AUTOMATON PROGRAMS AND REGULAR FUNCTIONAL EXPRESSIONS—ON AN EXTENSION OF DERIVATIVES

Nishizawa, Teruyasu
Faculty of Economics, Niigata University

西澤，輝泰
新潟大学経済学部

http://hdl.handle.net/2324/13362
AUTOMATON PROGRAMS AND REGULAR FUNCTIONAL EXPRESSIONS—ON AN EXTENSION OF DERIVATIVES

By

Teruyasu NISHIZAWA*

Abstract

A method for constructing a finite automaton by taking derivatives of a regular set is a method for synthesizing a recursive program. We extend the method to synthesize more general programs than finite automata by extending the notion of derivatives. The programs synthesizable by this method are called automaton programs and the predicates computable by these programs are characterized by regular functional expressions.

Let $P$ be a one-place predicate over a set $W$, $f$ be a partial function of $W$ to $W$ and $D(f)$ denote the domain of $f$. When it holds that $f(x) = f(y)$ implies $p(x) \equiv p(y)$ for all $x, y$ in $W$, $P$ has a derivative $\partial_f P$ by $f$ which is defined by

$$(\partial_f P)(x) \leftrightarrow \exists y \in D(f) \{ p(y) \land f(y) = x \}.$$ 

We can construct an automaton program computing a predicate by taking these extended derivatives of the predicate if it has a finite derivative-closure.

1. Introduction

A derivative $\partial x S$ of a set $S$ of words over an alphabet $\Sigma$ by a word $x$ over the $\Sigma$ is the set

$$\{ y \in \Sigma^* | xy \in S \},$$

where $\Sigma^*$ denotes the set of all words over $\Sigma$.

As for derivatives of a subset $S$ of $\Sigma^*$, the following equation trivially holds:

$$S = \bigcup_{\sigma \in \Sigma} \sigma \partial_{\sigma} S \cup E_S,$$

where, letting $\varepsilon$ denote the empty word and $\phi$ denote the empty set, $E_S$ is the set defined by

$$E_S = \text{if } \varepsilon \text{ is in } S \text{ then } \{ \varepsilon \} \text{ else } \phi.$$

Let $P$ and $P_{\sigma}$ ($\sigma \in \Sigma$) be one-place predicates over $\Sigma^*$ such that

$$P(x) \iff x \in S$$

$$P_{\sigma}(x) \iff x \in \partial_{\sigma} S.$$
Then the above equation can be rewritten as
\[ P(x) \leftrightarrow (\text{null}(x) \land t) \lor \bigvee_{\sigma \in \Sigma} (\text{top}(x) = \sigma \land P_{\sigma}(\text{tail}(x))), \]
where \( \text{null}(x) \) represents \( x = \epsilon \), \( t \) represents \( \epsilon \in S \), \( \text{top}(x) \) is the first (i.e. the leftmost) symbol of \( x \) and \( \text{tail}(x) \) is the word such that \( x = \text{top}(x) \text{tail}(x) \).

If we put \( X' = \{\alpha_1, 0, 2, \ldots, 6n\} \) then the above equivalence is rewritten as
\[ P(x) \leftrightarrow \text{if null}(x) \text{ then } t \text{ else } \]
\[ \text{if top}(x) = \alpha_1 \text{ then } P_{\alpha_1}(\text{tail}(x)) \text{ else } \]
\[ \text{if top}(x) = \alpha_2 \text{ then } P_{\alpha_2}(\text{tail}(x)) \text{ else } \]
\[ \ldots \]
\[ \text{if top}(x) = \alpha_n \text{ then } P_{\alpha_n}(\text{tail}(x)) \text{ else } \]
\[ P_{\alpha_n}(\text{tail}(x)) . \]

If there are only finitely many derivatives of \( S \) then \( S \) is a regular set and we can construct an automaton \( A \) recognizing \( S \), where the state set of \( A \) is the set of all derivatives of \( S \) and the state-transition function \( \delta \) of \( A \) is given by \( \delta(\partial_x S, \sigma) = \partial_{\sigma \delta} S \) for all \( \sigma \in \Sigma \) and for all derivatives \( \partial_x S \) of \( S \).

