マルチメディア通信と分散処理 66－10
（１９９４．７．７）

拡張有限状態機械モデルにおける通信プロトコルのテスト系列の自動生成の一手法

李湘東，福田真二，樋口昌宏，東野輝夫，谷口健一
大阪大学基礎工学部情報工学科

あらまし　本稿では，有限状態機械の制御部に加えて入力値を保持するためのレジスタを持つ拡張有限状態機械モデル（以下EFSM/inと呼ぶ）を考え，このモデルで書かれた通信プロトコルの形式仕様からE-UIO系列と呼ばれる適合性試験のためのUIO系列を自動生成するための新しい手法を提案する．このモデルで取り扱う入出力データ及びレジスタ値は整数に制限している．レジスタは入力値を保持するためのみに用いる．各遷移の遷移条件は，入力値を表す変数とレジスタ値を表す変数，整数上の加減算，大小比較のみからなる述語で記述する．このようなクラスに対して，幅優先逆探索ですべてのパスを順次生成しつつ，整数線形計画問題の手法を用いて，E-UIO系列になるかどうかを判定し，そうなら，入出力の具体値を決め，E-UIO系列を生成する．

Automatic Test Case Generation for Communication Protocols in an EFSM Model

Xiangdong Li, Shinji Fukuda, Masahiro Higuchi, Teruo Higashino and Kenichi Taniguchi
Department of Information and Computer Sciences, Osaka University
Machikaneyama 1-3, Toyonaka, Osaka 560, Japan

Abstract　In this paper, we propose an automatic test case derivation method for an extended finite state machine model called EFSM/in. In the EFSM/in model, each EFSM has a finite control and a finite number of registers. The data types of I/O data and registers' values are restricted to the integer type. The registers are used only for keeping input values. The transition conditions on input variables and register variables are written using addition, subtraction, comparison and some Boolean operators. For such a restricted class, we give an algorithm for deriving UIO sequences automatically where the derived UIO sequences contain not only I/O events but also their concrete data values, if the UIO sequences exist. The algorithm is carried out using a procedure for solving integer linear programming problems.

1. Introduction

Precise specifications are essential for the design and implementation of distributed systems and communication networks. The use of formal description techniques(FDT's) allows the automation of conformance testing [BoUy 91]. In the protocol conformance testing, there are two aspects. One is control flow. The other is data flow. The control flow is usually described as a simple specification model of finite state machines (FSM's). While the test cases can be generated for the FSM models automatically, they become usually infeasible for more complex models, such as extended FSM's because of the data parameters.

The selection of appropriate test cases including data parameters is an important issue for conformance testing of communication protocols. Certain authors have considered extended finite state machine (EFSM) specifications which include data parameters and additional state variables. Usually, the data flow relations between input/output parameters and state variables are considered in the test selection process [Sari 87], however, it is generally assumed that the transitions do not contain enabling conditions depending on the additional state variables, or such dependencies are treated in an informal manner. The approach of [ChZh 94] generates the values of the data parameters using a heuristic constraints solver automatically. In the approach of [UrYa 91], first, a flow graph is constructed. Then, the relations between input and output data are observed and the test cases are derived based on the observations. The approach of [WaLi 92] proposes an axiomatic approach to generate test cases where, by traversing a given path in the EFSM carefully, the extended path is generated.

In this paper, first, we introduce an extended finite state machine model called EFSM/in. In the EFSM/in model, each EFSM has a finite control and a finite number of registers. We only consider the data parameters belonging to the integer type. The registers are used only for keeping input data[1]. The operations used for representing transition conditions are restricted to addition, subtraction, comparison and some Boolean operators. For such a class, we give an algorithm for deriving UIO sequences automatically where the derived UIO sequences contain not only I/O events but also their concrete data values. It is known that the theory of integers with addition is decidable [HoUl 79]. This is the reason why we can generate the UIO sequences with data values automatically, as further discussed in this paper. The derivation algorithm is carried out using a decision procedure for solving integer linear programming problems.

The paper is structured as follows. The definition of our EFSM/in model is given in Section 2. The general methods for generating UIO sequences for FSM models cannot apply for EFSM models. In Section 3, the reason is explained. In Section 4, we provide an idea to solve this problem.

