Theory of Test Modeling Based on Regular Expressions

Pan Liu[1,2(✉)] and Huaikou Miao[3]

[1] College of Computer Engineering and Science, Shanghai Business School, Shanghai 201400, China
Pan1008@163.com
[2] Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai 201112, China
[3] School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China

Abstract. This paper presents a theory of test modeling by using regular expressions for software behaviors. Unlike the earlier modeling theory of regular expression, the proposed theory is used to build a test model which can derive effective test sequences easily. We firstly establish an expression algebraic system by means of transition sequences and a set of operators. And we then give the modeling method for behaviors of software under test based on this algebraic system. Some examples are also given for illustrating our test modeling method. Compared with the finite state machine model, the expression model is more expressive for the concurrent system and can provide the accurate and concise description of software behaviors.

Keywords: Test modeling · Regular expression · Expression algebraic system · Concurrent operation

1 Introduction

Software testing is a critical activity to assure software quality [1]. However, earlier studies have shown that software testing can consume more than fifty percent of the development costs [2]. Therefore automating software testing as a long-term goal has been highlighted in the industry for many years. Model-based testing [3–5], as a method of automatic test, has been widely studied to generate abstract test sequences. The finite state machine (FSM [6, 7]), a formal notation for describing software behaviors, is often employed for test modeling and test generation, forming a series of test generation methods [8–10].

For a concurrent system, however, it is hard to build a model by FSM due to the limitation of the expressive power of FSM. Therefore the other modeling methods have been suggested for modeling concurrent systems. For example, Petri nets [11, 12] was used for modeling software behaviors and generating test cases for accessibility test [13]. However, Petri nets easily causes the state-space explosion problem [14] when the system is complex. Regular expressions are also used to build the model of distributed systems, such as path expressions [15], behavior expressions [16] and

S. Liu and Z. Duan (Eds.): SOFL+MSVL 2013, LNCS 8332, pp. 17–31, 2014.
DOI: 10.1007/978-3-319-04915-1_2,

extended regular expression [17]. Garg et al. [16, 18] proposed an algebraic model called concurrent regular expressions for modeling and analysis of distributed systems. However, this algebraic model is suitable for model checking and not for test generation because it lacks of the essential path information, which consists of the initial node, the terminal node and path sequences. Ravi et al. [19] proposed a novel methodology for high-level testability analysis and optimization of register-transfer level controller/data path circuits based on regular expressions. Qian et al. [20] presented a method to generate test sequences from regular expressions describing software behaviors. This method firstly uses the FSM to build the model of software behaviors. And then the FSM is converted into a regular expression according to three construction rules. Finally, test sequences are obtained from this regular expression. However, the suggested expression model does not have the capability for describing concurrent operations because regular expressions are derived from FSM.

In this paper, we suggest constructing the test model by regular expressions for software behaviors. Referring to the modeling theories of concurrent regular expressions in [16, 18] and that of FSM in [7, 21], we set up an expression algebraic system. And some examples are employed for illustrating our modeling approaches.

The rest of this paper is organized as follows. Section 2 presents the expression algebraic system. Section 3 introduces the method of test modeling by regular expressions. Some examples of test modeling are presented in Sect. 4. Section 5 discusses the advantages and disadvantages between the traditional test generation method and our test generation method. Section 6 concludes the whole paper.

2 Expression Algebraic System

Before we introduce the expression algebraic system, the definition of FSM needs to be introduced so that we can build the bridge between the regular expression and FSM.

A finite-state machine (FSM) [22, 23] $M = <S, I, O, f, g, s_0>$ consists of a finite set S of states, a finite input alphabet I, a finite output alphabet O, a transition function f that assigns to each state and input pair a new state, an output function g that assigns to each state and input pair an output, and an initial state s_0. According to the definition of FSM, we give the definitions of both transition and transition sequence.

Definition 1 (transition): A transition of FSM is defined by $t = (s_1, i/o, s_2)$, where $f(s_1, i) = s_2, i \in I, g(s_1, i) = o, o \in O, s_1$ is the pre-state of t, s_2 is the next-state of t, i is the transition condition of t and o is the output result of t.

Definition 2 (transition sequence): For any transition a, the syntax of the transition sequence ts can be defined via Backus-Naur form:

$$ts ::= \varepsilon \,|\, a \,|\, a.ts \,|\, ts.a \,|\, ts.ts,$$

Where ε denotes the empty and ts is any transition sequence.

Let Σ be a nonempty set of transition sequences in FSM, and $\varepsilon = a^0$ for any $a \in \Sigma$. Let $\#ts$ denote the number of transitions in ts.

Definition 3 (software regular expression): A software regular expression describing software behaviors is an expression consisting of symbols from Σ and the operators |, +,., *, α (), °, and ||, which are defined as follows:

| denotes the choice operator;
. denotes the concatenation operator;
* is the Kleene closure;
\+ is the positive closure;
α is a positive integer which denotes the alpha closure;
() denotes the range;
° denotes the synchronization;
|| indicates the concurrent operator

In Defintion 3, the descriptions of four operators |,., * and + refer to the statements in [16, 20]. We set the priority of operators high to low: (), *, + and α,., $\circ$ and ||, |.

