Problem coding. Language and alphabet. Reasonable encoding rule.

Abstract problems

- To better understand problems and algorithms that solve them we need a concrete definition of what a problem is.
- We define an **abstract problem** Q to be a binary relation on a set I of problem instances and a set S of problem solutions.
- For example, an instance for SHORTEST-PATH is a triple consisting of a graph and two vertices. A solution is a sequence of vertices in the graph, with perhaps the empty sequence denoting that no path exists.

  The problem SHORTEST-PATH itself is the relation that associates each instance of a graph and two vertices with a shortest path in the graph that connects the two vertices. Since shortest paths are not necessarily unique, a given problem instance may have more than one solution.
- In case of abstract decision problems the set of solutions may be $\{0, 1\}$ where 1 denotes yes and 0 denotes no.

Encoding

- In order for a computer program to solve an abstract problem, we must represent problem instances in a way that the program understands.
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings. We can use encodings to map abstract problems to concrete problems.
- Thus, a computer algorithm that “solves” some abstract decision problem actually takes an encoding of a problem instance as input. We call a problem whose instance set is the set of binary strings a **concrete problem**.

- We say that an algorithm solves a concrete problem in time $\Theta(T(n))$ if, when it is provided a problem instance i of length $n = |i|$, the algorithm can produce the solution in $\Theta(T(n))$ time (T is just some function).
- Example:
  - Given an abstract decision problem Q mapping an instance set I to $\{0, 1\}$, an encoding $e : I \to \{0, 1\}^*$ can induce a related concrete decision problem, which we denote by $e(Q)$.
  - If the solution to an abstract-problem instance $i \in I$ is $Q(i) \in \{0, 1\}$ then the solution to the concrete-problem instance $e(i) \in \{0, 1\}^*$ is also $Q(i)$.
  - As a technicality, some binary strings might represent no meaningful abstract-problem instance. For convenience, we shall assume that any such string maps arbitrarily to 0.

Reasonable encoding

- We would like to extend the definition of time complexity of a concrete problem to an abstract problem. Unfortunately the time complexity of an algorithms depends quite heavily on the used encoding.
- Example:
  - Suppose that an integer k is the only input to an algorithm and the time complexity of that algorithm is $\Theta(k)$.
  - If the integer k is provided as a string of 1s and the value of k is the length of the string (unary representation) then the complexity of that algorithm is linear with respect to the input size n: $\Theta(k) = \Theta(n)$.
  - However when we use binary representation of numbers then the length of our input is $n = \lfloor \lg k \rfloor + 1$, thus the time complexity of the algorithm with respect to the input size is $\Theta(k) = \Theta(2^n)$, which is different then the above.
  - Thus the encoding used matters a lot when we want to analyse the complexity of an algorithm.

- In general, if we rule out such impractical representation as unary representation it doesn't make a huge difference in practice what encoding do we use. For example representing an integer in base 2 versus representing it in base 3 will be insignificant in the overall complexity analysis.
- In order to be able to talk about complexity of an algorithm without caring about the encoding we should assume that problem instances are encoded in a **reasonable** fashion.
- The general rule is:
  - Use a reasonable encoding (like binary instead of unary)
  - Don't use unnecessary elements

Alphabet

- An alphabet is a finite non-empty set of symbols.
- Alphabets are usually denoted by Σ.
- Examples:
  - $\Sigma = \{a, b, c\}$ is an alphabet
  - $\Sigma = \{0, 1\}$ is an alphabet
  - $\Sigma = \{\}$ is NOT an alphabet
  - $\Sigma = \mathbb{R}$ is NOT an alphabet

String

- A string (or a word) is a sequence of symbols from the alphabet.
- A string which contains no letters is also technically a string and is usually denoted as λ (empty string)

Language

- A language is a subset of all possible strings. We define all strings that can be generated with an alphabet Σ as Σ^* where * is the Kleene star. Then an alphabet L is just a subset of that set $L \subseteq \Sigma^*$.
- Example. Let $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, \ldots\}$. We can define a language as subset of those strings, for example a subset of strings that don't have any leading 0s.

Formal language and algorithms

- The formal-language framework allows us to express concisely the relation between decision problems and algorithms that solve them.
- Example:
  - From the point of view of language theory, the set of instances for any encoded decision problem Q is simply the set Σ^*, where $\Sigma = \{0, 1\}$.
  - Then we can say that the problem Q can be viewed as a language L (subset of $\Sigma^* = I$) which consists of all problem instances that produce the output 1 (yes).
  - In other words $L = \{x \in \Sigma^* : Q(x) = 1\}$