[1] Our EFSM/in model means an EFSM model for keeping input values.

2. Our EFSM Model

In this paper, we use an EFSM model called EFSM/in. First, we introduce some notations.

[Definition 2.1]

A term which consists of integers, variables of integer type, and operators "+" and "-" is called a *P-term*. A *P-sentence* is defined inductively as follows.

(A) If t1 and t2 are P-terms, then "t1=t2", "t1<t2", "t1≤t2", "t1≥t2" and "t1>t2" are P-sentences.

(B) If α and β are P-sentences, then "(α) and (β)", "(α) or (β)", "not (α)", "(α) $\supset$ (β)" are P-sentences. []

For example, "x", "3" and "x+y-3" are P-terms. The expressions "x=y+1" and "(x≥y-z) or (z=w)" are P-sentences. However, "x^2+2x-3=0" is not a P-sentence because multiplication is used.

Using these notations, we define the EFSM/in model formally. In our EFSM/in model, an EFSM has a finite state control and a finite number of registers $R_1,...,R_n$ where "n" denotes the number of the registers. The specification of an EFSM is described as a labeled directed graph such as Fig. 1. The types of all registers must be integers. We assume that the number of I/O gates is finite. Each node and edge represent a state of the finite control and a transition, respectively. Each edge has a label whose form is <C,a?x/b!E,RD>. Here, "a" and "b" denote gate names. The symbol "a?x" denotes an input event and the variable "x" represents an input value given from the gate "a". More than one input values may be given for an input event. Such an input event is described like as "a?y,z". The event "b!E(x,R_1,...,R_n)" denotes an output event and the value of the expression "E(x,R_1,...,R_n)" is emitted from the gate "b" (more than one outputs may be emitted). The output E(x,R_1,...,R_n) must be a P-term. The pair of one input event and one output event is executed at each transition. Such a pair is called an I/O event. At the initial state sinit , the initial values R_1.init,...,R_n.init of the registers are specified. The first element "C" of the label is called a *transition condition*. A transition condition "C" in <C,a?x/b!E, RD> must be a P-sentence which may contain the variable "x" and registers "R_1",...,"R_n". The value of the transition condition "C" is decided by substituting the current input value and the current values of the registers into "x" and "R_1,...,R_n", respectively. If the value of the transition condition "C" in <C,a?x/b!E, RD> is true, then the I/O event "a?x/b!E" can be executed. Otherwise, it is not executable. We assume that the EFSM is deterministic. That is, if there are state transitions "s-<C_1,a?x/b!E_1,RD_1>->t_1" , "s-<C_2,

a?x/d!E2,RD2>->t2" , ... , "s-<Ck,a?x/f!Ek,RDk>->tk" from a state "s", for any input value and any register values, at most one of C1,...,Ck must be true. The third element RD in the label <C,a?x/b!E,RD> is called a *register definition statement*. The register definition statement is described as an n-tuple of substitution statements [SS1, ... ,SSn] where each SSj must be either "Rj←Rj" or "Rj←fj(x)". Here, "fj(x)" must be a P-term which may contain only the variable "x". The substitution statement "Rj←Rj" denotes that the value of the register "Rj" is not changed in this transition. The substitution statement "Rj←fj(x)" denotes that the value of the register "Rj" after executing this transition becomes fj(x), where "fj" is a function to calculate the next value of the register "Rj" from the input value. In the labeled directed graph representing the EFSM, the substitution statement "Rj←Rj" is omitted.

We assume that there is a *reset* event for each state in a given EFSM. That is, we assume that there is a transition sk-<true, reset/null, {R1← R1.init, ... ,Rn← Rn.init}>->sinit for each state sk where sinit denotes the initial state of the EFSM and Rj.init denotes the initial value of the resister Rj. Such reset events are not described explicitly in the labeled directed graph representing the EFSM. The I/O event "reset/null" may be abbreviated as "reset". We also assume that each state is reachable from the initial state. Hence, for any state sk, there exists a test case starting from the initial state sinit and ending to the state sk (here, we treat a sequence of I/O events as a test case). Note that it is undecidable in general whether, for a given EFSM M and its state sk, there exists a test case for leading M to trace a transition sequence from the initial state to the state sk. However, if it exists, we can find it using the similar way to our test case derivation method which we propose in Section 4.1. For example, "a?0/e!0, a?1/e!0" is a test case starting from the initial state s1 and ending to the state s3 in Fig. 1. Since we assume the reset events for all states, this assumption means that the EFSM is strongly connected.

