Computability In Other Domains

Clearly the turing machines we have discussed can only compute functions over strings over a finite alphabet. What about computing functions over arbitrary (countable) domains such as the naturals, integers or the rationals or even cartesian products of such domains or even more generally functions from one domain to another?

Clearly if the elements of such domains can be represented as strings over an alphabet then we could talk about Turing machine computations over such domains.

Def. A set S is **represented** by (elements of) another set R if there exists a partial surjective function $i : R \to S$ called the **interpretation** of R onto S.

not every element in R need have a meaning in S

every element of S should have a representation in R

On the other hand two or more elements of R may be interpreted as representing the same element in S.

Further notice that since i is a function, $i^{-1} : S \to 2^R - \{\varphi\}$ defines a partitioning of $\text{Range}(i^{-1})$ into equivalence classes such that $s_1 \neq s_2 \Rightarrow i^{-1}(s_1) \cap i^{-1}(s_2) = \varphi$. and a class of "gibberish" elements

Example. Consider the unary representation of the naturals as strings of '1's terminated by a single '0'.

Clearly $i : \{0,1\}^* \longrightarrow \mathbb{N}$

is a *partial* function because strings such as '00100' have no interpretation in $\mathbb{N}$. i is also *surjective* since every natural does have a representation in $\{0,1\}^*$. If we were to allow "leading 0s" in the representation, then each natural has more than one possible representation.

Given two sets S_1 and S_2 represented by R_1 and R_2 respectively through interpretations i_1 and i_2 respectively;

$$\begin{array}{ccc} R_1 & \xrightarrow{i_1} & S_1 \\ f \downarrow & & \downarrow g \\ R_2 & \xrightarrow{i_2} & S_2 \end{array}$$

we say a partial function

$$f : R_1 \longrightarrow R_2$$

represents a partial function

$$g : S_1 \longrightarrow S_2$$

if for all $r_1 \in \mathrm{Dom}(i_1)$, $g(i_1(r_1)) = i_2(f(r_1))$ i.e. the above commutative diagram holds.

FIRST ATTEMPT

Alternatively, rather than expressing the diagram in terms of partial functions we could try to create f as a representation of g in terms of total functions.

Given $i_1 : R_1 \longrightarrow S_1$ consider

$$i_1^{-1} : S_1 \longrightarrow 2^{R_1} - \{\varphi\} \text{ where } i_1^{-1}(s_1) = \{ r_1 \in R \mid i_1(r_1) = s_1 \}$$

i.e. i_1^{-1} for any $s_1 \in S_1$ yields the set of all possible representations of s_1. Then $f : R_1 \longrightarrow R_2$ may also be extended to subsets of R_1 as follows $f : 2^{R_1} \longrightarrow 2^{R_2}$ is defined for each $X_1 \subseteq R_1$ as

$$f(X_1) = \{ f(r_1) \mid r_1 \in X_1 \} = X_2 \subseteq R_2$$

This extension allows the possibility that f may not be defined for some values in X_1

But we would be interested in ensuring that the following diagram holds

$$\begin{array}{ccc} S_1 & \xrightarrow{g} & S_2 \\ {\scriptstyle i_1^{-1}}\downarrow & & \downarrow{\scriptstyle i_2^{-1}} \\ 2^{R_1} - \{\varphi\} & \xrightarrow[f]{} \subseteq & 2^{R_2} - \{\varphi\} \end{array}$$

For any $s_1 \in S_1$, if $g(s_1) \in S_2$, we require that

(i) there is at least one representation $r_1 \in R_1$ such that $i_2(f(r_1)) = g(s_1)$

(ii) $r_1 \neq r_1' \wedge i_1(r_1) = i_1(r_2) = s_1 \wedge f(r_1), f(r_1') \in R_2 \Rightarrow i_2(f(r_1)) = i_2(f(r_1'))$

(iii) there may be representations of s_1 in R_1 for which $f(r_1)$ may not be defined

Def. A partial function $g: S_1 \longrightarrow S_2$ is **Turing-computable** if there exists a Turing-computable function $f: \Sigma^* \longrightarrow \Sigma^*$ on an alphabet Σ and interpretations i_1, i_2 with $i_1: \Sigma^* \longrightarrow S_1$ and $i_2: \Sigma^* \longrightarrow S_2$ such that

(i) for each $s_1 \in S_1$ and $f(s_1) = s_2 \in S_2$;