Definition 4 (expression algebraic system): An expression algebraic system consists of both Σ and the operators |, +,., *, α (), °, and ||, denoted as $<$ Σ, |, +,., *, α (), °, || $>$, and ε is the identity element of this system.

3 Test Modeling

In this section, we do not take account of the inputs and outputs on transitions and all transitions are directly labeled on the edges of the graphs.

3.1 Concatenation Operator

A software behavior model with the concatenation operator shown in Fig. 1 can be described by $t_1.t_2$, where t_1 and t_2 are two transitions, and t_2 is occurred after t_1. The concatenation operator satisfies the following properties:

(1) $\forall a, b \in \Sigma \bullet a.b \neq b.a \Rightarrow a \neq b \wedge a \neq \varepsilon \wedge b \neq \varepsilon$
(2) $\forall a, b, c \in \Sigma \bullet a.b.c = (a.b).c = a.(b.c)$
(3) $\forall a \in \Sigma \bullet a.\varepsilon = \varepsilon.a = a$

3.2 Choice Operator

Let the symbol | denote the choice operator. In the model shown in Fig. 2, the transitions t_3 and t_2 are alternative. So the model can be described by $t_1.(t_3|t_2)$, where t_3

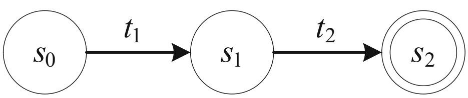


Fig. 1. The software behavior model with the concatenation operator.

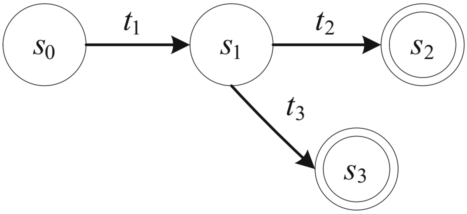


Fig. 2. The software behavior model with the choice operator.

or t_2 is executed in accordance with the different inputs on s_1. The choice operator satisfies the following properties:

(1) $\forall a, b \in \Sigma \bullet a|b \Rightarrow a \vee b.$
(2) $\forall a, b \in \Sigma \bullet a|b = b|a$ (Commutativity)
(3) $\forall a, b, c \in \Sigma \bullet a|b|c = (a|b)|c = a|(b|c)$ (Associativity)
(4) $\forall a \in \Sigma \bullet a|\varepsilon = \varepsilon|a = a$
(5) $\forall a \in \Sigma \bullet a|a = a$ (Identity)
(6) $\forall a, b_1, b_2, \ldots, b_n \in \Sigma \bullet a.(b_1|b_2|\ldots|b_n) = a.b_1|a.b_2|\ldots|a.b_n$ (Distributivity)
(7) $\forall a_1, a_2, \ldots, a_n, b \in \Sigma \bullet (a_1|a_2|\ldots|a_n).b = a_1.b|a_2.b|\ldots|a_n.b$ (Distributivity)

3.3 Kleene Closure

Let the symbol * denotes the Kleene closure. Then the model shown in Fig. 3 can be described as $t_1.\overset{*}{t_2}.t_3$, where t_2 can be executed repeatedly. The Kleene closure satisfies the following properties:

(1) $\forall a \in \Sigma \bullet a^* = \bigcup_{i=0,1,\ldots} a^i$
(2) $\forall a \in \Sigma \bullet (a^*)^* = a^*$ (Absorption)
(3) $a_i \in \Sigma \wedge 1 \leq i \leq n \bullet (a_1|a_2|\ldots|a_n)^* = a_1^*|a_2^*|\ldots|a_n^*$ (Distributivity)
(4) $\varepsilon^* = \varepsilon$

3.4 Positive Closure

Let the symbol + denote the positive closure. E.g., a^+ denotes that a is executed at least once. The model of a temperature control system is shown in Fig. 4, where

– s_0 is the initial state,
– s_2 is the terminal state,

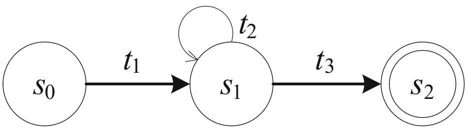


Fig. 3. The software behavior model with the Kleene closure.

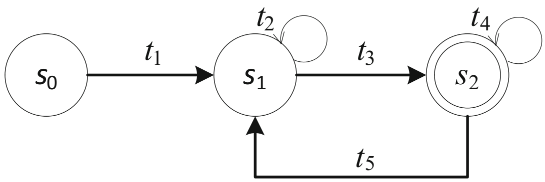


Fig. 4. The software behavior model with the positive closure.

- t_0 denotes that the engine of the temperature control system is launched,
- t_2 denotes the heating-up process when the temperature on s_1 is lower than the given threshold x,
- t_3 denoted that the engine stops working,
- t_4 denotes the cooling process when the temperature on s_2 is still greater than x,
- t_5 denoted the warming process is triggered and the system will return to s_1.