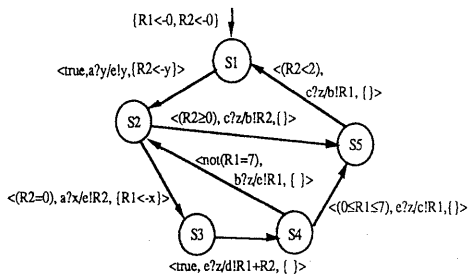

Fig. 1 An Example of Our EFSM/in Model

[Example 2.1]

Fig. 1 is an example of our EFSM/in model. There are five states and two registers. The state s1 is the initial state. The initial values of the registers R1 and R2 are both 0. Suppose that the current state is s2 and the current values of the registers R1 and R2 are both 0. Then, since the transition condition (R2=0) at s2 → s3 is true, the transition s2 → s3 is executable. If the input event "a?x" is executed, then the current value "0" of the register R2 is emitted to the gate "e" and the input value "x" is substituted to the register R1. []

3. Application of General UIO Methods for EFSM's and Their Problems

Some methods have been proposed for generating UIO sequences for FSM models (for survey, [BoUy 91]). In general, such methods cannot apply for EFSM models. In this section, first, we explain the reason.

Fig. 2 is the FSM which is obtained from the EFSM in Fig. 1 by ignoring the register values and I/O data. For the FSM, we can generate a UIO sequence for identifying each state (see Fig. 2). Here, such UIO sequences are also treated as the UIO sequences for our EFSM/in model. For example, "e/c" is a UIO sequence for identifying the state s4. The I/O event "e/c" corresponds to the transition s4 → s5. Let us consider to apply the UIO sequence for the EFSM in Fig. 1. In Fig. 1, in order to execute the event "e?x/c!R1", the current value of the register R1 must satisfy its transition condition "(0≤R1≤7)". However, this value is determined, for example, when the transition s2 → s3 (<(R2=0), a?x/e!R2,{R1←x}>) is executed since the input value "x" is substituted to the register R1. Therefore, in order to execute the transition s4 → s5, the value satisfying the condition "(0≤x≤7)" must be input when the transition s2 → s3 (I/O event a?x/e!R2) is executed. In order to execute the transition s2 → s3 (I/O event a?x/e!R2), the transition condition "(R2=0)" must hold. The value of R2 is determined when the transition s1 → s2 (<true,a?y/e!y,{R2←y}>) is executed where the input value "y" is substituted to R2. This input value "y" must be zero in order to execute the transition s2 → s3. That is, if we consider data values, then the event sequence "a?y/e!y, a?x/e!R2, e?z/d!R1+R2, e?z/c!R1" needs to be executed and the condition "(0≤x=R1≤7) and (y=R2=0)" for the input data must hold. The event sequence corresponds to the transition sequence "s1 → s2 → s3 → s4 → s5". By substituting "x" and "y" into "R1" and "R2" of the event sequence "a?y/e!y,a?x/e!R2,e?z/d!R1+R2,e?z/c!R1", respectively, the following event sequence is obtained.

a?y/e!y, a?x/e!y, e?z/d!x+y, e?z/c!x

Condition : (0≤x≤7) and (y=0)

For example, since <x,y,z>=<1,0,0> is a solution satisfying the condition "(0≤x≤7) and (y=0)", the event sequence "a?0/e!0, a?1/e!0, e?0/d!1, e?0/c!1" is a test case with data values to execute the UIO sequence "e?x/c!R1" (transition *s4* → *s5*) described above.

