# CS 13

# Mathematical Foundations of Computer Science

# Number Representation

**Definition (Symbols and Strings)**

- Let $\Sigma$ be a set of symbols.
- Let $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.

$\Sigma$ is called an **alphabet**, $x \in \Sigma$ is a **symbol**, and $s \in \Sigma^+$ is a **string** (note that we're omitting the "empty string" in our definition here).

$\Sigma = \{a\}$

$\Sigma^*$

$\Sigma^2 = \Sigma \times \Sigma = \{(a,a)\} = \{aa\}$

$\Sigma \times \Sigma \times \Sigma = \{(a,a,a)\}$

$\Sigma^* = \{\varepsilon\} \cup \Sigma^+$

## Symbols and Strings

**Definition (Symbols and Strings)**

- Let $\Sigma$ be a set of symbols.
- Let $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.

$\Sigma$ is called an **alphabet**; $x \in \Sigma$ is a **symbol**, and $s \in \Sigma^+$ is a **string** (note that we're omitting the "empty string" in our definition here).

**Example (Binary Numbers)**

Let $\Sigma = \{0, 1\}$. Then, $\Sigma^+ = \{0, 1\} \cup \{00, 01, 10, 11\} \cup \cdots$. That is, $\Sigma^+$ is the set of all binary numbers.

### Definition (Symbols and Strings)

- Let $\Sigma$ be a set of symbols.
- Let $\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.

$\Sigma$ is called an **alphabet**; $x \in \Sigma$ is a **symbol**, and $s \in \Sigma^+$ is a **string** (note that we're omitting the "empty string" in our definition here).

### Example (Binary Numbers)

Let $\Sigma = \{0, 1\}$. Then, $\Sigma^+ = \{0, 1\} \cup \{00, 01, 10, 11\} \cup \cdots$. That is, $\Sigma^+$ is the set of all binary numbers.

### Connection to CS 21

You'll see the ideas of **grammars**, **decision problems**, and **regular expressions** which are all fundamentally based on this definition of strings.

**Example (Unary Numbers)**

Let $\Sigma = \{\triangle\}$. Then, $\Sigma^+ = \{\triangle\} \cup \{\triangle\triangle\} \cup \cdots$. That is, $\Sigma^+$ is the set of all numbers represented with a single symbol (i.e., **unary** numbers).

**Example (Unary Numbers)**

Let $\Sigma = \{\triangle\}$. Then, $\Sigma^+ = \{\triangle\} \cup \{\triangle\triangle\} \cup \cdots$. That is, $\Sigma^+$ is the set of all numbers represented with a single symbol (i.e., **unary** numbers).

**Example (Binary Numbers)**

Let $\Sigma = \{0, 1\}$. Then, $\Sigma^+ = \{0, 1\} \cup \{00, 01, 10, 11\} \cup \cdots$. That is, $\Sigma^+$ is the set of all binary numbers.

**Example (Unary Numbers)**

Let $\Sigma = \{\triangle\}$. Then, $\Sigma^+ = \{\triangle\} \cup \{\triangle\triangle\} \cup \cdots$. That is, $\Sigma^+$ is the set of all numbers represented with a single symbol (i.e., **unary** numbers).

**Example (Binary Numbers)**

Let $\Sigma = \{0, 1\}$. Then, $\Sigma^+ = \{0, 1\} \cup \{00, 01, 10, 11\} \cup \cdots$. That is, $\Sigma^+$ is the set of all binary numbers.

**Example (Decimal Numbers)**

Let $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Then, $\Sigma^+$ is the set of all base-10 numbers (e.g., **decimal** numbers).

$X \times Y = \{(x, y) : x \in X, y \in Y\}$

## Example (Unary Numbers)

Let $\Sigma = \{\triangle\}$. Then, $\Sigma^+ = \{\triangle\} \cup \{\triangle\triangle\} \cup \cdots$. That is, $\Sigma^+$ is the set of all numbers represented with a single symbol (i.e., **unary** numbers).

## Example (Binary Numbers)

Let $\Sigma = \{0, 1\}$. Then, $\Sigma^+ = \{0, 1\} \cup \{00, 01, 10, 11\} \cup \cdots$. That is, $\Sigma^+$ is the set of all binary numbers.

## Example (Decimal Numbers)

Let $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Then, $\Sigma^+$ is the set of all base-10 numbers (e.g., **decimal** numbers).

