7

An Optimal State Identification Method Using a Dynamic–Programming–Based Approach for Protocol Testing

Rong S. Lin*§ and Maria C. Yuang*

*Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan

§Telecommunication Laboratories Chung–Li, Taiwan

Abstract

The Unique Input/Output (UIO) sequence has been regarded as an efficient technique for state identification for protocol testing. Unfortunately, it has been shown that some states in a protocol may possess no UIO sequences. Another input/output sequence called a signature can be generated as a substitute for a state without a UIO sequence. The existing signature technique performs state identification by distinguishing a given state from another single state at a time. The limitation is that it cannot assure a minimum–length signature. Moreover, a recently–proposed method, called the Partial UIO (PUIO) sequence, distinguishes a state from a nonempty proper subset of states at a time. The goal of the paper is to construct a minimum–length signature by selecting appropriate PUIO sequences from the set of all PUIO sequences. This paper first transforms the problem into a Minimum–Cost Pattern Covering Problem (MCPCP), where the pattern is the set of the remaining states from which the state under consideration is to be distinguished. To solve the MCPCP, the paper presents a dynamic–programming–based algorithm with three reduction rules and three termination rules. The reduction rules are used to reduce the original problem to simpler subproblems, and the termination rules are used to terminate the reduction process. The paper also discusses the time complexity of the algorithm. Consequently, an optimal–length signature can be efficiently constructed.

1. Introduction

State identification is an important subject in conformance testing. Since an Implementation Under Test (IUT) is typically regarded as a *black box*, its internal states are invisible. Hence in order to indirectly recognize a state of the IUT, testing is performed by applying a sequence of inputs to the IUT and verifying the sequence of outputs in response. Concisely, state identification is related to the problem of deriving an input/output sequence from the protocol specification. The derived sequence can be used to determine if the IUT is currently in a particular state.

Much research work on state identification has been undertaken using various methods [1,2,3,4,5]. The most effective one is the Unique Input/Output (UIO) method. The UIO sequence for a state is an input/output behavior not exhibited by any other state in the protocol. Owing to the fault detection and incurring shorter length than other methods, UIO sequences have been widely applied to the test–case generation for conformance testing. However, the major limitation of the UIO method is that some states in a protocol may possess no UIO sequences. To overcome this difficulty, a technique called the *signature* [4] has been proposed. The signature attempts to distinguish a state from other states one at a time. The signature for a given state S_i can be expressed as follows:

$$\text{signature}(S_i) = IO(S_i, S_1) \,||\, T_1 \,||\, IO(S_i, S_2) \,||\, T_2 \,||\, \ldots \,||\, IO(S_i, S_{i-1}) \,||\, T_{i-1} \,||\, IO(S_i, S_{i+1}) \,||\, T_{i+1} \,||\, \ldots \,||\, IO(S_i, S_n),$$

T. Mizuno et al. (eds.), *Protocol Test Systems*

where | | is the concatenation operation; $IO(S_i, S_j)$ $(S_i \neq S_j)$ is the minimum–length I/O subsequence (starting from S_i) used to distinguish S_i and S_j; and T_j is the transfer I/O sequence leading the ending state of $IO(S_i, S_j)$ back to S_i. In addition, instead of concatenating all I/O subsequences, another method of directly employing the set of I/O subsequences (called the *signature set* [6]) has been proposed to improve the fault detection capability. In practice, both the signature and signature–set methods can be derived in polynomial time [7] and are usually regarded as substitutes for UIO sequences.

The existing signature (or signature set) technique attempts to distinguish a state from the remaining states one at a time. Therefore, it does not generate the minimum–length I/O sequence, compared to a technique that distinguishes a state from multiple states at a time. A notion called a Partial UIO (PUIO) sequence [8] was then introduced. A PUIO sequence is an input/output sequence by which a state without a UIO sequence can be distinguished from a nonempty proper subset of the states in the protocol. An algorithm of searching all PUIO sequences for a given state has also been proposed. As a result, a *minimum–length signature* of a state can be generated by selecting appropriate PUIO sequences of the state. However, selecting the required PUIO sequences is not a trivial problem.