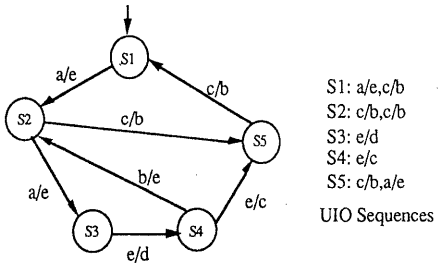

Fig. 2 FSM Transformed From EFSM in Fig. 1

In this paper, we call the event sequence "a?y/e!y, a?x/e!y, e?z/d!x+y, e?z/c!x" an *extended UIO (E-UIO) sequence for identifying the state s4*. The condition "(0≤x≤7) and (y=0)" is called the *executability condition* for the E-UIO sequence. We also call the event sequence "a?0/e!0, a?1/e!0, e?0/d!1, e?0/c!1" an *extended UIO (E-UIO) sequence with data values for identifying the state s4*. Hereafter, "e?0/c!1" is called a *UIO sequence with data values*. And the event sequence "a?y/e!y,a?x/e!y,e?z/d!x+y" ("a?0/e!0, a?1/e!0, e?0/d!1") is called a *preceding UIO sequence for the UIO sequence* "e?z/c!x" ("e?0/c!1"). Each E-UIO sequence (with data values) consists of a pair of a preceding UIO sequence (with data values) and a UIO sequence (with data values). From the above discussion, we must find not only UIO sequences with data values but also their preceding UIO sequences with data values in order to identify each state in a given EFSM. This is a difference for test case generations between FSM models and EFSM models. This problem has been also discussed in [UrYa 91, WaLi 92].

4. Test Case Generation for Our EFSM Model

In this section, first, we define the E-UIO sequences formally. Here, we assume the number of input variables of each transition is one for simplicity of discussion. However, the properties presented in this section also hold for the case that the number of the input variables is more than one.

[Definition 4.1]

Let UIOj denote a UIO sequence "$s_j \xrightarrow{tr_j} \cdots \xrightarrow{tr_{(h-2)}} s_{h-1} \xrightarrow{tr_{(h-1)}} s_h$" for a state s_j of an EFSM M. And let $TR_{1\text{-}h}$ denote a transition sequence "$s_1 \xrightarrow{tr_1} \cdots \xrightarrow{tr_{(j-1)}} s_j \xrightarrow{tr_j} \cdots \xrightarrow{tr_{(h-2)}} s_{h-1} \xrightarrow{tr_{(h-1)}} s_h$" starting from a state s_1 and ending to the state s_h via states $s_2,\ldots,s_j,\ldots,s_h$ where $TR_{1\text{-}h}$ contains UIO_j as the tail sequence. Here, we assume that each transition tr_k is "s_k-<C_k,a_k?x_k/b_k!E_k,RD_k>->s_{k+1}". If there exists an input event sequence "a_1?n_1,a_2?n_2,..., a_{h-1}?n_{h-1}" (n_1, ..., n_{h-1} are concrete data values) such that the input event sequence is executable for any given registers' values $u_1,\ldots,u_n$ at the state s_1 of M, then we say that the transition sequence $TR_{1\text{-}h}$ is an E-UIO sequence for the state s_j which corresponds to the given UIO sequence UIO_j. For those input values and registers' values u_1, ..., u_n at the starting state s_1, the corresponding output values are defined uniquely. We say that the transition sequence with data values $TC_{1\text{-}h}$ = "a_1?n_1/b_1!v_1, ..., a_{h-1}?n_{h-1}/b_{h-1}!v_{h-1}" is an E-UIO sequence with data values corresponding to $TR_{1\text{-}h}$. []

Note that the input values $n_1,\ldots,n_h$ do not depend on the registers' values $u_1,\ldots,u_n$, and that we call a given transition sequence as an E-UIO sequence if and only if an E-UIO sequence with data values can be generated for any register value at the starting state of the transition sequence. Therefore, we do not treat a transition sequence as an E-UIO sequence if the corresponding E-UIO sequence with data values can be generated only for some specific registers' values. This condition is used when the E-UIO sequences are connected as a test suite. For example, since we assume that a given EFSM M is strongly connected, there is a test case α leading the EFSM M to trace from the initial state s_{init} to the state s_1. The values of all registers when the test case α is executed are also determined. Using those values, an E-UIO sequence $TC_{1\text{-}h}$ with data values starting from the state s_1 is obtained. By connecting the test case α and the E-UIO sequence $TC_{1\text{-}h}$ with data values, we can check whether the state s_j is correctly implemented in a given IUT.

