# 11\_17\_19\_ch1.json

# atom 0

In the beginning, our goal is to build up, completely formally/mathematically, the important notions related to computation and algorithms. Our starting point is this chapter, which deals with how to formally represent data and how to formally define the concept of a computational problem.  
  
In theoretical computer science, every kind of data is represented/encoded using finite-length strings. In this chapter, we introduce you to the formal definitions related to strings and encodings of objects with strings. We also present the definitions of ``computational problem'' and ``decision problem''.   
  
All the definitions in this chapter are at the foundation of the formal study of computation.

# atom 1

An \defn{alphabet} is a non-empty, finite set, and is usually denoted by $\Sigma$.   
The elements of $\Sigma$ are called \defn{symbols} or \defn{characters}.

# atom 2

A unary alphabet consists of one symbol. A common choice for that symbol is $\s{1}$.   
So an example of a unary alphabet is $\Sigma = \{\s{1}\}$.

# atom 3

A binary alphabet consists of two symbols.   
Often we represent those symbols using $\s{0}$ and $\s{1}$.   
So an example of a binary alphabet is $\Sigma = \{\s{0},\s{1}\}$.  
Another example of a binary alphabet is $\Sigma=\{\s{a},\s{b}\}$ where $\s{a}$ and $\s{b}$ are the symbols.

# atom 4

A ternary alphabet consists of three symbols.   
So $\Sigma=\{\s{0},\s{1},\s{2}\}$ and $\Sigma=\{\s{a},\s{b},\s{c}\}$ are examples of ternary alphabets.

# atom 5

Given an alphabet $\Sigma$, a \defn{string} (or \defn{word}) over $\Sigma$ is a (possibly infinite) sequence of symbols, written as $a\_1a\_2a\_3\ldots$, where each $a\_i \in \Sigma$.   
The string with no symbols is called the \defn{empty string} and is denoted by $\epsilon$.

# atom 6

For $\Sigma = \{\s{1}\}$, the following is a list of 6 strings over $\Sigma$:   
\[  
 \epsilon,\; \s{1},\; \s{11},\; \s{111},\; \s{1111},\; \s{11111}.  
\]  
Furthermore, the infinite sequence $\s{111111}\ldots$ is also a string over $\Sigma$.

# atom 7

For $\Sigma = \{\s{0},\s{1}\}$, the following is a list of 8 strings over $\Sigma$:   
\[  
\epsilon,\; \s{0},\; \s{1},\; \s{00},\; \s{01},\; \s{10},\; \s{11},\; \s{000}.  
\]   
The infinite strings $\s{000000}\ldots$, $\s{111111}\ldots$ and $\s{010101}\ldots$ are also examples of strings over $\Sigma$.

# atom 8

In our notation of a string, we do not use quotation marks. For instance, we use the notation $\s{1010}$ rather than ``$\s{1010}$'', even though the latter notation using the quotation marks is the standard one in many programming languages. Occasionally, however, we may use quotation marks to distinguish a string like ``$\s{1010}$'' from another type of object with the representation $1010$ (e.g. the binary \emph{number} $1010$).

# atom 9

The \defn{length of a string} $w$, denoted $|w|$, is the the number of symbols in $w$.   
If $w$ has an infinite number of symbols, then the length is undefined.

# atom 10

Let $\Sigma=\{\s{0},\s{1}\}$.   
The length of the wordword $\s{01001}$, denoted by $|\s{01001}|$, is equal to $5$.   
The length of $\epsilon$ is 0.

