ELEKTROTEHNIŠKI VESTNIK 80(4): 1–5, 2013
ORIGINAL SCIENTIFIC PAPER

On the Use of Status Messages in Checking Sequences for the Distributed Test Architecture

Monika Kapus-Kolar

Jožef Stefan Institute, Department of Communication Systems, Jamova 39, SI-1111 Ljubljana, Slovenia
E-mail: monika.kapus-kolar@ijs.si

Abstract. In the testing of reactive systems, a checking sequence (CS) is a sequence of inputs for which a proper response from the system under test means that the system would respond properly also to every other input sequence. In the paper, we suggest how to construct cheap CSs for multi-port systems under the assumption that there is a tester placed at each port, that the testers can communicate exclusively through the system, that there is no global clock and that any tester can at any time ask the system to issue at every port a status report. We consider those CSs in which all the problems of controllability, observability and state recognition specific to the distributed test architecture are solved simply with status requests. For such a CS, our method to some extent minimizes also the number of the employed requests.

Key words: formal methods, testing, distributed test architecture, checking sequence, status message

O uporabi stanjskih sporočil v preverjalnih zaporedjih za porazdeljeno testno arhitekturo

V testiranju odzivnih sistemov je preverjalno zaporedje (PZ) zaporedje vhodnih sporočil, za katerega pravilen odziv testiranega sistema pomeni, da bi se sistem pravilno odzval tudi na vsako drugo zaporedje vhodnih sporočil. V članku svetujemo, kako konstruirati cenena PZ za večvratne sisteme ob predpostavki, da imajo vrata vsaka svojega preizkuševalca, da ti lahko komunirajo samo preko sistema, da ni skupne ure in da vsak preiskuševalec lahko kadarkoli zahteva, da sistem na vseh vratih objavi svoje trenutno stanje. Obravnavamo tista PZ, ki vse za privzeto porazdeljeno testno arhitekturo značilne probleme kontrolabilnosti, observabilnosti in ugotavljanja stanja rešujejo preprosto z zahtevami po objavi stanja. Za taka PZ naša metoda zmanjšuje tudi število teh zahtev.

1 INTRODUCTION

In the testing of *reactive systems*, e.g. hardware components or distributed service providers, a *checking sequence* (CS) is a sequence of inputs for which a proper response from the *system under test* means that the system would respond properly also to every other input sequence. CSs are the most desirable kind of *test suites*, for their application does not require that the system is repeatedly reliably reset into its initial state.

In the paper, we discuss CS construction under the assumption that the system, call it N, and its *specification*, call it M, are *multi-port deterministic finite state machines*, that M is *strongly connected*, that N has at most as many states as M, that there is a tester placed at each port, that the testers can communicate exclusively through N, that there is no global clock and that any tester can at any time ask the system to issue at every port a *status report* (SRP), i.e., an indication of its current state. With the exception of the last one, these are the usual assumptions in the automated construction of test suites for distributed systems and the last one is also increasingly popular [1], [2].

The assumed distributed test architecture brings specific problems of *controllability*, *observability* and *state recognition*. If, however, one decides to solve every such problem simply with *status requests* (SRQs), CS construction becomes very easy [1]. In the following, we, hence, focus on this particular class of CSs, showing how for such a CS, one can efficiently minimize, at least to some extent, the cost (e.g. the length) and, optionally, also and primarily the number of the employed SRQs.

2 THE SYSTEM AND ITS SPECIFICATION

We assume that M and N both operate on the same set of ports and can in every state execute every input defined on the ports. Upon executing an input x in a state s, they execute a set y of outputs, at most one per port, and enter a state s', thereby executing a *transition* $(s, s', x/y)$. Their choice of y and s' depends only on x and s. If x is an SRQ, $s' = s$ and y for every port comprises an SRP. If x is an *ordinary input*, every member of y is an *ordinary output*, i.e., not an SRP. Moreover, if x is not an SRQ, it is possible that the choice of y and s' which N makes differs from the specified, i.e., from the choice which M makes.

Received ??
Accepted ??

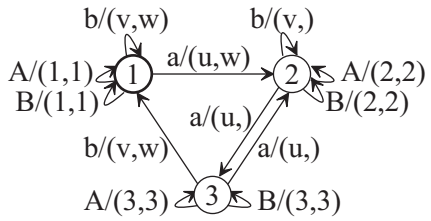

Figure 1. Example two-port M.

It is assumed that the only other way in which N can be an incorrect implementation of M is that its initial state is not the specified one. A CS is, hence, any input sequence checking the initial state of N and the implementation of every specified *ordinary transition*. It is assumed that via the latter, M can from every state reach every other state. For M, let $init(M)$ denote its initial state, $otr(M)$ the set of its ordinary transitions and $str(M)$ the set of its *status transitions*.

