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

Nishizawa, Teruyasu
Faculty of Economics, Niigata University

https://doi.org/10.5109/13362

出版情報：Bulletin of informatics and cybernetics. 21 (1/2), pp.113-120, 1984-03. Research Association of Statistical Sciences
バージョン：
権利関係：

Bulletin of Informatics and Cybernetics Vol. 21, No. 1~2, 1984

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 $D(f)$ devote 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

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

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 ∂xS of a set S of words over an alphabet Σ by a word x over the Σ is the set

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

where Σ^* denotes the set of all words over Σ.

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

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

where, letting ε denote the empty word and ϕ denote the empty set, E_S is the set defined by

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

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

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

$$P_\sigma(x)\longleftrightarrow x\in\partial_\sigma S.$$

* Faculty of Economics, Niigata University, Igarahi 2-no-cho, Niigata City.

Then the above equation can be rewritten as

$$P(x) \longleftrightarrow (null(x) \wedge t_s) \vee \bigvee_{\sigma \in \Sigma} (top(x) = \sigma \wedge P_\sigma(tail(x))),$$

where $null(x)$ represents $x = \varepsilon$, t_s represents $\varepsilon \in S$, $top(x)$ is the first (i.e. the leftmost) symbol of x and $tail(x)$ is the word such that $x = top(x)\,tail(x)$.

If we put $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$ then the above equivalence is rewritten as

$$\begin{aligned} P(x) \longleftrightarrow\ & \textbf{if } null(x) \textbf{ then } t_s \textbf{ else} \\ & \textbf{if } top\{x) = \sigma_1 \textbf{ then } P_{\sigma_1}(tail(x)) \textbf{ else} \\ & \textbf{if } top(x) = \sigma_2 \textbf{ then } P_{\sigma_2}(tail(x)) \textbf{ else} \\ & \vdots \\ & \textbf{if } top(x) = \sigma_{n-1} \textbf{ then } P_{\sigma_{n-1}}(tail(x)) \textbf{ else} \\ & P_{\sigma_n}(tail(x)). \end{aligned}$$

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 δ of A is given by $\delta(\partial_x S, \sigma) = \partial_{x\sigma} S$ 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

$$\begin{aligned} P_x(y) \longleftrightarrow\ & \textbf{if } null(y) \textbf{ then } t_{\partial_x S} \textbf{ else} \\ & \textbf{if } top(y) = \sigma_1 \textbf{ then } P_{x\sigma_1}(tail(y)) \textbf{ else} \\ & \vdots \\ & \textbf{if } top(y) = \sigma_{n-1} \textbf{ then } P_{x\sigma_{n-1}}(tail(y)) \textbf{ else} \\ & P_{x\sigma_n}(tail(y)) \end{aligned}$$

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 Σ^*, let $\langle x \rangle$ be the number represented by x in the usual diadic notation, that is, $\langle \varepsilon \rangle = 0$ 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

$$P(x) \longleftrightarrow [\langle x \rangle \equiv 0 (mod\ 3)].$$

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

$$P(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } true \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P_0(tail(x))$$
$$\textbf{else } P_1(tail(x)).$$

Then we have

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

(thus, $P_0=P$) and

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

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

$$P_1(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } false \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P_{10}(tail(x))$$
$$\textbf{else } P_{11}(tail(x)).$$

Then we have

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

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

(thus, $P_{11}=P$.)

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

$$P_{10}(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } false \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P_{100}(tail(x))$$
$$\textbf{else } P_{101}(tail(x)).$$

Then we have

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

and

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

(thus, $P_{100}=P_1$ 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.

$$P(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } true \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P(tail(x)) \textbf{ else } P_1(tail(x))$$

$$P_1(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } false \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P_{10}(tail(x)) \textbf{ else } P(tail(x))$$

$$P_{10}(x) \longleftrightarrow \textbf{if } null(x) \textbf{ then } false \textbf{ else}$$
$$\textbf{if } top(x)=0 \textbf{ then } P_1(tail(x)) \textbf{ else } P_{10}(tail(x)).$$

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,

$$D(f)=\{x\in W \mid f(x) \text{ is defined}\}$$

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

$$\partial_f P(x) \longleftrightarrow (\exists y\in D(f))[P(y)\wedge f(y)=x].$$

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

$$P(x) \longleftrightarrow \partial_f P(f(x))$$

because of

$$x\in D(f)\wedge\partial_f P(f(x))$$

$$\longleftrightarrow x\in D(f)\wedge(\exists y\in D(f))[P(y)\wedge f(y)=f(x)]$$

$$\longleftrightarrow x\in D(f)\wedge P(x).$$

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:

$$P(x)\wedge \bigvee_{f\in F} x\in D(f)$$

$$\longleftrightarrow \bigvee_{f\in F}(x\in D(f)\wedge\partial_f P(f(x))).$$

Let $\lambda_F(x)$ denote the predicate

$$\neg \bigvee_{f\in F} x\in D(f).$$

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

$$P(x) \longleftrightarrow \bigvee_{f\in F}(x\in D(f)\wedge\partial_f P(f(x)))\vee(\lambda_F(x)\wedge t_F).$$

DEFINITION 2. For a one-place predicate P over a set W, a system of predicates $P_0, P_1, \cdots, P_n$ and finite families $F_0, F_1, \cdots F_n$ 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 $P_0, P_1, \cdots, P_n$.