# atom 11

Let $\Sigma$ be an alphabet.   
We denote by $\Sigma^\*$ the set of \emph{all} strings over $\Sigma$ consisting of finitely many symbols.   
Equivalently, using set notation,  
\[  
 \Sigma^\* = \{a\_1a\_2\ldots a\_n : \text{ $n \in \mathbb{N}$, and $a\_i \in \Sigma$ for all $i$}\}.  
\][Ternary encoding of pairs of naturals]

# atom 12

For $\Sigma = \{\s{a}\}$, $\Sigma^\*$ denotes the set of all finite-length words consisting of $\s{a}$'s.   
So  
\[  
 \{\s{a}\}^\* = \{\epsilon, \s{a}, \s{aa}, \s{aaa}, \s{aaaa}, \s{aaaaa}, \ldots \}.  
\]

# atom 13

For $\Sigma = \{\s{0},\s{1}\}$, $\Sigma^\*$ denotes the set of all finite-length words consisting of $\s{0}$'s and $\s{1}$'s.   
So  
\[  
 \{\s{0},\s{1}\}^\* = \{\epsilon, \s{0}, \s{1}, \s{00}, \s{01}, \s{10}, \s{11}, \s{000}, \s{001}, \s{010}, \s{011}, \s{100}, \s{101}, \s{110}, \s{111}, \ldots\}.  
\]

# atom 14

We often use the words ``string'' and ``word'' to refer to a finite-length stringstring/word.   
When we want to talk about infinite-length stringsstrings, we explicitly use the word ``infinite''.

# atom 15

By Definition~\ref{definition:Alphabet-symbol-character}, an alphabet $\Sigma$ cannot be the empty set.   
This implies that $\Sigma^\*$ is an infinite set since there are infinitely many strings of finite length over a non-empty $\Sigma$. We will later see that $\Sigma^\*$ is always \emph{countably} infinite.

# atom 16

The \defn{reversal of a string} $w = a\_1a\_2\ldots a\_n$, denoted $w^R$, is the string $w^R = a\_na\_{n-1}\ldots a\_1$.

# atom 17

The reversal of $\s{01001}$ is $\s{10010}$.

# atom 18

The reversal of $\s{1}$ is $\s{1}$.

# atom 19

The reversal of $\epsilon$ is $\epsilon$.

# atom 20

The \defn{concatenation of strings} $u$ and $v$ in $\Sigma^\*$, denoted by $uv$ or $u \cdot v$, is the string obtained by joining together $u$ and $v$.

# atom 21

If $u = \s{101}$ and $v = \s{001}$, then $uv = \s{101001}$.

# atom 22

If $u = \s{101}$ and $v = \epsilon$, then $uv = \s{101}$.

# atom 23

If $u = \epsilon$ and $v = \epsilon$, then $uv = \epsilon$.

# atom 24

For $n \in \mathbb{N}$, the $n$'th \defn{power of a string} $u$, denoted by $u^n$, is the word obtained by concatenating $u$ with itself $n$ times.

# atom 25

If $u = \s{101}$ then $u^3 = \s{101101101}$.

# atom 26

For any string $u$, $u^0 = \epsilon$.

# atom 27

We say that a string $u$ is a \defn{substring} of string $w$ if $w = xuy$ for some strings $x$ and $y$.

# atom 28

The string $\s{101}$ is a substring of $\s{11011}$ and also a substring of $\s{0101}$.   
On the other hand, it is not a substring of $\s{1001}$.

# atom 29

Any (possibly infinite) subset $L \subseteq \Sigma^\*$ is called a \defn{language} over the alphabet $\Sigma$.

# atom 30

Let $\Sigma$ be an alphabet.  
Then $L = \{w \in \Sigma^\* : \text{ $|w|$ is even}\}$ is a language.

# atom 31

Let $\Sigma = \{\s{0},\s{1}\}$.  
Then $L = \{\s{101}\}$ is a language.

# atom 32

Let $\Sigma$ be an alphabet.  
Then $L = \Sigma^\*$ is a language.

# atom 33

Let $\Sigma$ be an alphabet.  
Then $L = \varnothing$ is a language.

# atom 34

Since a language is a set, the \emph{size of a language} refers to the size of that set.   
A language can have finite or infinite size.   
This is not in conflict with the fact that every language consists of finite-length stringsstrings.

# atom 35

The language $\{\epsilon\}$ is not the same language as $\varnothing$.   
The former has size $1$ whereas the latter has size $0$.