For a given input x, let $\zeta(x)$ denote the *cost of application to* N, presumably a real number greater than 0. Besides, let $\zeta'(x) = 1$ if x is an SRQ and maximum avoidance of SRQs is desirable and $\zeta'(x) = 0$ otherwise, with $cost(x)$ denoting the pair $(\zeta'(x), \zeta(x))$ in $\mathbb{R}^2$. A (q_1, q_2) in $\mathbb{R}^2$ is considered less than or equal to a (q'_1, q'_2) in $\mathbb{R}^2$ if $(q_1 < q'_1) \vee ((q_1 = q'_1) \wedge (q_2 \leq q'_2))$. The sum of two members (q_1, q_2) and (q'_1, q'_2) of $\mathbb{R}^2$ is $(q_1 + q'_1, q_2 + q'_2)$. For every (q_1, q_2) in $\mathbb{R}^2$, $-(q_1, q_2)$ denotes $(-q_1, -q_2)$.

Example 1. Suppose that there is a port α accepting an ordinary input a and an SRQ A and a port β accepting an ordinary input b and an SRQ B, with $\zeta(a) = \zeta(A) = \zeta(b) = \zeta(B) = 1$ and $\zeta'(A) = \zeta'(B) = 1$. M can then be the one depicted in Fig. 1, with the state set $\{1, 2, 3\}$, 1 the initial state and the output set in every transition label encoded as a (y_α, y_β) with y_α the output at α, if any, and y_β the output at β, if any. For every state s and input x, let $t_{s,x}$ denote the corresponding transition.

For a given *transition sequence* (TS) $\tau = (s_1, s_2, x_1/y_1) \dots (s_k, s_{k+1}, x_k/y_k)$, let $init(\tau)$ denote its initial state s_1, $fin(\tau)$ its final state s_{k+1}, $is(\tau)$ its input sequence $x_1 \dots x_k$, $ios(\tau)$ its input/output sequence (IOS) $x_1/y_1 \dots x_k/y_k$, $cost(\tau)$ its cost $\Sigma_{1 \leq i \leq k} cost(x_i)$, $port(\tau)$ the port of x_1 and $ports(\tau)$ the set of all the ports involved in $ios(\tau)$. Let ϵ denote a zero-length TS. Let $sts(M)$ denote the set of all those TSs $\tau = t_1 \dots t_k$ of M which are *synchronizable*, i.e., for every $1 \leq i < k$ satisfy $port(t_{i+1}) \in ports(t_i)$.

3 Candidate Checking Sequences

Like [1], we define that a *checking TS* (CTS), i.e., a TS whose input sequence is a CS, is a TS of M satisfying the following:

1) It is synchronizable.
2) There exists such a permutation $t_1 \dots t_{|otr(M)|}$ of the transitions in $otr(M)$ that for every $1 \leq i \leq |otr(M)|$, there is a segment $t'_i \tau_i t_i t''_i$ with t'_i and t''_i two status transitions and τ_i a TS whose every member belongs to the set $\{t_1, \dots, t_{i-1}\}$.
3) It starts in $init(M)$ and the input of its first transition is an SRQ.

To see that for such a TS τ, $is(\tau)$ is indeed a CS appropriate for the assumed distributed test architecture, observe the following:

1) The first constraint solves the problem of controllability, i.e., secures that during the application of $is(\tau)$ to N, every tester always knows the right time for enabling (so that it can be executed when N completes its current transition) the next input at its port.
2) The second constraint by induction secures implementation checking for the consecutive transitions in $t_1 \dots t_{|otr(M)|}$, as for every $1 \leq i \leq |otr(M)|$, proper implementation of the transitions $t_1 \dots t_{i-1}$ implies that whenever $ios(t'_i \tau_i)$ (actually its projections onto individual ports) is observed on N, the resulting state of N is exactly the initial state of t_i, so that if $ios(t_i t''_i)$ is subsequently observed, one knows that t_i is also properly implemented.
3) The third constraint secures the checking of the initial state of N.

4 Representing Candidate Checking Sequences in a Digraph

The first step of our CS-construction procedure is to construct a *digraph* G in which promising CTSs are represented as specific *walks*. A digraph is a set of vertices of which some are connected by directed edges. Whenever introducing an edge, we will specify it as a (v, v', l, c) with v its initial vertex, v' its final vertex, l its label and c its cost. For our purposes, we define that every edge label is a (possibly empty) sequence of transitions of M and minuses, whereas every edge cost is a member of $\mathbb{R}^2$. The cost of a digraph is the sum of its edge costs.