Defining \( P_x \) as the predicate representing membership of \( \partial_x S \) (that is, \( P_x(y) \leftrightarrow y \in \partial_x S \)) we have
\[ P_x(y) \leftrightarrow \text{if null}(y) \text{ then } t_{\partial_x S} \text{ else } \]
\[ \text{if top}(y) = \alpha_1 \text{ then } P_{\alpha_1}(\text{tail}(y)) \text{ else } \]
\[ \ldots \]
\[ \text{if top}(y) = \alpha_n \text{ then } P_{\alpha_n}(\text{tail}(y)) \text{ else } \]
\[ P_{\alpha_n}(\text{tail}(y)) \]

for each derivative \( \partial_x S \) of \( S \).

This system of equivalences for all derivatives \( \partial_x S \) of \( S \) is a representation of the automaton \( A \) and is a recursive program computing the predicate \( P \).

Thus the method for constructing automata by taking derivatives is the most classic method for program synthesis, although people may not have so recognized.

We consider that this program-synthesizing method is essential and very significant, and believe that a good extension of the concept of derivative will contribute in a large extent to the theory of automatic program synthesis.

Indeed, our attempt to apply an intuitive notion of extended derivative for making a pure- LISP program synthesizer seems to obtain a good result [2].

To explain our view more clearly, we show a simple example.

**Example 1.** Let \( \Sigma = \{0, 1\} \) and, for each \( x \) in \( \Sigma^* \), let \( \langle x \rangle \) be the number represented by \( x \) in the usual diadic notation, that is, \( \langle \epsilon \rangle = 0 \) and \( \langle x \sigma \rangle = 2 \langle x \rangle + \sigma \) for each \( \sigma \in \Sigma \) and \( x \in \Sigma^* \).

Consider a predicate \( P \) such that
\[ P(x) \leftrightarrow [\langle x \rangle \equiv 0(\text{mod } 3)]. \]

To make a program to compute \( P(x) \), we take derivatives of \( P(x) \) as follows.
First we take $P_0$ and $P_1$ such that

$$P(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{true} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P_0(\text{tail}(x)) \text{ else } P_1(\text{tail}(x)).$$

Then we have

$$P_0(x) \leftrightarrow P(0x) \leftrightarrow P(x)$$

(then, $P_0 = P$) and

$$P_1(x) \leftrightarrow P(1x).$$

Next we take $P_{10}$ and $P_{11}$ such that

$$P_1(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{false} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P_{10}(\text{tail}(x)) \text{ else } P_1(\text{tail}(x)).$$

Then we have

$$P_{10}(x) \leftrightarrow P_1(0x) \leftrightarrow P(10x) \text{ and }$$

$$P_{11}(x) \leftrightarrow P_1(1x) \leftrightarrow P(11x) \leftrightarrow P(x)$$

(then, $P_{11} = P_1$)

Finally we take $P_{100}$ and $P_{101}$ such that

$$P_{10}(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{false} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P_{100}(\text{tail}(x)) \text{ else } P_{10}(\text{tail}(x)).$$

Then we have

$$P_{100}(x) \leftrightarrow P_{10}(0x) \leftrightarrow P(100x) \leftrightarrow P(1x) \leftrightarrow P_1(x)$$

and

$$P_{101}(x) \leftrightarrow P_{10}(1x) \leftrightarrow P(101x) \leftrightarrow P(10x) \leftrightarrow P_{10}(x)$$

(then, $P_{100} = P_{10}$ and $P_{101} = P_{10}$)

Hence, $P$, $P_1$ and $P_{10}$ are the all derivatives of $P$ and we obtain the following program computing $P$.