This model can be described by $t_1.\left(t_2^+.t_3.t_4^+.t_5\right)^*.t_2^+.t_3.t_4^+$. The positive closure satisfies the following properties:

(1) $\forall a \in \Sigma \bullet a^+ = \bigcup_{i=1,2,\ldots} a^i$

(2) $\forall a \in \Sigma \bullet a^+ = a.a^* = a^*.a$

(3) $\forall a \in \Sigma \bullet a.a^+ = a^+.a = a^+$

(4) $\forall a \in \Sigma \bullet (a^+)^+ = a^+$ (Absorption)

(5) $a_i \in \Sigma \wedge 1 \le i \le n \bullet (a_1|a_2|\ldots|a_n)^+ = a_1^+|a_2^+|\ldots|a_n^+$ (Distributivity)

(6) $\varepsilon^+ = \varepsilon$

3.5 Alpha-closure

Let α be the alpha-closure, which denotes a maximum cycle times. E.g., b^α denotes that the transition b is executed repeatedly α times. The model of an online bank login system is shown in Fig. 5. If the user types the wrong username or password for three

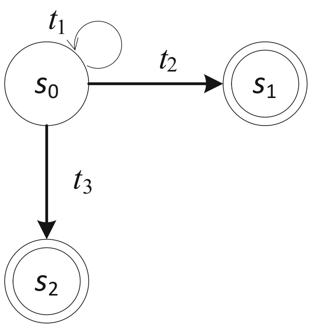


Fig. 5. The software behavior model with the alpha-closure.

times, the system will be automatically locked for 24 h. The symbols in this model denote as follows:

- s_0 denotes the login page,
- s_1 is the main page,
- s_2 denotes the locked page,
- t_1 denotes the self-check on s_0,
- t_2 denotes the login success,
- t_3 denotes the login failure.

According to the above description of system, there exists $\alpha = 3$ and this system can be described by $t_1^3.t_3|t_2|t_1.t_2|t_1^2.t_2$. The alpha-closure satisfies the following Properties:

(1) $\forall a \in \Sigma \bullet a^{\alpha} = \overbrace{a.a.\ldots.a}^{\alpha}$
(2) $\forall a \in \Sigma \bullet a.a^{\alpha} = a^{\alpha}.a = a^{\alpha}$
(3) $a_i \in \Sigma \wedge 1 \leq i \leq n \bullet (a_1|a_2|\ldots|a_n)^{\alpha} = a_1^{\alpha}|a_2^{\alpha}|\ldots|a_n^{\alpha}$ (Distributivity)
(4) $\forall a \in \Sigma \bullet (a^*)^{\alpha} = (a^{\alpha})^* = a^{\alpha}$ (Absorption)
(5) $\forall a \in \Sigma \bullet (a^+)^{\alpha} = (a^{\alpha})^+ = a^{\alpha}$ (Absorption)
(6) $\varepsilon^{\alpha} = \varepsilon$

3.6 Synchronous Operator

Let the symbol $\circ$ denote the synchronous operator, which can describe the synchronization between two or more transition sequences. E.g., $a \circ b$ denotes that both a and b are synchronized in the system. A simple model of the bus scheduling system at the terminal station is shown in Fig. 6. In this system, buses entering and leaving the station are synchronous. The symbols in this model denote as follows:

- s_0 denotes the initial state of the terminal station,
- s_1 denotes the state of the terminal station after a period of time,
- t_1 denotes the sequences of the buses entering the station,
- t_2 denotes the sequences of the buses leaving the station.

This model can be described by $t_1 \circ t_2$. The synchronous operator satisfies the following Properties:

(1) $\forall a, b \in \Sigma \bullet a \circ b = b \circ a$ (Commutativity)
(2) $\forall a \in \Sigma \bullet a \circ \varepsilon = a$

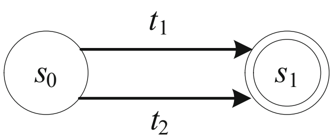


Fig. 6. The software behavior model with the synchronous operator.

(3) $\forall a, b, c \in \Sigma \bullet a \circ b \circ c = (a \circ b) \circ c = a \circ (b \circ c)$ (Associativity)
(4) $\forall a, b \in \Sigma \bullet \#a = \#b = 1 \Rightarrow a \circ b = a.b|b.a$
(5) $\forall a, b, c \in \Sigma \bullet a \circ b.c = ((a \circ b).c)|(b.(a \circ c))$
(6) $\forall a, b, c \in \Sigma \bullet a.b \circ c = ((a \circ c).b)|(a.(b \circ c))$
(7) $\forall a, b_1, \ldots, b_n \in \Sigma \bullet a \circ (b_1|b_2|\ldots|b_n)$
$= (a \circ b_1)|(a \circ b_2)|\ldots|(a \circ b_n)$ (Distributivity)

Theorem 1
$\forall a, b, c, d \in \Sigma \bullet \#a = \#b = \#c = \#d = 1 \Rightarrow (a.b \circ c.d = a.b.c.d|a.c.b.d|a.c.d.b|c.a.b.d|c.a.d.b|c.d.a.b)$.