[Procedure 4.1]

we can decide whether a transition sequence $TR_{1\text{-}h}$ is an E-UIO sequence of an EFSM M using the following procedure.

Input : a transition sequence "$s_1 \xrightarrow{tr_1} \cdots \xrightarrow{tr_{(j-1)}} s_j \xrightarrow{tr_j} \cdots \xrightarrow{tr_{(h-2)}} s_{h-1} \xrightarrow{tr_{(h-1)}} s_h$" where we assume that each transition tr_k is "s_k-<C_k,a_k?x_k/b_k!E_k, RD_k>->s_{k+1}".

1. Let *Cond* denote a Boolean variable, and let *p* denote an integer variable.
   Cond ← true *p* ← h
2. While *p* is not equal to 1, repeat the following three steps.
   2.1 If there is "R_j← $f_j(x_{p-1})$" in RD_{p-1}, then replace all "R_j" in *Cond* with $f_j(x)$. If the same variable name "x_{p-1}" has been already used in *Cond*, then use a new variable, and replace all "R_j" in *Cond* with $f_j(x')$ where x' denotes a new variable name.
   2.2 *Cond* ← *Cond* and C_{p-1}
   2.3 *p* ← *p-1*

3. $\Phi TR1\text{-}h \leftarrow Cond.$
4. If (1) $\Phi TR1\text{-}h$ does not contain register variables, (2) $TR1\text{-}h$ contains a UIO sequence for a state sj of an EFSM M as the tail sequence and (3) it is satisfiable, i.e., there exist concrete data values for the input variables $x1,...,xh\text{-}1$ such that the value of $\Phi TR1\text{-}h$ is true, then we decide that $TR1\text{-}h$ is an E-UIO sequence for the state sj. If one of three conditions does not hold, then we decide that $TR1\text{-}h$ is not an E-UIO sequence. []

Here, we say that $\Phi TR1\text{-}h$ is the *executability condition* for the transition sequence $TR1\text{-}h$. The executability condition $\Phi TR1\text{-}h$ denotes the condition which the EFSM must satisfy to execute the given transition sequence $TR1\text{-}h$. For example, if the transiion sequence β="s1 → s2 →s3 →s4 → s5" in Fig.1 is given, the executability condition $\Phi\beta$ is "(0≤x≤7) and (y=0)". If the transition sequence β'="s3→s4→s5" in Fig.1 is given, the executability condition $\Phi\beta'$ is "(0≤R1≤7)". For a given transition sequence β, if $\Phi\beta$ contains register variables, then the executability of the transition sequence β depends on the register values which have been defined in a preceding transition sequence. However, if $\Phi\beta$ does not contain register variables, then the executability of the transition sequence β depends on only the values of the input variables used in β. If $\Phi\beta$ is satisfiable, then we can generate concrete input data values of a test case tracing the transition sequence β from a solution of $\Phi\beta$. The input values of the variables not contained in $\Phi\beta$ can be arbitrarily determined. The output data values are determined depending on those input values and register values at the starting state of β.

[Lemma 4.1]

For a given transition sequence β, the executability condition $\Phi\beta$ is a P-sentence.

(Proof) From the definition, all transition conditions are P-sentences. In Procedure 4.1, some register variables, say Rj, may be replaced by fj(x) based on the substitution statements such as "Rj← fj(x)". However, fj(x) is a P-term. Therefore, the replaced expressions are also P-sentences. Then, this lemma holds. []

A logical expression containing only ∃, ∀ and P-sentences is called a Presburger sentence. It is known that it is decidable whether a given Presburger sentence is true or not [HoUl 79].