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

$$\overset{i=0}{\underset{n}{\{}} P_i(x) \longleftrightarrow \bigvee_{f\in F_i}(x\in D(f)\wedge\partial_f P_i(f(x)))\vee(\lambda_{F_i}(x)\wedge t_i)$$

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

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

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 $\{P_0, P_1, \cdots, P_n\}$,

(2) the initial state is P_0,

(3) the final states are P_i such that t_i=true,

(4) the state transition is represented by a finite directed labeled graph whose nodes are the states $P_0, P_1, \cdots, P_n$ such that an edge from P_i to P_j with a label f represents the relation $\partial_f P_i = P_j$. (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 spesified 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 shceme is called an *automaton program.*

A *computation* of an automaton program is a sequence

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

such that each x_i is in the support W of the interpretation, each q_i is a state and, for each $i=0, 1, \cdots, m-1$, 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

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

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. $\lambda_q(x)$ stands for $\neg \bigvee_{e \in E_q} x \in D(f_e)$.

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*:

$$_{q\in Q}\{P_q(x) \longleftrightarrow \bigvee_{e\in E_q}(x\in D(f_e)\wedge P_{s_e}(f_e(x)))\vee(\lambda_q(x)\wedge q\in T).$$

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* $\{P_0, P_1, \cdots, P_n : F_0, F_1, \cdots, F_n\}$ *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, $f(x)\precneqq x$ 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,

$$g_e(X)=\{x\in D(f_e)\mid f_e(x)\in X\}$$

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

$$g(\phi)=\phi \quad \text{and} \quad g(X)=\bigcup_{x\in X}g(\{x\}),$$

and said to be increasing if, for all y in $g(\{x\})$, $x\precneqq y$ 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:

$$_{q\in Q}\{U_q=\bigcup_{e\in E_q}g_e(U_{s_e})\cup L_q$$

where $L_q=$**if** $q\in T$ **then** M **else** ϕ.

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

$$\begin{aligned} & x\in \bigcup_{e\in E_q}g_e(U_{s_e})\cup L_q \\ \longleftrightarrow\ & (\exists e\in E_q)x\in g_e(U_{s_e})\vee x\in L_q \\ \longleftrightarrow\ & (\exists e\in E_q)[x\in D(f_e)\wedge f_e(x)\in U_{s_e}]\vee(x\in M\wedge q\in T) \end{aligned}$$

$$\longleftrightarrow \bigvee_{e\in E_q}(x\in D(f_e)\wedge P_{s_e}(f_e(x))\vee(\lambda_q(x)\wedge q\in T)$$

$$\longleftrightarrow P_q(x)$$

$$\longleftrightarrow x\in U_q\,.$$

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

$$\overset{i=0}{\underset{n}{\{}}U_i=\bigcup_{j=0}^{n} g_{ij}(U_j)\cup L_i\,.$$

Here, it should be noted that the all g_{ij}'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 $+$, . and $*$ to functions in G, the identity function I and ϕ (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 $X=g^*(L)$ satisfiees $X=g(X)\cup L$. Conversely, take a subset U of W satisfying $U=g(U)\cup L$ and suppose $U\supsetneq g^*(L)$. We take a minimal element x_0 of $U-g^*(L)$. x_0 is in $g(U)$ because of $x_0\notin L$. 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 $x_0\notin g^*(L)$. 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*:

$$\overset{i=0}{\underset{n}{\{}}X_i=\bigcup_{j=0}^{n} g_{ij}(X_j)\cup L_i\,,$$

where each L_i is a subset of a nonempty well-founded set W and each g_{ij} 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* ϕ *then each* U_i *is represented as* $\alpha_i(M)$ *by some regular functional expression* α_i *over* G.

Moreover, when the above system of set equations is derived from an automaton program, if in each α_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* α_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 α, an automaton program recognizing a subset of W represented by the α under an interpretation $\mathcal{I}$ (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 α and the interpretation such that each label σ is associated with a decreasing function f_e where the $\mathcal{I}$ is constituted by associating each σ with $g_e=f_e^{-1}$.

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

[1] NISHIZAWA, T.: *On derivatives of predicates over a wellfounded set* (in Japanese), at LA symposium in summer, (1981).

[2] NAGAI, M. and NISHIZAWA, T.: *A system for Automatically Synthesizing Pure LISP Programs*, Proceedings of 6-th IBM Symposium on MFCS, (1981).

Communicated by S. Arikawa
Received October 22, 1983