountable* if it is not countable.

Let us gives some examples of uncountable sets:

(1) the real numbers $\mathbb{R}$ (see Theorem 4.17 in [Sip]);
(2) the set of *languages* over an alphabet, $A$, meaning the set of all subsets of $A^*$ (see Corollary 4.18 of [Sip], or Cantor's Theorem below).

2.3. **Cantor's Theorem and Diagonalization.**

**Definition 2.5.** If $S$ is a set, the *power set of* $S$, denoted $\mathrm{Power}(S)$, is the set of all subsets of $S$.

For example, if $S$ is a finite set with $n$ elements, then its power set has $2^n$ ("two to the *power* $n$") elements.

**Theorem 2.6.** *Any set, $S$, is smaller than its power set; i.e., if $f\colon S \to \mathrm{Power}(S)$ is any function, then $f$ is not surjective.*

*Proof.* Given $S$ and such a function, $f$, let

$$T = \{s \in S \mid s \notin f(s)\}.$$

We claim that $T$ is not in the image of $S$: for the sake of contradiction, assume that there is a $t \in S$ such that $f(t) = T$. Then either (1) $t \in T$, or (2) $t \notin T$; we easily see (we will explain this in class) that either case leads to a contradiction. $\square$

In class we will explain why this proof uses *diagonalization*; the proof that $\mathbb{R}$ is uncountable is one way to illustrate this.

**Corollary 2.7.** *Let $A$ be an alphabet. Then the set of languages over $A$ is uncountable.*

In [Sip] we will describe many notions of what is meant by an "algorithms" (e.g., as described by Turing machines, finite automata, Python programs, C programs, etc.). In most such notions the set of algorithms is countable; for example, a program in any fixed language (Python, C, etc.) is just a finite string. It follows that from the above corollary that there exist languages that cannot be solved by any countable set of algorithms.

Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, CANADA.
*E-mail address*: jf@cs.ubc.ca
*URL*: http://www.cs.ubc.ca/~jf