$$P(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{true} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P(\text{tail}(x)) \text{ else } P_1(\text{tail}(x))$$

$$P_1(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{false} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P_{10}(\text{tail}(x)) \text{ else } P(\text{tail}(x))$$

$$P_{10}(x) \leftrightarrow \text{if } \text{null}(x) \text{ then } \text{false} \text{ else }$$

$$\text{if } \text{top}(x) = 0 \text{ then } P_1(\text{tail}(x)) \text{ else } P_{10}(\text{tail}(x)).$$

In this note, we give a formal extension of the notion of derivatives and give a characterization of predicates computed by programs which can be synthesized by taking extended derivatives of the predicates.
2. Derivatives of Predicates by Partial Functions and Automaton Programs

Let \( W \) be an arbitrary nonempty set. For a partial function \( f \) of \( W \) to \( W \), let \( D(f) \) denote the domain of \( f \), that is,

\[
D(f) = \{ x \in W | f(x) \text{ is defined} \}
\]

**Definition 1.** Let \( P \) be a one-place predicate over \( W \) and \( f \) be a partial function of \( W \) to \( W \). We say that \( P \) has a derivative by \( f \) if and only if, for all \( x, y \) in \( D(f) \), \( f(x) = f(y) \) implies \( P(x) \equiv P(y) \). When \( P \) has a derivative by \( f \), the derivative \( \partial_f P \) by \( f \) of \( P \) is defined by

\[
\partial_f P(x) \iff (\exists y \in D(f))[P(y) \land f(y) = x].
\]

If \( P \) has a derivative by \( f \) then, for each \( x \in D(f) \), we have

\[
P(x) \iff \partial_f P(f(x))
\]

because of

\[
x \in D(f) \land \partial_f P(f(x))
\]

\[
\iff x \in D(f) \land (\exists y \in D(f))[P(y) \land f(y) = f(x)]
\]

\[
\iff x \in D(f) \land P(x).
\]

Let \( F \) be a family of partial functions of \( W \) to \( W \).

If \( P \) has a derivative by each \( f \in F \) then the following holds:

\[
P(x) \land \bigvee_{f \in F} x \in D(f)
\]

\[
\iff \bigvee_{f \in F} (x \in D(f) \land \partial_f P(f(x))).
\]

Let \( \lambda_f(x) \) denote the predicate

\[
\forall_{f \in F} x \in D(f).
\]

If, for each \( x \) such that \( \lambda_f(x), P(x) \) has a constant value \( t_f \) (true or false), we have the equivalence

\[
P(x) \iff \bigvee_{f \in F} (x \in D(f) \land \partial_f P(f(x))) \lor (\lambda_f(x) \land t_f).
\]

**Definition 2.** For a one-place predicate \( P \) over a set \( W \), a system of predicates \( P_0, P_1, \ldots, P_n \) and finite families \( F_0, F_1, \ldots, F_n \) of partial functions \( W \to W \) is called a finite derivative-closure of \( P \) if the following conditions hold:

1. \( P = P_0 \),
2. For each \( P_i, P_i(x) \) has a constant value \( t_i \) for all \( x \) such that \( \lambda_{F_i}(x) \),
3. For each \( P_i \) and for each \( f \in F_i, P_i \) has a derivative \( \partial_f P_i \) by \( f \) which coincides with one of \( P_0, P_1, \ldots, P_n \).

If \( P \) has a finite derivative-closure \( \{ P_0, P_1, \ldots, P_n; F_0, F_1, \ldots, F_n \} \) then the following system of equivalences is considered as a (nondeterministic) program computing \( P \).

\[
\bigwedge_{f \in F_i} P_i(x) \iff \bigvee_{f \in F_i} (x \in D(f) \land \partial_f P_i(f(x))) \lor (\lambda_{F_i}(x) \land t_i)
\]
Automaton programs and regular functional expressions

This program becomes deterministic if, for each $F_i$, all $f \in F_i$ have mutually disjoint domains, and it becomes a terminating program if $W$ is a well-founded set with an order $\prec$ and all $f \in F_i$ for each $F_i$ satisfy the decreasing condition

$$f(x) \leq x \quad \text{for all } x \in D(f).$$

Hereafter, a partial function $f$ of a well-founded set to the same set is called a decreasing function if $f$ satisfies the decreasing condition.