[Property 4.1]

For any executability condition $\Phi\beta(x1,...,xk)$ where $x1,...,xk$ denote the variables appeared in $\Phi\beta$, it is decidable whether $\Phi\beta(x1,...,xk)$ is satisfiable or not. And if $\Phi\beta(x1,..,xk)$ is satisfiable, a solution $<n1,...,nk>$ satisfying $\Phi\beta(x1,...,xk)$ can be generated mechanically. []

A solution can be generated through integer linear programming [HiBo 94].

Let us consider the transition sequence (E-UIO sequence) β="s1 →s2 → s3→ s4 →s5" in Fig.1. For this transition sequence β, $\Phi\beta(x,y)$ is "(0≤x≤7) and (y=0)". Since the condition "(0≤x≤7) and (y=0)" is a P-sentence, a solution, say <x,y>=<1,0>, is obtained mechanically. Since the variable "z" in the transition sequence "a?y/e!y, a?x/e!y, e?z/d!x+y, e?z/c!x" is not appeared in the executability condition $\Phi\beta(x,y)$, any value, say "0", may be given as the value of "z". Using those values, we can generate an E-UIO sequence "a?0/e!0, a?1/e!0, e?0/d!1, e?0/c!1" with data values for identifying the state s4. Since the starting state of the E-UIO is the initial state s1, we can check whether the state s4 is correctly implemented in a given IUT by giving the input event sequence of this E-UIO from the initial state and observing its output event sequence.

We cannot determine the values of outputs directly from $\Phi\beta$ if the output values depend on the registers values at the starting state. Let us generate an E-UIO sequence for identifying the state s3 in Fig.1. A UIO sequence for the state s3 is "e?z/d!R1+R2". By giving the UIO sequence UIO3 to Procedure 4.1, the executability condition $\Phi UIO3$="true" is obtained. Since $\Phi UIO3$ does not contains the register variables, UIO3 is also an E-UIO sequence for the state s3. In this case, any integer value, say "1", may be given as the input value "z". However, the output value "R1+R2" is determined depending on the transition sequence which has been executed before executing this E-UIO sequence. For example, in order to identify the state s3, we must lead a given IUT to trace the initial state s1 to the state s3 and then we must give the E-UIO sequence "e?z/d!R1+R2". There exists a test case, for example "a?0/e!0, a?1/e!0", for leading the EFSM to trace the initial state s1 to the state s3. If the test case "a?0/e!0, a?1/e!0" is executed from the initial state, then the values of the registers R1 and R2 become 1 and 0, respectively. Then, the value of "R1+R2" becomes "1" when the EFSM enters the state s3. That is, "e?1/d!1" is an E-UIO sequence with data values for identifying the state s3. By connecting "a?0/e!0, a?1/e!0" and "e?1/d!1", we can check whether the state s3 is correctly implemented in a given IUT.

In both cases, for any executability condition $\Phi\beta$ without register variables, we can generate the transition sequences with data values satisfying the executability condition $\Phi\beta$ mechanically if all registers' values at the starting state of the transition sequence β are given and $\Phi\beta$ is satisfiable.

State	E-UIO	Φ_{UIOj}	Transition	E-UIO with data values
s_1	a?y/e!y,c?z/b!y	(y≥0)	$s_1 \rightarrow s_2 \rightarrow s_5$	a?1/e!1,c?4/b!1
s_2	a?y/e!y,c?z'/b!y, c?z/b!R_1	(y≥0) and (y<2)	$s_1 \rightarrow s_2 \rightarrow s_5 \rightarrow s_1$	a?1/e!1,c?4/b!1, c?3/b!0
s_3	c?z/d!R_1+R_2	true	$s_3 \rightarrow s_4$	e?1/d!1
s_4	a?y/e!y, a?x/e!y, c?z'/d!x+y,e?z/c!x	(0≤x≤7) and (y=0)	$s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow s_5$	a?0/e!0, a?1/e!0, e?0/d!1, e?0/c!1
s_5	a?y'/e!y',c?z'/b!y', c?z/b!R_1,a?y/e!y	(y≥0) and (y<2)	$s_1 \rightarrow s_2 \rightarrow s_5 \rightarrow s_1 \rightarrow s_2$	a?1/e!1,c?4/b!1, c?3/b!0, a?0/e!0