This paper thus transforms the problem into a Minimum–Cost Pattern Covering Problem (MCPCP), where the pattern is the set of the remaining states from which the state under consideration is to be distinguished. To solve the MCPCP, the paper presents a dynamic–programming–based algorithm with three reduction rules and three termination rules. The reduction rules are used to reduce the original problem to simpler subproblems, and the termination rules are used to terminate the reduction process. The paper also discusses the time complexity of the algorithm. Consequently, an optimal–length signature can be efficiently constructed.

The paper is organized as follows. Section 2 defines the MCPCP and addresses the association of the problem with the state identification problem. Section 3 presents the dynamic–programming–based algorithm and its reduction and termination rules. The time complexity of the algorithm is also discussed in the section. Finally, Section 4 concludes the paper.

2. Minimum–Cost Pattern Covering Problem (MCPCP)

Definition: Minimum–Cost Pattern Covering Problem (MCPCP)

Assume that there are n types of items $a_1, a_2, \ldots a_n$, and a finite set **U** of m packages $A_1, A_2, \ldots A_m$. Each package A_i is associated with a cost, denoted as $cost(A_i)$, and a set of items $\{a_{i1}, a_{i2}, \ldots a_{ik}\}$. In addition, no package contains all n items (i.e., the size of $A_i < n$). The MCPCP is to determine a package subset **U'** of **U**, such that the union of all packages A_i in **U'** is equal to $\{a_1, a_2, \ldots a_n\}$ and the sum of all $cost(A_i)$ is minimized. The set of items $\{a_1, a_2, \ldots a_n\}$ expected to be collected is hereinafter referred to as the *target pattern*, and a package in the set $\mathbf{U}=\{A_1, A_2, \ldots A_m\}$ that can be selected is referred to as an *accessible package*.

The relationship between the minimum–length signature and the MCPCP is realized as follows. Since each PUIO sequence can be used to distinguish a state from a nonempty proper subset of states, a PUIO sequence can be considered as a package, the states that a PUIO sequence can distinguish are the items contained in the package, and the length of the PUIO sequence corresponds to the cost of the package. All PUIO sequences generated for a given state thus correspond to accessible packages. Since the signature is used to distinguish a state from all remaining states in the protocol, the target pattern becomes the set of the states in the protocol except the state under consideration. Therefore, once the minimum–cost package subset **U'** is determined, the minimum–length signature is also obtained.

To ensure that there always exists a solution to the MCPCP, we make two additional assumptions concerning **U**. First, assume that there always exists a **V** which is a subset of **U**, such that **V** is a special package set { $B_1, B_2, \ldots B_n$}, where $B_i = \{a_i\}$ (i.e., the i–th package B_i contains the i–th item only). Therefore, the worst solution of the MCPCP is **V** with the cost being the sum of all cost(B_i). Any package B_i in **V**, referred to as a *basic package*, dose exist because there always exists an input/output sequence that can distinguish a state from any state in a protocol (if the protocol is modeled as a reduced Finite State Machine [4]). Notice that, solution **V** corresponds to the traditional signature which distinguishes a state from the remaining states one by one. Second, the cost of a package containing more than one item (e.g., $A_i=\{a_j, a_k\}$) is always less than the sum of the costs of the corresponding basic packages (i.e., cost(A_i) < cost(B_j) + cost(B_k)).

For solving the MCPCP, a brute–force algorithm can be considered as: each package is either selected or not. Since there are a total of m accessible packages and each of which requires O(n) time to determine the items it contains, the time complexity is thus $O(n*2^m)$. Moreover, since there are n items in total and each item a_i is either contained or not contained in a package, the number of possible accessible packages m can be associated with n by 2^n (i.e., $O(m) \approx O(2^n)$). Therefore, the time complexity of the brute–force algorithm becomes $O(n*2^{2^n})$. This algorithm is obviously impractical.