Proof. According to the Property (5) of $\circ$,

$$a.b \circ c.d = ((a.b \circ c).d)|(c.(a.b \circ d)) \tag{1}$$

By the Property (6) of $\circ$,

$$a.b \circ c = (a \circ c).b|a.(b \circ c) \tag{2}$$

By the Property (4) of $\circ$ and $\#a = \#b = \#c = \#d = 1$,

$$a \circ c = a.c|c.a \tag{3}$$

$$b \circ c = b.c|c.b \tag{4}$$

From Eqs. (2)–(4) and the Properties (3) and (7) of |,

$$\begin{aligned} a.b \circ c &= (a.c|c.a).b|a.(b.c|c.b) \\ &= (a.c.b|c.a.b)(a.b.c|a.b.c) \\ &= a.c.b|c.a.b|a.b.c|a.c.b \end{aligned} \tag{5}$$

By the Property (6) of $\circ$,

$$a.b \circ d = (a \circ d).b|a.(b \circ d) \tag{6}$$

According to the Property (4) of $\circ$ and $\#a = \#b = \#c = \#d = 1$,

$$a \circ d = a.d|d.a \tag{7}$$

$$b \circ d = b.d|d.b \tag{8}$$

From Eqs. (6)–(8) and the Properties (3) and (7) of |,

$$\begin{aligned} a.b \circ d &= (a.d|d.a).b|a.(b.d|d.b) \\ &= (a.d.b|d.a.b)|(a.b.d|a.d.b) \\ &= a.d.b|d.a.b|a.b.d \end{aligned} \tag{9}$$

By the Properties (3), (6) and (7) of |,

$$(a.b \circ c).d = (a.c.b|c.a.b|a.b.c|a.c.b).d$$
$$= a.c.b.d|c.a.b.d|a.b.c.d|a.c.b.d \quad (10)$$

$$c.(a.b \circ d) = c.(a.d.b|d.a.b|a.b.d|a.d.b)$$
$$= c.a.d.b|c.d.a.b|c.a.b.d \quad (11)$$

From Eqs. (1) (10) and (11),

$$a.b \circ c.d = a.c.b.d|c.a.b.d|a.b.c.d|a.c.b.d|c.a.d.b|c.d.a.b|c.a.b.d$$
$$= a.c.b.d|c.a.b.d|a.b.c.d|a.c.b.d|c.a.d.b|c.d.a.b$$

□

Theorem 2: The synchronous operator between any two transition sequences is equal to the ***C***hoice ***O***perator of the ***F***inite ***T***ransition ***S***equences, denoted as **COFTS**.

Proof. Assume that two transition sequences are $\mathrm{A} = a_1.a_2...a_i$ and $\mathrm{B} = b_1.b_2...b_j$, where $a_k(1 \le k \le i)$ and $b_l(1 \le l \le j)$ are two transitions. The Proof of Theorem 2 includes two phases: (1) let $i = 1$ and then prove $\mathrm{A} \circ \mathrm{B} = a_1 \circ (b_1.b_2...b_j)$ is **COFTS**, and (2) prove $\mathrm{A} \circ \mathrm{B} = (a_1.a_2...a_i) \circ (b_1.b_2...b_j)$ is **COFTS**.

Base case 1: $i = 1$ and $j = 1$.

According to the Property (4) of ∘ and the assumption that a_1 and b_1 are two transitions,

$$\mathrm{A} \circ \mathrm{B} = a_1 \circ b_1 = a_1.b_1|b_1.a_1, \quad (12)$$

which are the choice operation of two transition sequences.

Base case 2: $i = 1$ and $j = 2$.

According to the Properties (4) and (5) of ∘ and the Properties (3), (6) and (7) of |,

$$\mathrm{A} \circ \mathrm{B} = a_1 \circ b_1.b_2$$
$$= ((a_1 \circ b_1).b_2)|(b_1.(a_1 \circ b_2))$$
$$= ((a_1.b_1|b_1.a_1).b_2)|(b_1.(a_1.b_2|b_2.a_1))$$
$$= a_1.b_1.b_2|b_1.a_1.b_2|b_1.a_1.b_2|b_1.b_2.a_1 \quad (13)$$

which are the choice operation of four transition sequences.

Inductive hypothesis 1. Assume that Theorem 2 is true for $i = 1$ and $j = m{-}1$. That is,

$$\mathrm{A} \circ \mathrm{B} = \mathrm{C}_1|\mathrm{C}_2|...|\mathrm{C}_k, \quad (14)$$

where $\mathrm{C}_1...\mathrm{C}_k$ are transition sequences and k is a finite positive integer.

We need to prove A∘ B is also **COFTS** for $i = 1$ and $j = m$. Assume $\mathrm{B}_1 = b_1.b_2...b_{m-1}$. Then according to the property (5) of ∘ ,

$$\mathrm{A} \circ \mathrm{B} = a_1 \circ \mathrm{B}_1.b_m$$
$$= (a_1 \circ \mathrm{B}_1).b_m|\ \mathrm{B}_1.(a_1 \circ b_m) \quad (15)$$

According to **Inductive hypothesis 1**,

$$a_1 \circ \mathrm{B}_1 = \mathrm{C}_1|\mathrm{C}_2|\ldots|\mathrm{C}_k \tag{16}$$

By the property (4) of ∘ , and both a_1 and b_m are two transitions,

$$a_1 \circ b_m = a_1.b_m|b_m.a_1 \tag{17}$$

which is **COFTS**.