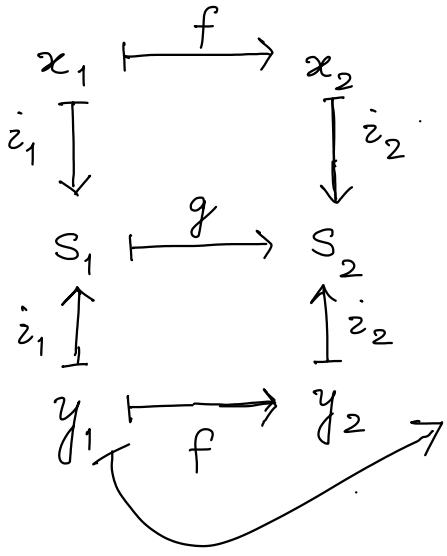
$$\exists x_1, x_2 \in \Sigma^* \Big[i_1(x_1) = s_1 \wedge i_2(x_2) = s_2 \wedge f(x_1) = x_2 \wedge$$
$$\forall y_1 \in \Sigma^* \big[x_1 \neq y_1 \wedge i_1(y_1) = s_1 \Longrightarrow f(y_1) \notin \Sigma^* \vee i_2(f(y_1)) = s_2 \big] \Big]$$

i.e. f is undefined for y_1

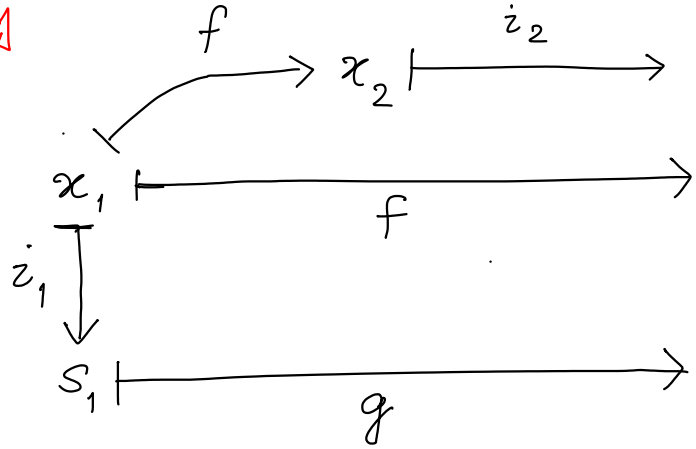
(ii) for each $s_1 \in S_1$ such that $g(s_1) \notin S_2$;

$$\forall x_1 \in \Sigma^* \Big[i_1(x_1) = s_1 \Longrightarrow f(x_1) \notin \Sigma^* \vee i_2(f(x_1)) \notin S_2 \Big]$$

i.e. $f(x_1) \in \Sigma^*$ but is "gibberish"



(i)



(ii)

for any partial function $g: A \longrightarrow B$ we write $g(a) \notin B$ when g is not defined for $a \in A$. Equivalently we may write $a \notin \mathrm{Dom}(g)$

A consequence of the above definition is that the implementation of any function of any arity is that of a unary function on Σ^* for the chosen alphabet.

Example. Consider addition on the naturals. We may denote it as a binary function $+ : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$. However for any pair of naturals $m, n \in \mathbb{N}$ we encode it in $\Sigma^* = \{0,1\}^*$ in unary with '0' as a separator between the components of the pair. Hence

$$i_1 : \Sigma^* \longrightarrow \mathbb{N} \times \mathbb{N}$$

in this case is defined as

Note that the string '0' denotes the ordered pair (0,0)

$$i_1(1^m 0 1^n) = (m, n)$$

and for all other patterns of strings $x \in \Sigma^* - \mathcal{L}(1^*01^*)$ we have $i_1(x)$ is undefined. Similarly i_2 for the result is simply defined as

ε denotes the natural number 0.

$$i_2(y) = \begin{cases} p & \text{if } y = 1^p \in 1^* \\ \text{undefined} & \text{otherwise} \end{cases}$$

With above representations the partial function $\hat{+} : \Sigma^* \to \Sigma^*$ that we require is given by

$$\hat{+}(x) = \begin{cases} 1^{m+n} & \text{if } x = 1^m 0 1^n \text{ for all } m, n \geq 0 \\ \text{undefined otherwise} \end{cases}$$

Any Turing machine $T_{\hat{+}}$ which implements $\hat{+}$ is a correct implementation of addition on the naturals.