2.2 Uncomputability; Goedel Theorem

Universal and Complete Functions

Notations: Let us choose a special mark and after its k-th occurrence, break any string x into $\text{Prefix}_k(x)$ and $\text{Suffix}_k(x)$. Let $f^+(x)$ be $f(\text{Prefix}_k(x)\ x)$ and $f^-(x)$ be $f(\text{Suffix}_k(x))$. We say u ***k-simulates*** f iff for some $p = \text{Prefix}_k(p)$ and all s, $u(ps) = f(s)$. The prefix can be intuitively viewed as a program which simulating function u applies to the suffix (input). We also consider a symmetric variant of relation "k-simulate" which makes some proofs easier. Namely, u ***k-intersects*** f iff $u(ps) = f(ps)$ for some $\text{prefix}_k\ p$ and all s. E.g., length preserving functions can intersect but cannot simulate one another.

We call ***universal*** for a class F, any u which k-simulates all functions in F for a fixed k. When F contains f^-, f^+ for each $f \in F$, universality is equivalent to (or implies, if only $f^+ \in F$) ***completeness***: u k-intersects all $f \in F$. Indeed, u k-simulates f iff it k-intersects f^-; u $2k$-intersects f if it k-simulates f^+.

Exercise: Describe **explicitly** a function, **complete** for the class of all *linear* (e.g., $5x$ or $23x$) functions.

A ***negation*** of a (partial or total) function f is the total predicate $\neg f$ which yields 1 iff $f(x)$=0 and yields 0 otherwise. Obviously, no closed under negation class of functions contains a complete one. So, there is no universal function in the class of all (computable or not) predicates. This is the well known Cantor Theorem that the set of all sets of strings (as well as the sets of all functions, reals etc.) is not countable.

Goedel's Theorem

There is no complete function among the **total** computable ones, as this class is closed under negation. So the universal in R function u (and $u_2 = (u \bmod 2)$) has no total computable extensions.

Formal proof systems are computable functions $A(P)$ which check if P is an acceptable proof and output the proven statement. $\vdash s$ means $s = A(P)$ for some P. A is ***rich*** iff it allows computable translations s_x of statements "$u_2(x) = 0$", provable whenever true, and refutable ($\vdash \neg s_x$), whenever $u_2(x) = 1$. A is ***consistent*** iff **at most** one of any such pair $s_x, \neg s_x$ is provable, and ***complete*** iff **at least** one of them always (even when $u(x)$ diverges) is. Rich consistent and complete formal systems cannot exist, since they would provide an obvious total extension u_A of u_2 (by exhaustive search for P to prove or refute s_x). This is the famous Goedel's Theorem – one of the shocking surprises of the 20th century science. (Here A is any extension of the formal Peano Arithmetic; we skip the details of its formalization and proof of richness.)[3]

Computable Functions. Another byproduct is that the Halting (of $u(x)$) Problem would yield a total extension of u and, thus, is not computable. This is the source of many other uncomputability results. Another source is an elegant ***Fixed Point*** Theorem by S. Kleene: any total computable transformation A of programs (prefixes) maps some program into an equivalent one. Indeed, the complete/universal $u(ps)$ intersects computable $u(A(p)s)$. This implies, e.g., Rice theorem: the only computable invariant (i.e. the same on programs computing the same functions) property of programs is constant (**exercise**).

Computable (partial and total) functions are also called recursive (due to an alternative definition). Their ranges (and, equivalently, domains) are called (computably) ***enumerable*** or ***c.e.*** sets. An c.e. set with an c.e. complement is called computable (as is its yes/no characteristic function) or ***decidable***. A function is computable iff its graph is c.e. An c.e. graph of a total function is computable. Each infinite c.e. set is the range of an injective total computable function ("enumerating" it, hence the name c.e.).

We can reduce membership problem of a set A to the one of a set B by finding a computable function f s.t. $x \in A \iff f(x) \in B$. Then A is called ***m-*** (or ***many-to-1-***) ***reducible*** to B. A more complex ***Turing*** reduction is given by an algorithm which, starting from input x, interacts with B by generating strings s and receiving answers to $s \in ?B$ questions. Eventually it stops and tells if $x \in A$. c.e. sets (like Halting Problem) to which all c.e. sets can be m-reduced are called c.e.-complete. One can show a set c.e.-complete (and, thus, undecidable) by reducing the Halting Problem to it. So Ju.Matijasevich proved c.e.-completeness of Diophantine Equations Problem: given a multivariate polynomial of degree 4 with integer coefficients, find if it has integer roots. The above (and related) concepts and facts are broadly used in Theory of Algorithms and should be learned from any standard text, e.g., [Rogers 67].

[3] A closer look at this proof reveals another famous Goedel theorem: Consistency C of A (expressible in A as divergence of the search for contradictions) is itself an example of unprovable $\neg s_x$. Indeed, u_2 intersects $1-u_A$ for some prefix a. C implies that u_A extends u_2 and, thus, $u_2(a), u_A(a)$ both diverge. So, $C \Rightarrow \neg s_a$. This proof can be formalized in Peano Arithmetic, thus $\vdash C \Rightarrow \vdash \neg s_a$. But $\vdash \neg s_a$ implies $u_A(a)$ converges, so $\vdash C$ contradicts C: Consistency of A is provable in A if and only if false !