Hence according to the Property (6) of |,

$$\begin{aligned} \mathrm{B}_1.(a_1 \circ b_m) &= \mathrm{B}_1.(a_1.b_m|b_m.a_1) \\ &= (b_1.b_2\ldots.b_{\mathrm{m}-1}).(a_1.b_m|b_m.a_1) \\ &= b_1.b_2\ldots.b_{\mathrm{m}-1}.a_1.b_m|b_1.b_2\ldots.b_{\mathrm{m}-1}.b_m.a_1 \end{aligned} \tag{18}$$

which is **COFTS**.

From Eqs. (15), (17)–(18),

$$\mathrm{A} \circ \mathrm{B} = a_1.b_m|b_m.a_1|b_1.b_2\ldots.b_{\mathrm{m}-1}.a_1.b_m|b_1.b_2\ldots.b_{\mathrm{m}-1}.b_m.a_1 \tag{19}$$

which is **COFTS**.

$$\text{Hence theorem 2 is true for } i = 1 \text{ and any } j. \tag{20}$$

Inductive hypothesis 2. Assume that Theorem 2 is true for $i = n\text{–}1$ and any j. That is,

$$\mathrm{A} \circ \mathrm{B} = \mathrm{D}_1|\mathrm{D}_2|\ldots|\mathrm{D}_l, \tag{21}$$

where $\mathrm{D}_1 \ldots \mathrm{D}_l$ are transition sequences and l is a finite positive integer.

We need to prove A ∘ B is also **COFTS** for $i = n$ and any j. Assume $\mathrm{A}_1 = a_1.a_2\ldots.a_{n-1}$. Then according to the property (6) of ∘ ,

$$\begin{aligned} \mathrm{A} \circ \mathrm{B} &= \mathrm{A}_1.\mathrm{a}_n \circ \mathrm{B} \\ &= \mathrm{A}_1.a_n \circ b_1.b_2\ldots.b_j \\ &= (\mathrm{A}_1 \circ b_1.b_2\ldots.b_j).a_n|\mathrm{A}_1.(a_n \circ b_1.b_2\ldots.b_j) \end{aligned} \tag{22}$$

According to **Inductive hypothesis 2**,

$$(\mathrm{A}_1 \circ b_1.b_2\ldots.b_j) = \mathrm{D}_1|\mathrm{D}_2|\ldots|\mathrm{D}_l \tag{23}$$

which is **COFTS**.

From (22) and the Property (7) of |,

$$\begin{aligned} (\mathrm{A}_1 \circ b_1.b_2\ldots.b_j).a_n &= (\mathrm{D}_1|\mathrm{D}_2|\ldots|\mathrm{D}_l).a_n \\ &= \mathrm{D}_1.a_n|\mathrm{D}_2.a_n|\ldots|\mathrm{D}_l.a_n \end{aligned} \tag{24}$$

which is **COFTS**.

By (20), $a_n \circ b_1.b_2\ldots b_j$ is **COFTS**. Assume that

$$a_n \circ b_1.b_2\ldots.b_j = \mathrm{K}_1|\mathrm{K}_2|\ldots|\mathrm{K}_p \tag{25}$$

where $\mathrm{K}_1 \ldots \mathrm{K}_p$ are transition sequences and p is a finite positive integer.

Then according to the Property (6) of |,

$$\begin{aligned} \mathrm{A}_1.(a_n \circ b_1.b_2 \ldots .b_j) = &\ \mathrm{A}_1.(\mathrm{K}_1|\mathrm{K}_2|\ldots|\mathrm{K}_p) \\ = &\ \mathrm{A}_1.\mathrm{K}_1|\mathrm{A}_1.\mathrm{K}_2|\ldots|\mathrm{A}_1.\mathrm{K}_p \end{aligned} \quad (26)$$

which is **COFTS**.

From Eqs. (24) and (26), A∘ B is also **COFTS** for $i = n$ and any j. To sum up, Theorem 2 is proved. □

According to Theorem 2, we always make use of the choice operator of finite transition sequences to denote the synchronous operations among some transition sequences.

3.6.1 Concurrent Operator

Let the symbol ‖ denote the concurrent operator. a ‖ b denotes a or b is a single occurrence, or the synchronous occurrence denoted as $a \circ b$. The model described as the stock trading requests is shown in Fig. 7. In the stock trading system, the trading requests that the buyers and the sellers are concurrent. The symbols in the model are described as follows:

- s_0 denotes the current state of the stock trading,
- s_1 denotes the next state of the stock trading,
- t_1 denotes the sequences of the buyer requests,
- t_2 denotes the sequences of the seller requests,
- t_3 denotes the next state is converted into the current state.