3. Dynamic–Programming–Based Algorithm

Before our algorithm is presented, three reduction and three termination rules are first derived. The algorithm and its complexity are presented afterwards.

3.1 Reduction and Termination Rules

For the convenience of illustration, all items and packages are ordered. Let P(i, $\{a_{i1}, a_{i2}, \ldots a_{ik}\}$) (where i is the largest index of accessible packages being selected and $\{a_{i1}, a_{i2}. \ldots a_{ik}\}$ is the target pattern being collected) denote the minimum cost to collect items $a_{i1}, a_{i2}, \ldots a_{ik}$, from packages $A_1, A_2, \ldots A_i$. Consequently, solving the MCPCP corresponds to solving P(m, $\{a_1, a_2, \ldots a_n\}$).

To solve P(i, $\{a_{i1}, \ldots a_{ik}\}$), one can easily get P(i, $\{a_{i1}, \ldots a_{ik}\}$) = P(i–1, $\{a_{i1}, \ldots a_{ik}\}$) if the last package A_i is not selected. Otherwise, if A_i is selected, P(i, $\{a_{i1}, \ldots a_{ik}\}$) = cost($A_i$) + P(i–1, $\{a_{w1}, \ldots a_{ws}\}$), where the set $\{a_{w1}, \ldots a_{ws}\}$ contains the remaining items after the items included in package A_i have been removed. This resolves our first reduction rule (R–rule 1) as:

- **R–rule 1**: P(i, $\{a_{i1}, \ldots a_{ik}\}$) = min(P(i–1, $\{a_{i1}, \ldots a_{ik}\}$), cost($A_i$)+P(i–1, $\{a_{w1}, \ldots a_{ws}\}$)).

Notice that this rule reduces problem P(i, $\{a_{i1}, \ldots a_{ik}\}$) to two subproblems in which the number of accessible packages and the size of the target pattern are respectively reduced.

- **R–rule 2**: If the current observed package A_i does not contain any item in the target pattern $\{a_{i1}, \ldots a_{ik}\}$, package A_i is definitely not selected, i.e., P(i, $\{a_{i1}, \ldots a_{ik}\}$) = P(i–1, $\{a_{i1}, \ldots a_{ik}\}$).

To illustrate R–rule 3, an array, called FIRST, with size n is employed. FIRST[i] denotes the index of the first package from A_{n+1} to A_m in which item a_i is included. For example as shown in Figure 2, FIRST[5] = 8, since item a_5 first appears in A_8 (from A_6 to A_{10}).

- **R–rule 3**: Given an MCPCP P(i, $\{a_{w1}, a_{w2}, \ldots a_{wk}, a_k\}$), where a_k is the last item of the required target pattern, if FIRST(k)>i, then P(i, $\{a_{w1}, a_{w2}, \ldots a_{wk}, a_k\}$) = cost($A_k$)+P(i, $\{a_{w1}, a_{w2}, \ldots a_{wk}\}$), since the remaining accessible packages do not contain item a_k except the basic package A_k.

Briefly, there are a total of four types of reductions derived on the basis of R–rules 1, 2, and 3, as shown in Figure 1. The first two types, R–rules 1.a and 1.b, are based on R–rule 1. There are two possibilities for package A_i. If A_i contains the last item of the target pattern, say a_j, the outcome of the reduction is shown in R–rule 1.a. Otherwise, the outcome is shown in R–rule 1.b. Notice that the size of the target pattern is decremented by at least one after A_i is selected, regardless of whether or not A_i contains item a_j. For R–rule 2, the original problem is directly replaced by a new subproblem with the number of accessible packages decremented by one. For R–rule 3, the original problem is also directly replaced by a new subproblem with the last item removed from the target pattern.

In addition to the reduction rules given above, three termination rules (T–rules 1 to 3) are presented next.