Fig. 3 E-UIO Sequences

For the EFSM in Fig. 1, E-UIO sequences are given in Fig. 3. Let E-UIO$_k$ denote the E-UIO sequence for identifying the state s_k in Fig. 3. Integer "0" is given as the values of the registers "R_1" in E-UIO$_2$ and E-UIO$_4$ since the initial register value of "R_1" at the initial state s_1 is "0". In E-UIO$_3$, integer "1" is given as the value of "R_1+R_2" since the value of "R_1+R_2" becomes "1" when the EFSM enters the state s_3 after executing a test case "a?0/e!0, a?1/e!0" from the initial state.

[Theorem 4.1]

For a given UIO sequence UIO$_j$ for identifying a state s_j of an EFSM M, if there exists an E-UIO sequence with data values containing UIO$_j$ as the tail sequence, then we can generate it mechanically.

(Proof) Using breadth first search, we can enumerate every transition sequence β containing UIO$_j$ as the tail sequence. If (1) $\Phi\beta$ does not contain register variables and (2) it is satisfiable, then β is an E-UIO sequence. From Property 4.1, it is decidable whether $\Phi\beta$ is satisfiable or not. If it is satisfiable, then concrete data values for β are obtained from the solution satisfying $\Phi\beta$ and the registers' values at the starting state of β mechanically. If there exists an E-UIO sequence, such an E-UIO sequence can be found eventually. []

If a state s_i of an EFSM M is reachable from the initial state, then we can find a test case α leading M from the initial state to the state s_i using the above technique. Since, in this paper, we assume that all states of an EFSM M are reachable from the initial state, such test cases can be generated mechanically.

Since our EFSM/in model can simulate Turing machines, it is undecidable whether there exists an E-UIO sequence with data values for a given UIO sequence. There may exist a case that there does not exist an E-UIO sequence with data values for a given UIO sequence. For such a case, the above breadth first search continues infinitely. Therefore, we must decide by ourselves when we stop to extend the transitions.

5. Conclusion

In this paper, we have proposed a new technique to generate E-UIO sequences mechanically for a restricted class of EFSM's. We have been developing a tester for generating E-UIO sequences. In order to generate each E-UIO sequence, some integer linear programming problems must be solved. In our experiences, the transition conditions are very simple for most practical examples. Such an observation is also shown in [ChZh 94]. From those results, we can conclude that the number of the constraints of integer linear programming problems which we must solve is at most 10 or 20. It takes about 20 second for our tester to solve the integer linear programming problems whose constraints' numbers are 30 using a SUN SPARCstation ELC in most cases. One of the future works is to show the usefulness of our approach using the tester.

References

[BoUy 91] B. S. Bosik and M. U. Uyar : "Finite State Machine Based Formal Methods in Protocol Conformance Testing", Computer Networks ISDN Systems, 22, pp. 7-33, 1991.

[ChZh 94] S. T. Chanson and J. Zhu : "Automatic Protocol Test Suite Derivation", IEEE INFOCOM'94 (to appear).

[HiBo 94] T. Higashino and G. v. Bochmann : "Automatic Analysis and Test Case Derivation for a Restricted Class of LOTOS Expressions with Data Parameters", IEEE Trans. on Soft. Eng., 20, 1, pp. 29-42, 1994.

[HoUl 79] J. E. Hopcroft and J. D. Ullman : "Introduction to Automata Theory, Languages, and Computation", Addison-Weslay, 1979.

[Sari 87] B. Sarikaya, G. v. Bochmann and E. Cerny : "A Test Design Methodology for Protocol Testing", IEEE Trans. on Soft. Eng., pp. 518-531, May 1987.

[UrYa 91] H. Ural and B. Yang : "A Test Sequence Selection Method for Protocol Testing", IEEE Trans. on Comm., 39, 4, pp.514-523, 1991.

[WaLi 92] C.-J. Wang and M. T. Liu : "A Test Suite Generation method for Extended Finite State Machines using Axiomatic Semantics Approach", Proc. 12th IFIP Symp. on Protocol Specification, Testing and Verification, pp. 29-43, 1992.