planetmath.org

Math for the people, by the people.

state-output machine

Canonical name	StateoutputMachine
Date of creation	2013-03-22 18:59:49
Last modified on	2013-03-22 18:59:49
Owner	CWoo (3771)
Last modified by	CWoo (3771)
Numerical id	14
Author	CWoo (3771)
Entry type	Definition
Classification	msc 03D05
Classification	msc 68Q45
Related topic	GeneralizedSequentialMachine
Related topic	Semiautomaton
Defines	sequential machine
Defines	complete machine
Defines	incomplete machine

Definition

A *state-output machine* can be thought of as state machine with an output feature: when a word is fed into the machine as input, the machine goes through a series of internal "states" where certain translations take place, and finally a set of words are produced as outputs.

Formally, a *state-output machine* M is a five-tuple $(S, \Sigma, \Delta, \delta, \lambda)$ where

1. (S, Σ, δ) is a state machine (or semiautomaton),
2. Δ is a non-empty set whose elements are called *output symbols*, and
3. $\lambda : S \times \Sigma \to P(\Delta)$ is a function called the *output function*.

The sets S, Σ, and Δ are generally considered to be finite. In the literature, a finite state-output machine is also known as *transducer*.

Note that there is no restrictions on the sizes of $\lambda(s, a)$ and $\delta(s, a)$. Various classifications based on the cardinalities of $\lambda(s, a)$ and $\delta(s, a)$ are possible: for all $(s, a) \in S \times \Sigma$,

- M is *complete* if $|\lambda(s, a)| \geq 1$ and $|\delta(s, a)| \geq 1$; otherwise, it is *incomplete*;
- M is *sequential* if $|\lambda(s, a)| \leq 1$ and $|\delta(s, a)| \leq 1$.

Both δ and λ can be extended so its first component takes on a set T of states:

$$\delta(T, a) := \bigcup\{\delta(t, a) \mid t \in T\} \qquad \text{and} \qquad \lambda(T, a) := \bigcup\{\lambda(t, a) \mid t \in T\}.$$

Note that $\delta(\varnothing, a) = \lambda(\varnothing, a) = \varnothing$ for any input symbol $a \in \Sigma$.

Words as Input

The transition and the output functions of a state-output machine M are defined to work only over individual symbols in Σ as inputs. However, finite strings of symbols over Σ, or words, are usually fed to the M, instead of individual symbols. Therefore, we would like modify δ and λ in order to handle finite strings as well.

Extending δ. When a machine M receives an input word u, it reads u one symbol at a time, starting from the left, until the last symbol is read. After reading each symbol, the machine goes into a next state, dictated by the transition function δ. If M is at state s upon receiving u, we define a next state as a state that M enters after reading the last symbol of u.

Based on the above discussion, we are ready to extend δ so it takes on words over Σ. This is done inductively:

- $\delta'(s,\epsilon) := \{s\}$, where ϵ is the empty word, and s is any state;
- $\delta'(s,ua) := \delta(\delta'(s,u),a)$, where $a \in \Sigma$ and $u \in \Sigma^*$.

It is easy to see that $\delta'(s,uv) = \delta'(\delta'(s,u),v)$.

Extending λ. There are in general two ways to view output(s) for a given input word:

1. The first, more common, approach, is to view outputs as being produced after the last symbol of the input word is processed:
   - $\lambda'(s,\epsilon) := \varnothing$, and
   - $\lambda'(s,ua) := \lambda(\delta'(s,u),a)$, where u is a word over Σ.

   If λ does not depend on input symbols, say $\lambda(s,a) = \beta(s)$ for all $(s,a) \in S \times \Sigma$, the above definition may be modified so that non-empty output(s) may be produced by the empty input word ϵ:

   - $\lambda'(s,u) := \beta(\delta'(s,u))$, where u is any word over Σ.

   It is easy to see that $\lambda(s,\epsilon) = \beta(s)$. Note that this is not a true extension of the original output function, because the new output function now depends on inputs.

2. Alternatively, outputs may be produced each time a transition occurs. In other words, outputs are words over Δ. Thus, outputs are inductively as follows:
   - $\lambda'(s,\epsilon) := \{\epsilon\}$, where ϵ is the empty word, and
   - $\lambda'(s,ua) := \lambda'(s,u)\lambda(\delta'(s,u),a)$, where $a \in \Sigma$ and $u \in \Sigma^*$.

When there is no confusion, we may continue to denote λ and δ as the extensions of the original next-state and output functions.

Given M, define an *input configuration* as a pair (s, u) for some $s \in S$ and $u \in \Sigma^*$, and an *output configuration* as a pair (t, v) for some $t \in S$ and $v \in \Delta^*$. The set of output configurations for a given input configuration (s, u) is given by $\delta(s, u) \times \lambda(s, u)$.

Generator and Acceptor

One may treat a state-output machine $M = (S, \Sigma, \Delta, \delta, \lambda)$ as either a *language generator* or a *language acceptor*. The idea is that a set of states and a set of words need to be specified as initial conditions, so that words can either be generated or accepted from these initial conditions. The way this works is as follows:

M **as a generator.** Fix a non-empty set $I \subseteq S$ of *starting states*, and a non-empty set $G \subseteq \Sigma^*$. The triple (M, I, G) is called a *generator*. A string $b \in \Delta^*$ is *generated by* (M, I, G) if $b \in \lambda(s, a)$ for some $(s, a) \in I \times G$. The set of all strings generated by (M, I, G) is also denoted by $L(M, I, G)$.

A typical example of a generator is a Post system: a state machine where the output alphabet is the input alphabet, and the set of states and the state function is suppressed (S may be taken as a singleton).

M **as an acceptor.** Dually, fix a non-empty set $F \subseteq S$ called the *final states*, and a non-empty set $A \subseteq \Delta^*$. The triple (M, F, A) is called an *acceptor*. A string $a \in \Sigma^*$ is said to be accepted by (M, F, A) if $\delta(s, a) \in F$ and $\lambda(s, a) \in A$ for some state $s \in S$. The set of all strings accepted by (M, F, A) is denoted by $L(M, F, A)$.

A typical example of an acceptor is an automaton: a state machine where the output alphabet and the output function are not essential (Δ^* may be taken as a singleton).

Remark. Observe that the functions δ and λ can be combined to form a single function $\tau : S \times \Sigma \to P(S) \times P(\Delta)$ such that $\tau = (\delta, \lambda)$. One can generalize this so that τ is a function from $S \times \Sigma$ to $P(S \times \Delta)$, or more generally, to $P(S \times \Delta^*)$. The resulting construct is commonly known as a *generalized sequential machine*.

References

[1] S. Ginsburg, *An Introduction to Mathematical Machine Theory*, Addision-Wesley, (1962).

[2] M. Arbib, *Algebraic Theory of Machines, Languages, and Semigroups*, Academic Press, (1968).

[3] J. Hartmanis, R.E. Stearns, *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, (1966).