The model shown in Fig. 7 can be described by $((t_1||t_2).t_3)^*$. The concurrent operator satisfies the following properties:

(1) $a||b = a|b|a \circ b \forall a, b \in \Sigma$

(2) $a||b = b||a \forall a, b \in \Sigma$ (Commutativity)

(3) $a||\varepsilon = \varepsilon||a = a \forall a \in \Sigma$ (Commutativity and Identity)

(4) $a||b||c = (a||b)||c = a||(b||c) \forall$a,b,c $\in \Sigma$ (Associativity)

(5) $(a_1|a_2|\ldots|a_n)||b = (a_1||b)|(a_2||b)|\ldots|(a_n||b) \forall a_1, \ldots, a_n, b \in \Sigma$ (Distribution)

Corollary 1: The concurrent operation of any two transition sequences is **COFTS**.

Proof. Assume that two transition sequences are A and B. Then A ‖ B = A | B | A∘ B. According to Theorem 2, A∘ B is **COFTS**, hence A ‖ B is also **COFTS**. Corollary 1 is proved. □

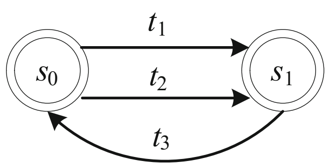


Fig. 7. The software behavior model with the concurrent operator.

Corollary 2: The concurrent operations among finite transition sequences are **COFTS**.

Proof. Assume that there are a suite of test sequences A_1, A_2, …, and A_i, where i is a finite positive integer. Then Corollary 2 can be rewritten as $A_1 \| A_2 \| \ldots \| A_i$ is **COFTS**.

Base case: $i = 1$

Since A_1 is a transition sequence, Corollary 2 is true.

Base case: $i = 2$

By Corollary 1, $A_1 \| A_2$ is **COFTS**, hence Corollary 2 is true.

Inductive hypothesis. Assume that Corollary 2 is true for $i = n$-1. That is,

$$A_1 || A_2 || \ldots || A_{n-1} = B_1 | B_2 | \ldots | B_k, \quad (27)$$

where B_i $(1 \leq i \leq k)$ is a transition sequence.

We need to prove $A_1 \| A_2 \| \ldots \| A_n$ is also **COFTS** for $i = n$.

By **Inductive hypothesis** and the property (5) of ||,

$$\begin{aligned} A_1 || A_2 || \ldots || A_n &= (A_1 || A_2 || \ldots || A_{n-1}) || A_n \\ &= (B_1 | B_2 | \ldots | B_k) || A_n \\ &= (B_1 || A_n) | (B_2 || A_n) | \ldots | (B_k || A_n) \end{aligned} \quad (28)$$

By Corollary 1,

$$B_i || A_n (1 \leq i \leq n) \text{ is } \mathbf{COFTS}. \quad (29)$$

From (27)–(29),

$$A_1 || A_2 || \ldots || A_n \text{ is } \mathbf{COFTS}. \quad (30)$$

To sum up, Corollary 2 is proved. □

According to Corollary 2, any one of regular expressions with concurrent operators can be denoted as the choice operation of finite transition sequences.

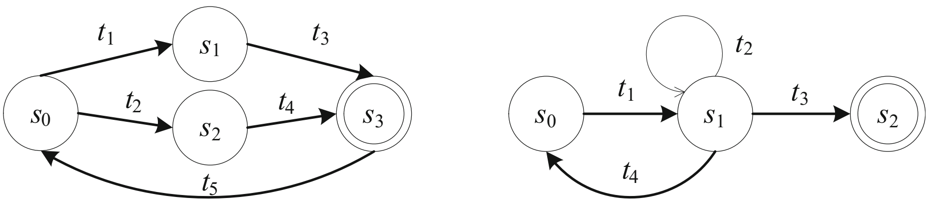
4 Modeling Capability

Using the expression algebraic system, we can construct the model of the complex system. Now we consider building the expression models for two complex systems with the different software requirements.

Figure 8 shows two FSM models. Assume that there exist many different software requirements for two models shown in Fig. 8.

Case 1: Software requirements for the model shown in Fig. 8 (a) include that

- s_0 is the start state,
- s_3 is the terminal state,
- $t_1.t_3$ and $t_2.t_4$ are choice,
- t_5 is a return transition.



(a) (b)

Fig. 8. Two models of the complex systems.

Therefore the system in Case 1 can be described by $(t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^*$.

Case 2: Software requirements for the model shown in Fig. 8 (a) include that

- s_0 is the start state,
- S_3 is the terminal state,
- $t_1.t_3$ and $t_2.t_4$ are concurrent,
- t_5 must be executed at least once.

Therefore the system in Case 2 can be described by $(t_1.t_3 \parallel t_2.t_4).(t_5.(t_1.t_3 \parallel t_2.t_4))^+$.

Case 3: Software requirements for the model shown in Fig. 8 (b) include that

- s_0 is the start state
- s_2 is the terminal state.

Therefore the system in Case 3 can be described by $t_1.(t_2^*.(t_4.t_1.t_2^*)^*).t_3$.