Now, the above program computing $P$ can be represented by the following automaton:

1. the state set is $\{P_0, P_1, \ldots, P_n\}$,
2. the initial state is $P_0$,
3. the final states are $P_i$ such that $t_i = \text{true}$,
4. the state transition is represented by a finite directed labeled graph whose nodes are the states $P_0, P_1, \ldots, P_n$ such that an edge from $P_i$ to $P_j$ with a label $f$ represents the relation $\partial_j P_i = P_j$. (Thus, for each $f \in F_i$, an edge labeled by $f$ issues from $P_i$.)

**Definition 3.** An automaton program scheme is a finite directed graph with labeled edges such that one node is specified as an initial state and some nodes are specified as final states. In the scheme, each node called a state.

An interpretation of the scheme is an association of each edge label with a partial function of a nonempty set $W$ to $W$. The $W$ is called the support of the interpretation.

An interpreted (i.e. given an interpretation) automaton program scheme is called an automaton program.

A computation of an automaton program is a sequence

$$(q_0, x_0), (q_1, x_1), \ldots, (q_m, x_m)$$

such that each $x_i$ is in the support $W$ of the interpretation, each $q_i$ is a state and, for each $i = 0, 1, \ldots, m-1$, there exists an edge from $q_i$ to $q_{i+1}$ whose label is interpreted under the given interpretation as a partial function $f : W \to W$ satisfying

$$x_i \in D(f) \land f(x_i) = x_{i+1}.$$

The computation is said to be terminating if there exists no edge issuing from $q_m$ whose label is interpreted as an $f$ such that $x_m \in D(f)$.

The terminating computation is said to be accepting if the $q_m$ is a final state.

An element $x$ of $W$ is said to be accepted by starting from $q$ if there exists an accepting computation starting from $(q, x)$.

The recognized set of the automaton program is the set of all elements $x$ in $W$ accepted by starting from the initial state, and a predicate whose extension is the recognized set is said to be computed by the automaton program.

**Notation.** For convenience, we use the following notation. For a considered given automaton program, $Q$ is the set of all states, $q_0$ is the initial state, $T$ is the set of all final states and $E_q$ is the set of all edges issuing from $q$ for each state $q$.

For each edge $e$, $s_e$ is the state to which $e$ leads and $f_e$ is the partial function associated with the label of $e$ in the considered interpretation.

The support of the interpretation is denoted by $W$. $\lambda_e(x)$ stands for $\forall e \in E_q, x \in D(f_e)$. 


By the above definitions and notations, the following proposition obviously holds.

**Proposition 1.** Consider an automaton program. For each \( q \in Q \), let \( P_q \) be the predicate whose extension is the set of all \( x \) in \( W \) accepted by starting from the \( q \). Then the following system of equivalences holds:

\[
\forall q \in Q \left( P_q(x) \iff \bigvee_{e \in E_q} (x \in D(f_e) \land P_e(f_e(x))) \land (x \in L_q) \land q \in T \right).
\]

The predicate computed by the automaton program is \( P_{q_0} \) and, for each \( e \in E_q \), the equation \( \partial f_e P_q = P_{s_e} \) holds.

Hence the following proposition also holds.

**Proposition 2.** Let \( F \) be a family of partial functions of a nonempty set \( W \) to \( W \). A one-place predicate \( P \) is computed by an automaton program whose interpretation consists of partial functions in \( F \) if and only if \( P \) has a finite derivative-closure \( \{ P_0, P_1, \ldots, P_n \} = \{ F_0, F_1, \ldots, F_n \} \) such that each \( F_i \) is a subfamily of \( F \).

### 3. Regular Functional Expressions

In this section, we restrict supports \( W \) of interpretations in automaton programs to be well-founded sets and restrict partial functions \( f \) constituting the interpretations to be decreasing, that is, \( f(x) \leq x \) holds for all \( x \in D(f) \).