# atom 36

Let language $L \subseteq \{\s{0},\s{1}\}^\*$ be recursively defined as follows:  
\begin{itemize}  
 \item (base case) $\epsilon \in L$;  
 \item (recursive rule) if $x, y \in L$, then $\s{0}x\s{1}y\s{0} \in L$.  
\end{itemize}  
This means that every word in the language is derived starting from the base case, and applying the recursive rule a finite number of times.   
Show, using (structural) induction, that for any word $w \in L$, the number of $\s{0}$'s in $w$ is exactly twice the number of $\s{1}$'s in $w$.

# atom 37

Let $\mathbf{0}(w)$ denote the number of $\s{0}$'s in $w$ and let $\mathbf{1}(w)$ denote the number of $\s{1}$'s in $w$. Given $L$ as defined above, the question asks us to show that for any $w \in L$, $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$. We will do so by structural induction.\footn{This means that implicitly, the parameter being inducted on is the minimum number of applications of the recursive rule needed to create an object. And in this case, explicitly stating the parameter being inducted on or the induction hypothesis is not needed.}  
The base case corresponds to $w = \epsilon$, and in this case, $\mathbf{0}(w) = \mathbf{1}(w) = 0$, and therefore $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$ holds.  
  
To carry out the induction step, consider an arbitrary word $w \neq \epsilon$ in $L$. Then by the definition of $L$, we know that there exists $x$ and $y$ in $L$ such that $w = \s{0}x\s{1}y\s{0}$. Furthermore, by induction hypothesis,   
\begin{equation\*} %\label{eq:structural-induction-1}  
 \mathbf{0}(x) = 2 \cdot \mathbf{1}(x) \quad \quad (\*)  
\end{equation\*}  
and   
\begin{equation\*} %\label{eq:structural-induction-2}  
 \mathbf{0}(y) = 2 \cdot \mathbf{1}(y). \quad \quad (\*\*)  
\end{equation\*}  
We are done once we show $\mathbf{0}(w) = 2 \cdot \mathbf{1}(w)$. We establish this via the following chain of equalities:   
\begin{align\*}  
 \mathbf{0}(w) & = 2 + \mathbf{0}(x) + \mathbf{0}(y) & \text{since $w = \s{0}x\s{1}y\s{0}$} \\  
 & = 2 + 2 \cdot \mathbf{1}(x) + 2 \cdot \mathbf{1}(y) & \text{by $(\*)$ and $(\*\*)$}\\  
 & = 2 \cdot (1 + \mathbf{1}(x) + \mathbf{1}(y)) \\  
 & = 2 \cdot \mathbf{1}(w).  
\end{align\*}

# atom 38

The \defn{reversal of a language} $L \subseteq \Sigma^\*$, denoted $L^R$, is the language  
\[  
 L^R = \{w^R \in \Sigma^\* : w \in L\}.   
\]

# atom 39

The reversal of the languagereversal of the languagelanguage $\{\epsilon, \s{1}, \s{1010}\}$ is $\{\epsilon, \s{1}, \s{0101}\}$.

# atom 40

The \defn{concatenation of languages} $L\_1, L\_2 \subseteq \Sigma^\*$, denoted $L\_1L\_2$ or $L\_1 \cdot L\_2$, is the language  
\[  
 L\_1L\_2 = \{uv \in \Sigma^\* : u \in L\_1, v \in L\_2\}.  
\]

# atom 41

The concatenation of languagesconcatenation of languageslanguages $\{\epsilon, \s{1}\}$ and $\{\s{0}, \s{01}\}$ is the language  
\[  
 \{\s{0}, \s{01}, \s{10}, \s{101}\}.  
\]

# atom 42

For $n \in \N$, the $n$'th \defn{power of a language} $L \subseteq \Sigma^\*$, denoted $L^n$, is the language obtained by concatenating $L$ with itself $n$ times, that is,\footn{We can omit parentheses as the order in which the concatenation $\cdot$ is applied does not matter.}  
\[  
 L^n = \underbrace{L \cdot L \cdot L \cdots L}\_{n \text{ times}}.  
\]  
Equivalently,   
\[  
 L^n = \{u\_1u\_2\cdots u\_n \in \Sigma^\* : u\_i \in L \text{ for all } i \in \{1,2,\ldots,n\}\}.  
\]

# atom 43

The 3rd power of $\{\s{1}\}$ is the language $\{\s{111}\}$.