## But What Does It **MEAN**?

Unfortunately, these are just "strings" and don't **actually mean anything**. To fix this, we'll define what we call a **valuation** function for each numerical system to explain how to **interpret** the strings of symbols.

Let $\Sigma = \{\triangle\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N} \setminus \{0\}$, such that:

- $V(\triangle) = 1$
- $V(\triangle X) = 1 + V(X)$ for all $X \in \Sigma^+$

Let $\Sigma = \{\triangle\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N} \setminus \{0\}$, such that:

- $V(\triangle) = 1$
- $V(\triangle X) = 1 + V(X)$ for all $X \in \Sigma^+$

$\forall k \, (V(\Delta^k) = k)$ $\quad \Delta^k = \underbrace{\Delta\Delta \cdots \Delta}_{k}$

**Existence (surjectivity of $V$)**

$V(\Delta^1) = 1$ by def

$V(\Delta^{k+1}) = V(\Delta \Delta^k) = 1 + \underbrace{V(\Delta^k)}_{k}$

$V(\text{"}\Delta\text{"} + x) = 1 + V(x)$

Let $\Sigma = \{\triangle\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N} \setminus \{0\}$, such that:

- $V(\triangle) = 1$
- $V(\triangle X) = 1 + V(X)$ for all $X \in \Sigma^+$

**Uniqueness (injectivity of $V$)**

$V(X) = V(Y) \Rightarrow X = Y$

$V(\Delta^k) = k$

Let $k \in \mathbb{N} \setminus \{0\}$, $\ell > k$

$V(\Delta^k) = k < \ell = V(\Delta^\ell)$

$V$

Let $\Sigma = \{\triangle\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N} \setminus \{0\}$, such that:

- $V(\triangle) = 1$
- $V(\triangle X) = 1 + V(X)$ for all $X \in \Sigma^+$

## Uniqueness (injectivity of $V$)

**Lemma.** We show that $V$ is strictly increasing based on the length of the input. That is, for all $k \in \mathbb{N} \setminus \{0\}$, if $k < \ell$, then $V(\triangle^k) < V(\triangle^\ell)$.
We go by strong induction.

- Base Case ($\ell = 1$). Vacuously, this claim holds since there are no $k < 1$.
- Induction Hypothesis: Suppose for some $\ell \in \mathbb{N} \setminus \{0\}$, for all $k \in \mathbb{N} \setminus \{0\}$, if $k < \ell$, then $V(\triangle^k) < V(\triangle^\ell)$.
- Induction Step. Let $k \in \mathbb{N} \setminus \{0\}$ where $k < \ell + 1$. Then, $V(\triangle^{\ell+1}) = 1 + V(\triangle^\ell) \geq 1 + V(\triangle^k) > V(\triangle^k)$.

**Proof.** We show that if $V(X) = V(Y)$, then $X = Y$ by contrapositive. Suppose $X \neq Y$. Then, $X = \triangle^k$ and $Y = \triangle^\ell$ for some $k, \ell \in \mathbb{N} \setminus \{0\}$ where $k \neq \ell$. Without loss of generality, assume $k < \ell$. Then, by the lemma $V(k) < V(\ell)$ which means they are not equal.

Let $\Sigma = \{0, 1\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N}$, such that:

- $V(b) = b$ for all $b \in \Sigma$
- $V(Xb) = 2V(X) + b$ for all $X \in \Sigma^+$, for $b \in \Sigma$

**Find and prove a summation form for $V$**

We claim a summation form for $V$ is $V(b_{n-1}b_{n-2}\cdots b_0) = \sum 2^i b_i$

$\forall b \in \Sigma^+ \quad V(b) = V(b_{n-1}b_{n-2}\cdots b_0) = \sum 2^i b_i$

Suppose the claim holds for all strings of length $k$ for some $k \in \mathbb{N} \setminus \{0\}$

$b_k b_{k-1} \cdots b_1$

Let $\Sigma = \{0,1\}$. Define our valuation function, $V : \Sigma^+ \to \mathbb{N}$, such that:

- $V(b) = b$ for all $b \in \Sigma$
- $V(Xb) = 2V(X) + b$ for all $X \in \Sigma^+$, for $b \in \Sigma$

### Find and prove a summation form for $V$

We claim a summation form for $V$ is $V(b_{n-1}b_{n-2}\cdots b_0) = \sum_{k=0}^{n-1} b_k 2^k$.