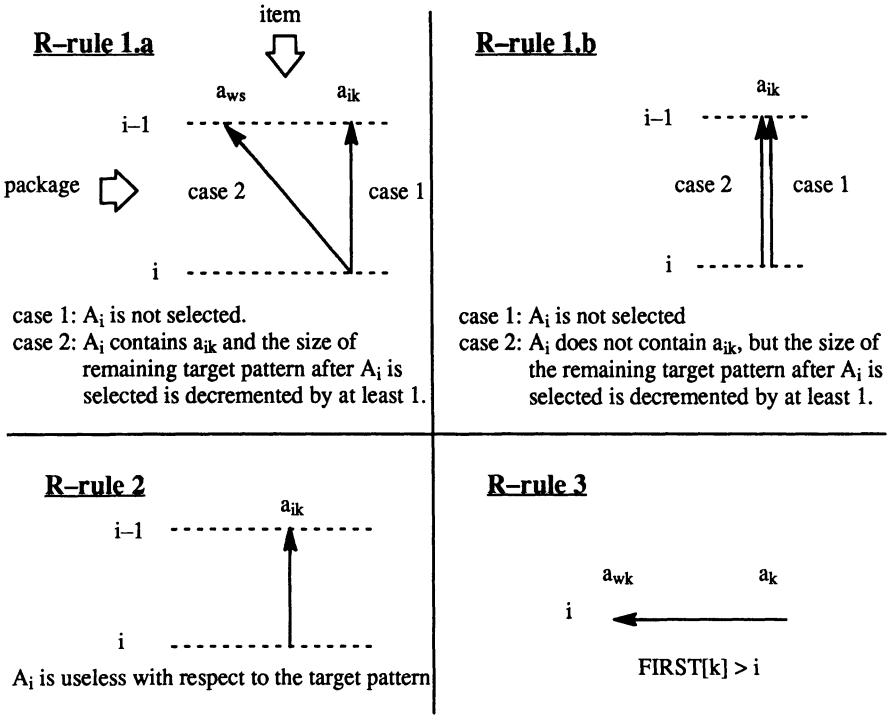

Figure 1. Summary of reduction rules.

- **T–rule 1**: If the target pattern is an empty set (i.e., no more items are dealt with), then $P(i, \{\})=0$.
- **T–rule 2**: If the target pattern contains just one item, say a_i, the solution can be immediately derived as: $P(i, \{a_i\})=cost(A_i)$, where A_i is the basic package containing item a_i only.