# atom 44

The 3rd power of $\{\epsilon, \s{1}\}$ is the language $\{\epsilon, \s{1}, \s{11}, \s{111}\}$.

# atom 45

The 0th power of any languagelanguage $L$ is the language $\{\epsilon\}$.

# atom 46

The \defn{star of a language} $L \subseteq \Sigma^\*$, denoted $L^\*$, is the language   
\[  
L^\* = \bigcup\_{n \in \N} L^n.  
\]  
Equivalently,   
\[  
L^\* = \{u\_1u\_2\cdots u\_n \in \Sigma^\* : n \in \N, u\_i \in L \text{ for all } i \in \{1,2,\ldots,n\}\}.  
\]

# atom 47

Given an alphabet $\Sigma$, consider the language $L = \Sigma \subseteq \Sigma^\*$\footn{Technically $L$ is a set of strings and $\Sigma$ is a set of symbols, so the equality notation is not entirely accurate. Hopefully the intention is clear however: symbols can be viewed as length-1 strings.}. Then $L^\*$ is equal to $\Sigma^\*$.

# atom 48

If $L = \{\s{00}\}$, then $L^\*$ is the language consisting of all words containing an even number of $\s{0}$'s and no other symbol.

# atom 49

Let $L$ be the language consisting of all words containing an even number of $\s{0}$'s and no other symbol. Then $L^\* = L$.

# atom 50

Prove or disprove: If $L\_1, L\_2 \subseteq \{\s{a},\s{b}\}^\*$ are languages, then $(L\_1 \cap L\_2)^\* = L\_1^\* \cap L\_2^\*$.

# atom 51

We disprove the statement by providing a counterexample. Let $L\_1 = \{\s{a}\}$ and $L\_2 = \{\s{aa}\}$. Then $L\_1 \cap L\_2 = \emptyset$, and so $(L\_1 \cap L\_2)^\* = \{\epsilon\}$. On the other hand, $L\_1^\* \cap L\_2^\* = L\_2^\* = \{\s{aa}\}^\*$.

# atom 52

Is it true that for any language $L$, $(L^\*)^R = (L^R)^\*$? Prove your answer.

# atom 53

We will prove that for any language $L$, $(L^\*)^R = (L^R)^\*$. To do this, we will first argue $(L^\*)^R \subseteq (L^R)^\*$ and then argue $(L^R)^\* \subseteq (L^\*)^R$.  
  
To show the first inclusion, it suffices to show that any $w \in (L^\*)^R$ is also contained in $(L^R)^\*$. We do so now. Take an arbitrary $w \in (L^\*)^R$. Then for some $n \in \N$, $w = (u\_1u\_2\ldots u\_n)^R$, where $u\_i \in L$ for each $i$. Note that $w = (u\_1u\_2\ldots u\_n)^R = u\_n^R u\_{n-1}^R \ldots u\_1^R$, and $u\_i^R \in L^R$ for each $i$. Therefore $w \in (L^R)^\*$.  
  
To show the second inclusion, it suffices to show that any $w \in (L^R)^\*$ is also contained in $(L^\*)^R$. We do so now. Take an arbitrary $w \in (L^R)^\*$. This means that for some $n \in \N$, $w = v\_1 v\_2 \ldots v\_n$, where $v\_i \in L^R$ for each $i$. For each $i$, define $u\_i = v\_i^R$ (and so $u\_i^R = v\_i$). Note that each $u\_i \in L$ because $v\_i \in L^R$. We can now rewrite $w$ as $w = u\_1^R u\_2^R \ldots u\_n^R$, which is equal to $(u\_n u\_{n-1} \ldots u\_1)^R$. Since each $u\_i \in L$, this shows that $w \in (L^\*)^R$.  
  