Also we assume that, for each state \( q \) and for each non-minimal element \( x \) in \( W \), there exists an edge \( e \) issuing from \( q \) such that \( x \in D(f_e) \). (This assumption brings no loss of generality.)

Now for each edge \( e \) of automaton programs, we denote the function \( f_e^{-1} \) of \( 2^W \) to \( 2^W \) by \( g_e \), that is,

\[
g_e(X) = \{ x \in D(f_e) | f_e(x) \in X \}
\]

for every subset \( X \) of \( W \).

\( g_e \) is strictly additive and increasing, where a function \( g \) of \( 2^W \) to \( 2^W \) is said to be strictly additive if \( g \) satisfies

\[
g(\emptyset) = \emptyset \quad \text{and} \quad g(X) = \bigcup_{x \in X} g(\{x\}),
\]

and said to be increasing if, for all \( y \) in \( g(\{x\}) \), \( x \lesssim y \) holds.

For a considered automaton program, let \( U_q \) be the set of all elements of \( W \) accepted by starting from a state \( q \) and let \( M \) be the set of all minimal element of \( W \).

Then the following system of set equations holds:

\[
\forall q \in Q \left( U_q = \bigcup_{e \in E_q} g_e(U_{s_e}) \cup L_q \right)
\]

where \( L_q = \text{if } q \in T \text{ then } M \text{ else } \emptyset \).

Indeed, for each state \( q \), under the same notation as in the proposition 1, we obtain

\[
x \in \bigcup_{e \in E_q} g_e(U_{s_e}) \cup L_q
\]

\[
\quad \leftrightarrow (\exists e \in E_q) x \in g_e(U_{s_e}) \land x \in L_q
\]

\[
\quad \leftrightarrow (\exists e \in E_q) [x \in D(f_e) \land f_e(x) \in U_{s_e}] \land (x \in M \land q \in T)
\]
Automaton programs and regular functional expressions

\[ \forall e \in E, \exists q \in q \quad (x \in D(f_e) \land P_{q_e}(f_e(x)) \lor (\lambda_q(x) \land q \in T)) \]

\[ P_q(x) \]

\[ x \in U_q. \]

Now, putting \( Q = \{ q_0, q_1, \ldots, q_n \} \), \( g_{i\cdot j}(X) = \bigcup \{ g_{e\cdot j}(X) \mid e \in E_{q_i} \land q_j = s_e \} \) for each \( i, j = 0, 1, \ldots, n \), \( U_i = U_{q_i} \) and \( L_i = L_{q_i} \), the above system of set equations is rewritten as in the following form:

\[ \bigcup_{i=0}^{n} \bigcup_{j=0}^{n} \{ U_i = \bigcup_{j=0}^{n} g_{i\cdot j}(U_j) \cup L_i \} \]

Here, it should be noted that the all \( g_{i\cdot j} \)'s are strictly additive and increasing.

**Definition 3.** Let \( g \) and \( g' \) be functions of \( 2^W \) to \( 2^W \). We define functions \( g + g' \), \( g \cdot g' \), \( g^m \) and \( g^* \) as follows:

\[ (g + g')(X) = g(X) \cup g'(X), \]

\[ (g \cdot g')(X) = g(g'(X)), \]

\[ g^0 = I \quad \text{(the identity function)}, \]

\[ g^{m+1} = g \cdot g^m, \]

\[ g^*(X) = \bigcup_{m=0}^{\infty} g^m(X). \]

(It should be noted that if \( g \) and \( g' \) are strictly additive and increasing then so are the \( g + g' \), \( g \cdot g' \), \( g' \cdot g^* \) and \( g^* \cdot g' \).)

Let \( G \) be a family of functions of \( 2^W \) to \( 2^W \). A regular functional expression over \( G \) is an expression obtained by finite number of applications of operations \( + \), \( \cdot \) and \( * \) to functions in \( G \), the identity function \( I \) and \( \phi \) (the constant valued function whose value is constantly the empty set.)