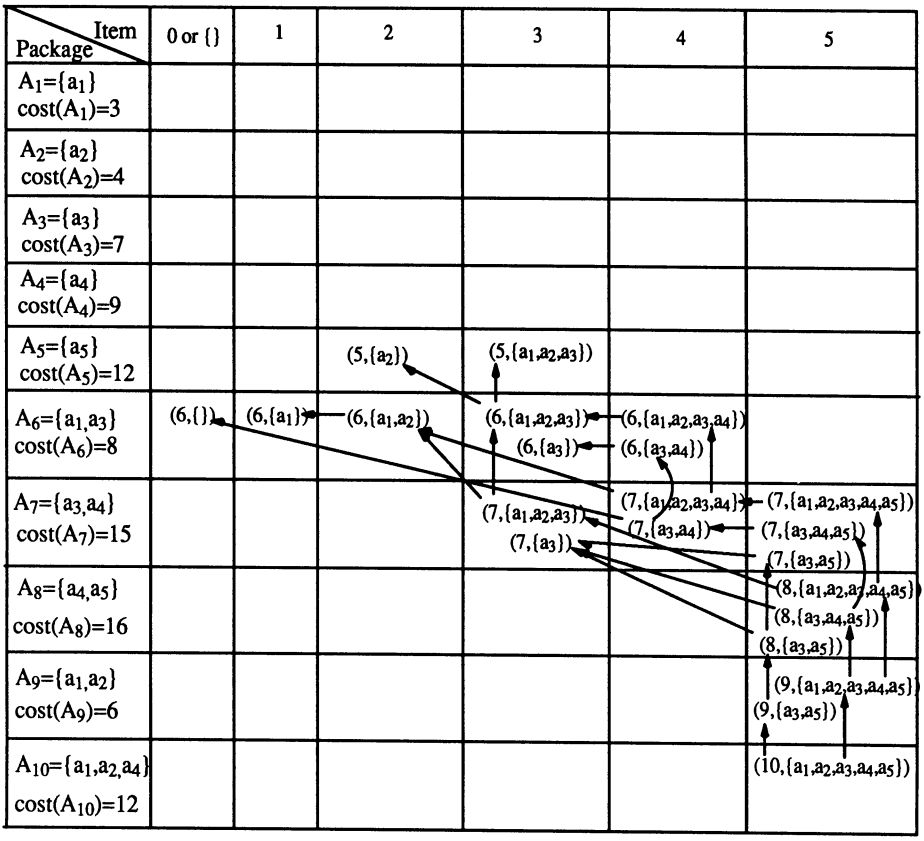

Figure 2. Reduction and termination of an MCPCP.

- **T–rule 3**: If the number of accessible packages i is equal to (not possible to be smaller) the number of items n, which implies the accessible packages are confined to the basic packages A_1, ... A_n, the solution can be immediately derived as: P(i, $\{a_{i1}, a_{i2}, \ldots a_{ik}\}$) = cost($A_{i1}$)+cost($A_{i2}$)+ ... +cost($A_{ik}$), where each package contains one item only.

To illustrate the concept of reduction, we show an example in Figure 2. In this example, five items a_1, ... a_5 are expected to be collected. Ten packages, including five basic packages A_1 to A_5 and other five packages A_6 to A_{10}, are given. All subproblems produced are shown in the two–dimensional table with 10*6 entries (one additional column for the empty target pattern). A problem, dealing with accessible packages up to i and the last item of the target pattern a_j, is placed at entry[i,j]. The target problem P(10, $\{a_1, \ldots a_5\}$) is thus located in the down right corner (entry[10,5]).

Initially, the last package $A_{10}=\{a_1, a_2, a_4\}$ is examined. According to R–rule 1, P(10, $\{a_1, \ldots a_5\}$) = min (P(9, $\{a_1, \ldots a_5\}$), cost(A_{10})+P(9, $\{a_3, a_5\}$)). Two new subproblems P(9, $\{a_1, \ldots a_5\}$) and P(9,

$\{a_3,a_5\}$) are produced and accordingly placed at entry[9,5]. According to R–rule 2, since the package $A_9=\{a_1, a_2\}$ contains no items in the target pattern $\{a_3, a_5\}$, $P(9, \{a_3, a_5\}) = P(8, \{a_3, a_5\})$. According to R–rule 3, since FIRST(5)>7, $P(7, \{a_1, \ldots a_5\}) = cost(A_5)+P(7, \{a_1,a_2,a_3,a_4\})$. A new subproblem $P(7, \{a_1,a_2,a_3,a_4\})$ is produced and placed at entry[7, 4]. The same reduction procedure is repeated until all subproblems are solved and terminated according to the termination rules (T–rules 1 to 3).

3.2 Algorithm

A minimum cost of the MCPCP can be attained based on the following algorithm.

Algorithm: The minimum cost of the MCPCP.
Input: In an MCPCP, there are n items $a_1, \ldots a_n$; m accessible packages $A_1, \ldots A_n, A_{n+1}, \ldots A_m$, where A_1 to A_n are the basic packages and A_{n+1} to A_m are the packages containing more than one item; and an array FIRST of size n.
Output: The minimum cost $P(m, \{a_1, \ldots a_n\})$.