Since we have shown both $(L^\*)^R \subseteq (L^R)^\*$ and $(L^R)^\* \subseteq (L^\*)^R$, we conclude that $(L^\*)^R = (L^R)^\*$.

# atom 54

Let $A$ be a set (which is possibly countably infinite\footn{We assume you know what a countable set is, however, this concept is reviewed in the next chapter.}), and let $\Sigma$ be an alphabet.   
An \defn{encoding} of the elements of $A$, using $\Sigma$, is an injective function $\text{Enc}: A \to \Sigma^\*$.   
We denote the encodinencoding of $a \in A$ by $\langle a \rangle$.\footn{Note that this angle-bracket notation does not specify the underlying encoding function as the particular choice of encoding function is often unimportant.}   
  
If $w \in \Sigma^\*$ is such that there is some $a \in A$ with $w = \langle a \rangle$, then we say $w$ is a \defn{valid encoding} of an element in $A$.   
  
A set that can be encoded is called \defn{encodable}.\footn{Not every set is encodable. Can you figure out exactly which sets are encodable?}

# atom 55

When we (humans) communicate numbers among ourselves, we usually use the base-10 representation, which corresponds to an encoding of $\N$ using the alphabet $\Sigma = \{\s{0},\s{1},\s{2},\ldots, \s{9}\}$. For example, we encode the number four as $\s{4}$ and the number twelve as $\s{12}$.

# atom 56

As you know, every number has a base-$2$ representation (which is also known as the binary representation). This representation corresponds to an encoding of $\N$ using the alphabet $\Sigma = \{\s{0},\s{1}\}$. For example, four is encoded as $\s{100}$ and twelve is encoded as $\s{1100}$.

# atom 57

An integer is a natural number together with a sign, which is either negative or positive. Let $\text{Enc} : \N \to \{\s{0},\s{1}\}^\*$ be any binary encoding of $\N$. Then we can extend this encoding to an encoding of $\Z$, by defining $\text{Enc}':\Z \to \{\s{0},\s{1}\}^\*$ as follows:  
\[  
\text{Enc}'(x) =   
\begin{cases}  
\s{0}\text{Enc}(x) & \text{if $x \geq 0$}, \\  
\s{1}\text{Enc}(|x|) & \text{if $x < 0$}.  
\end{cases}  
\]  
Effectively, this encoding of integers takes the encoding of natural numbers and precedes it with a bit indicating the integer's sign.

# atom 58

It is possible (and straightforward) to encode the natural numbers using the alphabet $\Sigma = \{\s{1}\}$ as follows. Let $\text{Enc}(n) = \s{1}^n$ for all $n \in \N$.

# atom 59

Suppose we want to encode the set $A = \N \times \N$ using the alphabet $\Sigma = \{\s{0},\s{1},\s{2}\}$. One way to accomplish this is to make use of a binary encoding $\text{Enc}': \N \to \{\s{0},\s{1}\}^\*$ of the natural numbers. With $\text{Enc}'$ in hand, we can define $\text{Enc} : \N \times \N \to \{\s{0},\s{1},\s{2}\}^\*$ as follows. For $(x,y) \in \N \times \N$, $\text{Enc}(x,y) = \text{Enc}'(x)\s{2}\text{Enc}'(y)$. Here the symbol $\s{2}$ acts as a separator between the two numbers. To make the separator symbol advertise itself as such, we usually pick a symbol like $\s{\$}$ rather than $\s{2}$. So the ternary alphabet is often chosen to be $\Sigma = \{\s{0},\s{1},\s{\$}\}$.

# atom 60

Having a ternary alphabet to encode pairs of naturals was convenient since we could use the third symbol as a separator. It is also relatively straightforward to take that ternary encoding and turn it into a binary encoding, as follows. Encode every element of the ternary alphabet in binary using two bits. For instance, if the ternary alphabet is $\Sigma = \{\s{0},\s{1},\s{\$}\}$, then we could encode $\s{0}$ as $\s{00}$, $\s{1}$ as $\s{01}$ and $\s{\$}$ as $\s{11}$. This mapping allows us to convert any encoded string over the ternary alphabet into a binary encoding. For example, a string like $\s{\$0\$1}$ would have the binary representation $\s{11001101}$.

