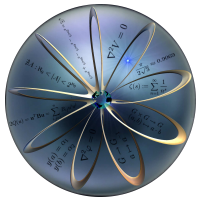

planetmath.org

Math for the people, by the people.

Markov algorithm

Canonical name	MarkovAlgorithm
Date of creation	2013-03-22 18:57:50
Last modified on	2013-03-22 18:57:50
Owner	CWoo (3771)
Last modified by	CWoo (3771)
Numerical id	13
Author	CWoo (3771)
Entry type	Definition
Classification	msc 03D10
Defines	Markov-computable

Definition

A *Markov algorithm* is a variant of a rewriting system, invented by mathematician Andrey Andreevich Markov Jr. in 1960. Like a rewriting system, a Markov algorithm consists of an alphabet and a set of productions. Furthermore, rewriting is done by applying productions one at a time, if any. However, unlike a rewriting system, applications of productions are regulated, in the following sense:

- P is ordered so that, among all applicable productions in a single rewriting step, the first applicable one $x \to y$ must be used;
- among all occurrences where rewriting can take place with respect to $x \to y$, the leftmost occurrence must be applied.

Formally, a *Markov algorithm* is a quadruple $\mathscr{M} = (\Sigma, P, F, T)$, where

1. Σ is an alphabet;
2. P is a non-empty finite ordered set (ordered by, say, $\leq$) of pairs of words over Σ, whose elements are called *productions*, and are usually written $x \to y$ rather than (x, y);
3. F is a subset of P, whose elements are called the *final productions*; and
4. T is a subset of Σ, called the terminal alphabet.

Rewriting Process

Next, we describe the rewriting process for $\mathscr{M}$. A production $x \to y$ is *applicable* to a pair (u, v) of words over Σ, if there are two words p, q such that $u = pxq$ and $v = pyq$.

A binary relation $\Rightarrow$ on Σ^* called the *rewriting step relation*, is defined as follows: $u \Rightarrow v$ iff there is a production $x \to y$ such that

1. $u = pxq$ and $v = pyq$ for some words p, q over Σ,
2. if $(r, s) < (x, y)$, then $r \to s$ is not applicable to (u, v),
3. if $u = p'xq'$ and $v = p'yq'$, then px is a prefix of $p'x$.

The first condition ensures that there is a production ($x \to y$ in this case) applicable to (u, v), the second condition says that $x \to y$ is the first available production that can be applied, and the last condition says that the occurrence of x in u is the leftmost.

By the definition above, every rewriting step $u \Rightarrow v$ determines a unique production $x \to y$, called the *associated production* of $u \Rightarrow v$.

A word u over Σ is said to be *terminal* if there are no words v such that $u \Rightarrow v$.

A rewriting step is called *final* if its associated production is final (in F).

Computation

Take the reflexive transitive closure $\Rightarrow^*$ of $\Rightarrow$. If $u \Rightarrow^* v$, we say that v can be *computed* from u, or that v is a *computation* from u.

Given any word u over Σ, we may iteratively apply rewriting steps to u using the methods described above. Three scenarios may emerge:

- The process terminates: $u \Rightarrow^* v$, where v is a terminal word;
- The process reaches a final rewriting step: $u \Rightarrow^* v$, where the last rewriting step $u_{n-1} \Rightarrow u_n = v$ is final; or
- The process never reaches any final rewriting step, and goes on indefinitely.

If the third scenario occurs, we say that $\mathscr{M}$ *loops* on u. Otherwise, $\mathscr{M}$ *halts* on u. In any case, we get a unique sequence

$$u = u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n \Rightarrow \cdots$$

of rewriting steps $u_i \to u_{i+1}$. In the first scenario, the sequence is finite, and in the third scenario, the sequence is infinite. In the second scenario, the sequence may be finite or infinite, depending if the rewriting is to be continued after a final rewriting step is reached (if v is not a terminal word, then rewriting can continue). Let us make the rule:

> when a final rewriting is reached, no further rewriting is to be done.

Thus, the sequence is finite iff $\mathscr{M}$ halts on u. When $\mathscr{M}$ halts on u, the finite sequence

$$u = u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_n = v$$

produces a unique word v. The word v is said to be *the word computed by* $\mathscr{M}$ from u. Notice that v is either a terminal word, or is computed from a final production $u_{n-1} \to v$. In either case, no earlier productions $u_i \to u_{i+1}$ are final.

Thus, one can think of a computation by a Markov algorithm as a partial function

$$m_{\mathscr{M}} : \Sigma^* \to \Sigma^*,$$

where $m_{\mathscr{M}}(u)$ is defined iff $\mathscr{M}$ halts on u, and the value $m_{\mathscr{M}}(u)$ is set to *the unique word* v computed by $\mathscr{M}$ from u.

Language Acceptor

$\mathscr{M}$ can be thought of as a language acceptor, which is the purpose of the terminal alphabet T: the set

$$L(\mathscr{M}) := \{u \in T^* \mid m_{\mathscr{M}}(u) = \lambda\}$$

is called the *language accepted by* $\mathscr{M}$. The partial function $m_{\mathscr{M}}$ is defined in the previous section.

Remark. It turns out that a Markov algorithm is just another form of Turing Machine. One can show that a language is recursively enumerable iff it can be accepted by a Markov algorithm.

Equivalence to a Turing machine can be restated in terms of functional computability. Before formalizing this notion, we need to first encode tuples of natural numbers by words. Suppose $(n_1, \ldots, n_m)$ is an m-tuple of natural numbers. Set

$$E(n_1, \ldots, n_m) := ab^{n_1}ab^{n_2}a \cdots ab^{n_m}a,$$

a word over the alphabet $\{a, b\}$. If non-negative integers are allowed instead, we may use the word $E(n_1 + 1, \ldots, n_m + 1)$ instead.

We say an m-ary number-theoretic partial function $f : \mathbb{N}^m \to \mathbb{N}$ is *Markov-computable* if there is a Markov algorithm $\mathscr{M}$ such that $f(n_1, \ldots, n_m)$ is defined iff $m_{\mathscr{M}}(E(n_1, \ldots, n_m))$ is defined, and is equal to $E(f(n_1, \ldots, n_m))$.

It can be shown that a partial function is Turing-computable iff it is Markov-computable.

References

[1] A. Salomaa *Computation and Automata, Encyclopedia of Mathematics and Its Applications, Vol. 25*. Cambridge (1985).

[2] N. Cutland, *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, (1980).

[3] J. D. Monk, *Mathematical Logic*, Springer, New York (1976).