Case 4: Software requirements for the model shown in Fig. 8 (b) include that

- s_0 is the start state
- s_2 is the terminal state.
- t_2 and t_4 are choice.

Therefore the system in Case 4 can be described by $t_1.(t_2^* \mid (t_4.t_1)^*).t_3$.

Discussion 1: Through Cases 1–4, we find the fact that the FSM model can't distinguish the system with the nice distinctions in software requirements, while the expression model can distinguish them. Therefore the modeling capability of regular expressions is more expressive than that of the FSM.

5 Test Sequences

In the traditional test generation method, a graph (or FSM) is usually transformed to a test tree. And then all paths from the root to all leaves in this tree are produced. According to this method, we obtain two test sequences (as test paths) $t_1.t_3.t_5$ and $t_2.t_4.t_5$ from the test tree shown in Fig. 9 (b) for the model shown in Fig. 9 (a). However, $t_1.t_3.t_5$ and $t_2.t_4.t_5$ are two ineffective test segments because the last state s_0 in two sequences is not the terminal state s_3 of the system shown in Fig. 9(a).

Now we demonstrate the method of test sequence generation from regular expressions.

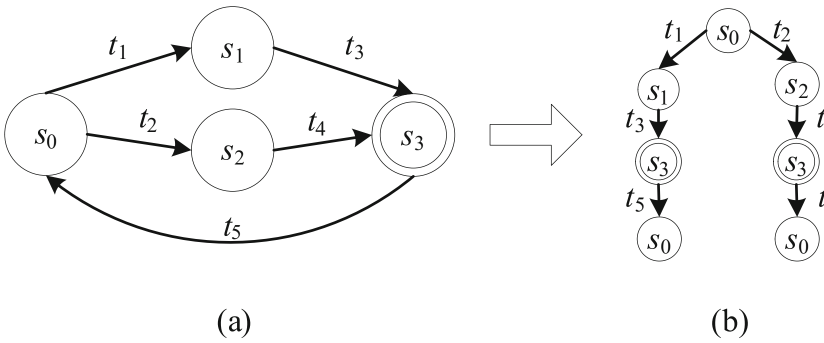


Fig. 9. The traditional test generation method.

Assume that software requirements satisfy case 1 in Sect. 4. The model shown in Fig. 9 (a) can be described by $(t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^*$. Then we assign 0, 1 and k for * in regular expression. Hence

$$
\begin{aligned}
&(t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^* = (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^0 \mid (t_1.t_3 \mid \\
&t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^1 \mid (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^k \\
&= (t_1.t_3 \mid t_2.t_4).\varepsilon \mid (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4)) \mid (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^k \\
&= (t_1.t_3 \mid t_2.t_4) \mid (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4)) \mid (t_1.t_3 \mid t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^k \mid (t_1.t_3 \mid \\
&\quad t_2.t_4).(t_5.(t_1.t_3 \mid t_2.t_4))^k \\
&= t_1.t_3 \mid t_2.t_4 \mid (t_1.t_3.(t_5.(t_1.t_3 \mid t_2.t_4)) \mid t_2.t_4.(t_5.(t_1.t_3 \mid t_2.t_4)) \mid t_1.t_3.(t_5.(t_1.t_3 \mid \\
&\quad t_2.t_4))^k \mid t_2.t_4.(t_5.(t_1.t_3 \mid t_2.t_4))^k \\
&= t_1.t_3 \mid t_2.t_4 \mid t_1.t_3.t_5.t_1.t_3 \mid t_1.t_3.t_5.t_2.t_4 \mid t_2.t_4.t_5.t_1.t_3 \mid t_2.t_4.t_5.t_2.t_4 \mid t_1.t_3. \\
&\quad \left((t_5.t_1.t_3)^k \mid (t_5.t_2.t_4)^k\right) \mid t_2.t_4.((t_5.t_1.t_3)^k \mid (t_5.t_2.t_4)^k) \\
&= t_1.t_3 \mid t_2.t_4 \mid t_1.t_3.t_5.t_1.t_3 \mid t_1.t_3.t_5.t_2.t_4 \mid t_2.t_4.t_5.t_1.t_3 \mid t_2.t_4.t_5.t_2.t_4 \mid \\
&\quad t_1.t_3.(t_5.t_1.t_3)^k \mid t_1.t_3.(t_5.t_2.t_4)^k \mid t_2.t_4.(t_5.t_1.t_3)^k \mid t_2.t_4.(t_5.t_2.t_4)^k
\end{aligned}
$$

Discussion 2: (1) Sometimes, test sequences generated from the traditional method can't be taken as the effective test paths. For example, the terminal node in test path $t_1.t_3.t_5$ is s_0 which deviates from the actual software requirements. (2) Test coverage of test sequences generated from the traditional method is not complete, resulting in the low fault detection capability. (3) Based on the operations in the algebraic system, we can obtain test sequences from regular expressions. And all operations can be automatically achieved. (4) Test sequences derived from our method include all possible paths, hence they have the higher fault detection capability than those derived from the traditional method. (5) A shortcoming of our method is that the number of test sequences is too much. Therefore the redundant test sequences need to be reduced according to some techniques in Ref. [24].