# atom 61

Let $A$ be the set of all undirected graphs.\footn{We will define graphs formally in a future chapter, however, we assume you are already familiar with the concept.} Every graph $G=(V,E)$ can be represented by its $|V|$ by $|V|$ adjacency matrix. In this matrix, every row corresponds to a vertex of the graph, and similarly, every column corresponds to a vertex of the graph. The $(i,j)$'th entry contains a $1$ if $\{i, j\}$ is an edge, and contains a $0$ otherwise. Below is an example.  
\begin{center}  
 \includegraphics[width=10cm]{01\_Strings\_and\_Encodings/media/graph-adj-matrix.png}  
\end{center}  
Such a graph can be encoded using a ternary alphabet as follows. Take the adjacency matrix of the graph, view each row as a binary string, and concatenate all the rows by putting a separator symbol between them. The encoding of the above example would be  
\[  
\langle G \rangle = \s{0101\$1010\$0101\$1010}.  
\]

# atom 62

Let $A$ be the set of all functions in the programming language Python. Whenever we type up a Python function in a code editor, we are creating a string representation/encoding of the function, where the alphabet is all the Unicode symbols.\footn{\url{https://en.wikipedia.org/wiki/Unicode}} For example, consider a Python function named absValue, which we can write as  
\begin{verbatim}  
def absValue(N):  
 if (N < 0): return -N  
 else: return N  
\end{verbatim}  
By writing out the function, we have already encoded it. More specifically, $\langle \text{absValue} \rangle$ is the string  
% \[  
% \s{def absValue(N):\n if (N < 0): return -N\n else: return N}  
% \]  
\begin{verbatim}  
def absValue(N):\n if (N < 0): return -N\n else: return N  
\end{verbatim}  
%(Here we are using quotation marks to denote the string so it is clear that the period at the end does not belong to the string.)

# atom 63

Describe an encoding of $\Z$ using the alphabet $\Sigma = \{\s{1}\}$.

# atom 64

Let $\text{Enc}:\Z \to \{\s{1}\}^\*$ be defined as follows:  
\[  
\text{Enc}(x) =   
\begin{cases}  
\s{1}^{2x-1} & \text{if $x > 0$}, \\  
\s{1}^{-2x} & \text{if $x \leq 0$.}  
\end{cases}  
\]  
This solution is inspired by thinking of a bijection between integers and naturals. Indeed, the function $f: \Z \to \N$ defined by   
\[  
f(x) =   
\begin{cases}  
2x-1 & \text{if $x > 0$}, \\  
-2x & \text{if $x \leq 0$,}  
\end{cases}  
\]  
is such a bijection.

# atom 65

Let $\Sigma$ be an alphabet. Any function $f: \Sigma^\* \to \Sigma^\*$ is called a \defn{computational problem} over the alphabet $\Sigma$.

# atom 66

Consider the function $g:\N \times \N \to \N$ defined as $g(x, y) = x + y$.   
This is a function that expresses the addition problem in naturals.   
We can view $g$ as a computational problem over an alphabet $\Sigma$ once we fix an encoding of the domain $\N \times \N$ using $\Sigma$ and an encoding of the codomain $\N$ using $\Sigma$.   
For convenience, we take $\Sigma = \{\s{0},\s{1},\s{\$}\}$. Let $\text{Enc}$ be the encoding of $\N \times \N$ as described in Example~\ref{example:Ternary-encoding-of-pairs-of-naturals}.   
Let $\text{Enc}'$ be the encoding of $\N$ as described in Example~\ref{example:Binary-encoding-of-naturals}.   
Note that $\text{Enc}'$ leaves the symbol $\s{\$}$ unused in the encoding.   
We now define the computational problem $f$ corresponding to $g$.   
If $w \in \Sigma^\*$ is a word that corresponds to a valid encodingencoding of a pair of numbers $(x, y)$ (i.e., $\text{Enc}(x,y) = w$), then define $f(w)$ to be $\text{Enc}'(x+y)$.   
If $w \in \Sigma^\*$ is \emph{not} a word that corresponds to a valid encodingencoding of a pair of numbers (i.e., $w$ is not in the image of $\text{Enc}$), then define $f(w)$ to be $\s{\$}$.   
In the codomain, the $\s{\$}$ symbol serves as an ``error'' indicator.