Step 1: Sequentially search the accessible packages from A_{n+1} to A_m to construct array FIRST.
Step 2: By applying R–rules 1, 2, and 3, reiteratively reduce the target problem $P(m, \{a_1, \ldots a_n\})$ to subproblems until one of the three T–rules is satisfied. A series of ordered subproblems will be derived.
Step 3: By reversing the order of the subproblems produced, derive the solution of each subproblem based on the previously solved subproblems. The target problem $P(m, \{a_1, \ldots a_n\})$ can then be solved.

The second and third steps of the algorithm are illustrated in Figures 2 and 3, respectively. In Figure 3, For example, $P(6, \{a_1,a_2,a_3\}) = \min(P(5, \{a_1,a_2,a_3\}), cost(A_6)+P(5, \{a_2\})) = \min(cost(A_1) + cost(A_2) + cost(A_3), cost(A_6) + cost(A_2)) = \min(14, 8+4) = 12$. Similar calculation is performed until the cost of the target pattern $P(10, \{a_1, \ldots a_5\})$ is determined. Consequently, the optimal solution for the MCPCP $P(10, \{a_1, \ldots a_5\})$ is to select packages A_2, A_6 and A_8 with cost 28.

3.3 Time Complexity

The time complexity of the algorithm is obviously dominated by the total number of subproblems produced. Let us first consider the maximum number of subproblems placed at a table *entry*. Since the target pattern of the subproblems placed at entry[i, j] must contain item a_j, but with or without item a_k ($1 \le k \le j-1$) contained, the number of subproblems possibly placed at entry[i,j] is thus at most 2^{j-1}. Therefore, the maximum number of subproblems of an entry is 2^{n-1}. Second, let us determine the maximum number of subproblems placed in a *row*. Since the maximum number of subproblems at entry[i,1] is 2^0, that at entry[i,2] is 2^1, ... , and that at entry[i,n] is 2^{n-1}, the maximum number of subproblems placed in row i is thus equal to $2^0+2^1+ \ldots +2^{n-1} = 2^n-1 = O(2^n)$.

Considering the worst case, i.e., the R–rule 1, which creates two new subproblems at a time, the number of subproblems in a row will be doubled from 1 to 2 to 4 and so on until the maximum 2^n is reached. Furthermore, the number of rows with maximum 2^n subproblems is at most n, since the maximum size of the target pattern is n and the size of the target pattern is decremented by at least one during each reduction, as shown in R–rules 1.a and 1.b in Figure 1. Therefore, the total number of the subproblems produced becomes $O(n*2^n)$.

The time complexity of the proposed algorithm can now be analyzed as follows. Step 1 requires O(m*n) time, since O(m) packages will be sequentially checked and at most O(n) items are included in each package. Step 2 requires the time needed for each reduction operation multiplied by the total