6 Conclusions

In this paper, we present an expression algebraic system to support test modeling. This system consists of regular expressions denoted by transition sequences and operators, including., |, *, +, $\propto$ (), ° and ||. Some examples are given to illustrate our modeling method and test generation method. Compared with the FSM model, the expression model not only is more expressive for the concurrent system, but also can generate high quality test sequences from the model. In the future, we will plan to unify test modeling and test generation into a frame by regular expressions. And we will also research the techniques for reduced-order modeling and redundant reduction.

Acknowledgments. This work is supported by National Natural Science Foundation of China (NSFC) under grant No. 61170044 and No. 61073050, Shanghai Natural Science Fund (No. 13ZR1429600), Innovation Program of Shanghai Municipal Education Commission (No. 13YZ141), and Young teacher training scheme of Shanghai Universities (No. SXY12014).

References

1. Liu, P., Miao, H.: A new approach to generating high quality test cases. In: 2010 19th IEEE Asian Test Symposium (ATS), pp. 71–76. IEEE (2010)
2. Bertolino, A.: Software testing research: achievements, challenges, dreams. In: Future of Software Engineering, FOSE'07, pp. 85–103. IEEE (2007)
3. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J., Lott, C.M., Patton, G.C., Horowitz, B.M.: Model-based testing in practice. In: Proceedings of the 21st International Conference on Software Engineering, pp. 285–294. ACM (1999)
4. Utting, M., Legeard, B.: Practical Model-Based Testing: a Tools Approach. Morgan Kaufmann, San Francisco (2010)
5. Hemmati, H., Arcuri, A., Briand, L.: Achieving scalable model-based testing through test case diversity. ACM Trans. Softw. Eng. Methodol. (TOSEM) **22**, 6 (2013)
6. Belinfante, A., Frantzen, L., Schallhart, C.: 14 tools for test case generation. In: Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.) Model-Based Testing of Reactive Systems. LNCS, vol. 3472, pp. 391–438. Springer, Heidelberg (2005)
7. Liu, P., Miao, H.-K., Zeng, H.-W., Liu, Y.: FSM-based testing: Theory, method and evaluation. Jisuanji Xuebao(Chinese J. Comput.) **34**, 965–984 (2011)
8. Fujiwara, S., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. Softw. Eng. **17**, 591–603 (1991)
9. Sidhu, D.P., Leung, T.-K.: Formal methods for protocol testing: a detailed study. IEEE Trans. Softw. Eng. **15**, 413–426 (1989)
10. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng. **4**, 178–187 (1978)
11. Lai, R.: A survey of communication protocol testing. J. Syst. Softw. **62**, 21–46 (2002)
12. Wu, N.Q., Zhou, M.: Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets. IEEE Trans. Autom. Sci. Eng. **9**, 446–454 (2012)
13. Notomi, M., Murata, T.: Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis. IEEE Trans. Softw. Eng. **20**, 325–336 (1994)

14. Chu, F., Xie, X.-L.: Deadlock analysis of Petri nets using siphons and mathematical programming. IEEE Trans. Robot. Autom. **13**, 793–804 (1997)
15. Li, Q., Moon, B.: Indexing and querying XML data for regular path expressions. In: VLDB, pp. 361–370 (2001)
16. Garg, V.K., Ragunath, M.: Concurrent regular expressions and their relationship to Petri nets. Theoret. Comput. Sci. **96**, 285–304 (1992)
17. Sen, K., Roşu, G.: Generating optimal monitors for extended regular expressions. Electron. Notes Theoret. Comput. Sci. **89**, 226–245 (2003)
18. Garg, V.K.: Modeling of Distributed Systems by Concurrent Regular Expressions. In: FORTE, pp. 313–327 (1989)
19. Ravi, S., Lakshminarayana, G., Jha, N.K.: TAO: regular expression-based register-transfer level testability analysis and optimization. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **9**, 824–832 (2001)
20. Qian, Z.S.: Model-based approaches to generating test cases for web applications. Ph.D. thesis, Shanghai University (2008)
21. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): Model-Based Testing of Reactive Systems. LNCS, vol. 3472. Springer, Heidelberg (2005)
22. Hopcroft, J.E.: Introduction to Automata Theory, Languages, and Computation, 3/E. Pearson Education India (2008)
23. Rosen, K.H., Krithivasan, K.: Discrete Mathematics and Its Applications, 5th edn. McGraw-Hill, New York (2003)
24. Miao, H.K., Liu, P., Mei, J., Zeng, H.W.: A new approach to automated redundancy reduction for test sequences. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009. PRDC'09, pp. 93–98. IEEE (2009)

http://www.springer.com/978-3-319-04914-4

Structured Object-Oriented Formal Language and Method
Third International Workshop, SOFL+MSVL 2013,
Queenstown, New Zealand, October 29, 2013, Revised
Selected Papers
Liu, S.; Duan, Z. (Eds.)
2014, X, 193 p. 64 illus., Softcover
ISBN: 978-3-319-04914-4