We go by induction on the length of the string.

# Fixed-Width Binary Numbers

The assembly instructions our computers use only work on a fixed number of bits. That is, basic operations act on vectors of $\{0,1\}^w$ for some fixed width $w$.

The assembly instructions our computers use only work on a fixed number of bits. That is, basic operations act on vectors of $\{0,1\}^w$ for some fixed width $w$.

Let's look at addition. To make our machine work, we need add to output a vector of $w$ bits, like so:

$$\mathsf{add} : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$$

The assembly instructions our computers use only work on a fixed number of bits. That is, basic operations act on vectors of $\{0,1\}^w$ for some fixed width $w$.

Let's look at addition. To make our machine work, we need add to output a vector of $w$ bits, like so:

$$\mathsf{add} : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$$

As above, we have $V(b_{w-1}b_{w-2}\cdots b_0) = \sum_{k=0}^{w-1} b_k 2^k$.

The assembly instructions our computers use only work on a fixed number of bits. That is, basic operations act on vectors of $\{0,1\}^w$ for some fixed width $w$.

Let's look at addition. To make our machine work, we need add to output a vector of $w$ bits, like so:

$$\text{add} : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$$

As above, we have $V(b_{w-1}b_{w-2}\cdots b_0) = \sum_{k=0}^{w-1} b_k 2^k$.

Unfortunately, this formula can "overflow" and need $w+1$ bits to be represented. To fix this, we can define add as:

$$\text{add}(a,b) = [V(a) + V(b)] \bmod 2^w$$

Notably, $V$ always outputs a **non-negative** number which is a problem because we'd like to be able to represent negative numbers in binary. To fix this, we define an alternate valuation function as follows:

$$S(b_{w-1}b_{w-2}\cdots b_0) = -b_{w-1}2^{w-1} + \sum_{k=0}^{w-2} b_k 2^k$$

Notably, $V$ always outputs a **non-negative** number which is a problem because we'd like to be able to represent negative numbers in binary. To fix this, we define an alternate valuation function as follows:

$$S(b_{w-1}b_{w-2}\cdots b_0) = -b_{w-1}2^{w-1} + \sum_{k=0}^{w-2} b_k 2^k$$

Note that the co-domain of $S$ is $[-2^{w-1}, 2^{w-1}-1]$ make it **not symmetric**.

Notably, $V$ always outputs a **non-negative** number which is a problem because we'd like to be able to represent negative numbers in binary. To fix this, we define an alternate valuation function as follows:

$$S(b_{w-1}b_{w-2}\cdots b_0) = -b_{w-1}2^{w-1} + \sum_{k=0}^{w-2} b_k 2^k$$

Note that the co-domain of $S$ is $[-2^{w-1}, 2^{w-1}-1]$ make it **not symmetric**.

Interestingly, our previous definition of add still works perfectly for this system.

Notably, $V$ always outputs a **non-negative** number which is a problem because we'd like to be able to represent negative numbers in binary. To fix this, we define an alternate valuation function as follows:

$$S(b_{w-1}b_{w-2}\cdots b_0) = -b_{w-1}2^{w-1} + \sum_{k=0}^{w-2} b_k 2^k$$

Note that the co-domain of $S$ is $[-2^{w-1}, 2^{w-1}-1]$ make it **not symmetric**.

Interestingly, our previous definition of add still works perfectly for this system.

We call this representation **Two's Complement**, and it's how your computer represents signed numbers internally.

Notably, $V$ always outputs a **non-negative** number which is a problem because we'd like to be able to represent negative numbers in binary. To fix this, we define an alternate valuation function as follows:

$$S(b_{w-1}b_{w-2}\cdots b_0) = -b_{w-1}2^{w-1} + \sum_{k=0}^{w-2} b_k 2^k$$

Note that the co-domain of $S$ is $[-2^{w-1}, 2^{w-1}-1]$ make it **not symmetric**.

Interestingly, our previous definition of add still works perfectly for this system.

We call this representation **Two's Complement**, and it's how your computer represents signed numbers internally.

**Connection to CS 24**

You'll see Two's Complement come up repeatedly in CS 24 where we actually work with memory at the bit level.