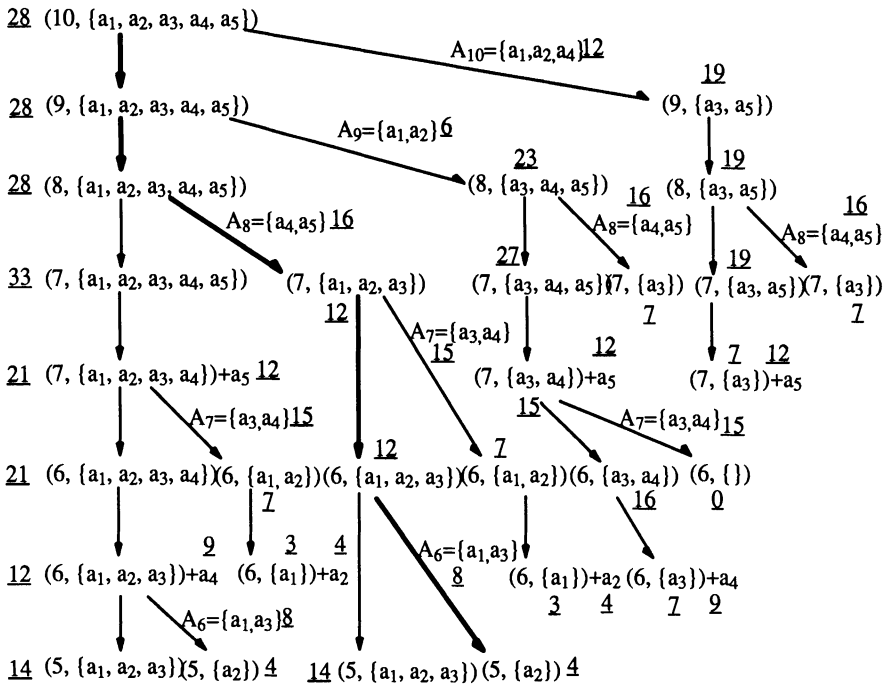

Figure 3. The cost computation process (the 3rd step) for the example in Figure 2.

number of subproblems derived. Since each accessible package will be compared with the target pattern, O(n) time is required during the reduction. Step 2 thus requires $O(n*n*2^n)$ time. Step 3 requires the time needed to solve each subproblem multiplied by the total number of subproblems derived. Since each subproblem can be solved based on the previously solved subproblems, a constant time is enough. Step 3 thus requires $O(n*2^n)$ time. The aggregate time complexity becomes $O(m*n) + O(n*n*2^n) + O(n*2^n) = O(n^2*2^n)$ (since $m \leq 2^n$). Finally, as was previously stated, the number of possible accessible packages is at most 2^n, i.e., $O(m) \approx O(2^n)$, so the time complexity $O(n^2*2^n)$ of the proposed algorithm becomes $O(n^2*m)$. Compared to the brute–force algorithm, our new algorithm reduces the time complexity from $O(n*2^m)$ to $O(n^2*m)$, where $O(m) \approx O(2^n)$.

4. Conclusions

The paper first introduced the concept of UIO sequences, the minimum–length signature, and PUIO sequences. The paper then defined the Minimum–Cost Pattern Covering Problem (MCPCP), which corresponds to constructing a minimum–length signature of a given state by making use of all of its PUIO sequences. A dynamic–programming–based method was then presented to solve the MCPCP. Three R–rules were derived to reduce the target problem to simpler subproblems and three T–rules were employed to terminate the reduction process. Compared to a brute–force algorithm requiring $O(n*2^m)$ time, the algorithm reduced the time complexity to $O(n^2*m)$.

References

[1] A . V. Aho, A . T. Dahbura, D. Lee, and M. U. Uyar, "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours," Protocol Specification, Testing, and Verification VII, pp.75–86, 1988.

[2] F. C. Hennie, "Fault Detecting Experiments for Sequential Circuits," in Proc. Fifth Ann. Symposium Switching Circuit Theory and Logical Design, pp. 95–110, 1964.

[3] R. E. Miller, and S. Paul, "Generating Minimal Length Test Sequences for Conformance Testing of Communication Protocols," Proc. IEEE INFOCOM '91, pp. 970–979, 1991.

[4] K. K. Sabnani, and A. T. Dahbura, "A Protocol Test Generation Procedure," Computer Networks and ISDN Systems, Vol. 15, No. 4, pp. 285–297, 1988.

[5] D. Sidhu, and T. Leung, "Fault Coverage of Protocol Test Methods," Proc. IEEE INFOCOM'88, pp.80–85, 1988.

[6] Wendy Y. L. Chan, Son T. Vuong, and M. Robert Ito, "An Improved Protocol Generation Procedure Based on UIOS," SIGCOMM'89, pp. 283–294, 1989.

[7] W. H. Chen, and C. Y. Tang, "Computing the Optimal IO Sequence of a Protocol in Polynomial Time," Information Processing Letters, Vol. 8, No. 40, pp. 145 – 148, 1991.

[8] W. Chun, and P. D. Amer, "Improvements on UIO sequence generation and partial UIO sequence," Proc. of Int'l Symposium on Protocol Specification, Testing and Verification, 1992.