# atom 67

A computational problemcomputational problem is often derived from a function $g: I \to S$, where $I$ is a set of objects called \emph{instances} and $S$ is a set of objects called \emph{solutions}.   
The derivation is done through encodings $\text{Enc}: I \to \Sigma^\*$ and $\text{Enc}': S \to \Sigma^\*$.   
With these encodings, we can create the computational problem $f : \Sigma^\* \to \Sigma^\*$.   
In particular, if $w = \langle x \rangle$ for some $x \in I$, then we define $f(w)$ to be $\text{Enc}'(g(x))$.  
  
\begin{center}  
\includegraphics[width=6cm]{01\_Strings\_and\_Encodings/media/encoding-comm-diagram.png}  
\end{center}  
  
One thing we have to be careful about is defining $f(w)$ for a word $w \in \Sigma^\*$ that does not correspond to an encoding of an object in $I$ (such a word does not correspond to an instance of the computational problem).   
To handle this, we can identify one of the strings in $\Sigma^\*$ as an \emph{error} string and define $f(w)$ to be that string.

# atom 68

Let $\Sigma$ be an alphabet. Any function $f: \Sigma^\* \to \{0,1\}$ is called a \defn{decision problem} over the alphabet $\Sigma$.   
The codomain of the function is not important as long as it has two elements.   
Other common choices for the codomain are $\{\text{No}, \text{Yes}\}$, $\{\text{False}, \text{True}\}$ and $\{ \text{Reject}, \text{Accept}\}$.

# atom 69

Consider the function $g:\N \to \{\text{False}, \text{True}\}$ such that $g(x) = \text{True}$ if and only if $x$ is a prime number.  
We can view $g$ as a decision problem over an alphabet $\Sigma$ once we fix an encoding of the domain $\N$ using $\Sigma$.   
Take $\Sigma = \{\s{0},\s{1}\}$.   
Let $\text{Enc}$ be the encoding of $\N$ as described in Example~\ref{example:Binary-encoding-of-naturals}.   
We now define the decision problem $f$ corresponding to $g$.   
If $w \in \Sigma^\*$ is a word that corresponds to an encodingencoding of a prime number, then define $f(w)$ to be $\text{True}$. Otherwise, define $f(w)$ to be $\text{False}$. (Note that in the case of $f(w) = \text{False}$, either $w$ is the encodingencoding of a composite number, or $w$ is not a valid encodingencodingencoding of a natural number.

# atom 70

As with a computational problemcomputational problemproblem, a decisiondecision problem is often derived from a function $g: I \to \{0,1\}$, where $I$ is a set of instances.   
The derivation is done through an encoding $\text{Enc}: I \to \Sigma^\*$, which allows us to define the decision problem $f: \Sigma^\* \to \{0,1\}$. Any word $w \in \Sigma^\*$ that does not correspond to an encoding of an instance is mapped to $0$ by $f$.

# atom 71

There is a one-to-one correspondence between decision problems and languages. Let $f:\Sigma^\* \to \{0,1\}$ be some decision problem. Now define $L \subseteq \Sigma^\*$ to be the set of all words in $\Sigma^\*$ that $f$ maps to 1. This $L$ is the language corresponding to the decision problem $f$. Similarly, if you take any language $L \subseteq \Sigma^\*$, we can define the corresponding decision problem $f:\Sigma^\* \to \{0,1\}$ as $f(w) = 1$ if and only if $w \in L$. We consider the set of languages and the set of decision problems to be the same set of objects.  
\begin{center}  
 \includegraphics[width=12cm]{01\_Strings\_and\_Encodings/media/decision-problem-language.png}  
\end{center}