**Lemma 1.** For a strictly additive and increasing function \( g \) of \( 2^W \) to \( 2^W \) and for a subset of \( L \) of \( W \), there exists a unique subset \( X \) of \( W \) satisfying the set equation

\[ X = g(X) \cup L, \]

where the unique \( X \) is given by \( X = g^*(L) \).

**Proof.** Clearly \( X = g^*(L) \) satisfies \( X = g(X) \cup L \). Conversely, take a subset \( U \) of \( W \) satisfying \( U = g(U) \cup L \) and suppose \( U \supseteq g^*(L) \). We take a minimal element \( x_0 \) of \( U - g^*(L) \). \( x_0 \) is in \( g(U) \) because of \( x_0 \in L \). So, there exists an \( x \in U \) such that \( x_0 \in g(\{x_1\}) \) because of \( g(\phi) = \phi \).

If \( x_1 \) is in \( g^*(L) \) then \( x_0 \) is also in \( g^*(L) \), contradicting with \( x_0 \in g^*(L) \). Thus, \( x_1 \) is in \( U - g^*(L) \), that is, \( x_1 \) is an element of \( U - g^*(L) \) less than \( x_0 \). This contradicts with the minimality of \( x_0 \) in \( U - g^*(L) \). Hence, \( U \) must coincide with \( g^*(L) \).

**Theorem 1.** The following system of set equations has a unique solution:

\[ \bigcup_{i=0}^{n} \bigcup_{j=0}^{n} \{ X_i = \bigcup_{j=0}^{n} g_{i\cdot j}(X_j) \cup L_i \}, \]

where each \( L_i \) is a subset of a nonempty well-founded set \( W \) and each \( g_{i\cdot j} \) is a strictly
additive and increasing function of $2^W$ to $2^W$.

Each component $U_i$ of the unique solution

$$(X_0, X_1, \cdots, X_n) = (U_0, U_1, \cdots, U_n)$$

is represented in such a form as

$$\alpha_{i0}(L_0) \cup \alpha_{i1}(L_1) \cup \cdots \cup \alpha_{in}(L_n)$$

for some regular functional expressions $\alpha_{i0}, \alpha_{i1}, \cdots, \alpha_{in}$ over $G = \{g_{ij} | i, j = 0, 1, \cdots, n\}$.

Especially, if each $L_i$ is either the set $M$ of all minimal elements of $W$ or the empty set $\phi$ then each $U_i$ is represented as $\alpha_i(M)$ by some regular functional expression $\alpha_i$ over $G$.

Moreover, when the above system of set equations is derived from an automaton program, if in each $\alpha_k$, we replace each $g_{ij}$ with $\sigma_1 + \sigma_2 + \cdots + \sigma_m$ where $\sigma_1, \sigma_2, \cdots, \sigma_m$ are the labels of all edges from $q_i$ to $q_j$, then $\alpha_k$ becomes an usual regular expression representing a regular set recognized by the automaton program scheme by starting from the state $q_k$ when it is considered as an usual finite automaton.

The proof of the former part of this theorem can be easily obtained by the mathematical induction on the $n$ using Lemma 1 repeatedly, and the latter part is obvious. So the proof is omitted. Thus, the set recognized by an automaton program is represented by a usual regular expression with a suitable interpretation corresponding to the interpretation in the automaton program. Conversely, it is also obvious that, for a given usual regular expression $\alpha$, an automaton program recognizing a subset of $W$ represented by the $\alpha$ under an interpretation $J$ (that is, an association of each symbol with a strictly additive and increasing function of $2^W$ to $2^W$) is obtained by a finite automaton recognizing the set represented by $\alpha$ and the interpretation such that each label $\sigma$ is associated with a decreasing function $f_\sigma$ where the $J$ is constituted by associating each $\sigma$ with $g_{ij} = f_\sigma^{-1}$.

Desiring broad applications, we need also an extension of derivatives applicable to multi-place predicates. We will write an another paper on this theme.

References


Communicated by S. Arikawa

Received October 22, 1983