For a given walk $w = (v_1, v_2, l_1, c_1) \dots (v_k, v_{k+1}, l_k, c_k)$, let $init(w)$ denote its initial vertex v_1, $fin(w)$ its final vertex v_{k+1}, $lab(w)$ its label $l_1 \dots l_k$ and $cost(w)$ its cost $\Sigma_{1 \leq i \leq k} c_i$. If $v_1 = v_{k+1}$, w is a *tour*. If $(k > 0) \wedge (\wedge_{1 \leq i < j \leq k+1} ((v_i = v_j) = ((i, j) = (1, k+1))))$, w is a *cycle*. An edge set is *acyclic* if no subset of it forms a cycle. An *Euler tour* of a digraph is a tour traversing each of its edges exactly ones.

The construction of G starts by selecting, for every transition t in $otr(M)$, a minimum-cost transition t' in $str(M)$ with $tt' \in sts(M)$. Call it $next(t)$ and let $str'(M)$ denote the set $\{next(t) | t \in otr(M)\}$. One

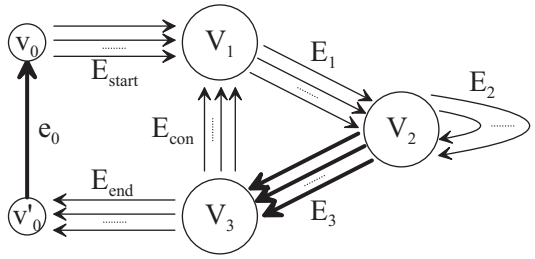

Figure 2. Schematic representation of the digraph G.

also selects, for every state s of M and transition t in $str'(M)$, a minimum-cost TS τ in $sts(M)$ with $(init(\tau) = s) \wedge (\tau t \in sts(M))$, call it $\tau_{s,t}$.

We now describe the vertices and edges of G. As we see from its schematic representation in Fig. 2, its vertices are those in the sets V_1, V_2 and V_3 plus the vertices v_0 and v'_0, whereas its edges are those in the sets E_1, E_2, E_3, E_{con}, E_{start} and E_{end} plus the edge e_0. The vertices in V_1 are the initial vertices of the edges in E_1. The vertices in V_2 are the initial vertices of the edges in $E_2 \uplus E_3$. The vertices in V_3 are the final vertices of the edges in E_3.

$$E_1 = \{(v^1_{init(t),port(t)}, v^2_{init(t'),port(t')}, t, cost(t)) \\ |(t \in str'(M)) \wedge (t' \in otr(M)) \wedge (tt' \in sts(M))\}$$
$$E_2 = \{(v^2_{init(t),port(t)}, v^2_{init(t'),port(t')}, t, cost(t)) \\ |(t \in otr(M)) \wedge (t' \in otr(M)) \wedge (tt' \in sts(M))\}$$
$$E_3 = \{(v^2_{init(t),port(t)}, v^3_{fin(t'),port(t')}, tt', cost(tt')) \\ |(t \in otr(M)) \wedge (t' = next(t))\}$$
$$E_{con} = \{(v^3_{init(t),port(t)}, v^1_{init(t),port(t)}, -t, -cost(t)) \\ |t \in str'(M)\} \uplus \\ \{(v^3_{s,p}, v^1_{init(t),port(t)}, \tau_{s,t}, cost(\tau_{s,t})) \\ |(t \in str'(M)) \wedge (v^3_{s,p} \in V_3) \wedge \\ ((s,p) \neq (init(t), port(t)))\}$$
$$E_{start} = \{(v_0, v^1_{init(M),p}, \epsilon, (0,0)) | v^1_{init(M),p} \in V_1\}$$
$$E_{end} = \{(v^3_{s,p}, v'_0, \epsilon, (0,0)) | v^3_{s,p} \in V_3\}$$
$$e_0 = (v'_0, v_0, \epsilon, (0,0))$$

Example 2. For the M of Example 1, we choose

$next(t_{1,a}) = next(t_{2,b}) = next(t_{3,a}) = t_{2,A}$
$next(t_{2,a}) = t_{3,A}$
$next(t_{1,b}) = next(t_{3,b}) = t_{1,A}$
$\tau_{1,t_{2,A}} = t_{1,a}$
$\tau_{1,t_{3,A}} = t_{1,a}t_{2,a}$
$\tau_{2,t_{1,A}} = t_{2,a}t_{3,A}t_{3,b}$
$\tau_{2,t_{3,A}} = t_{2,a}$
$\tau_{3,t_{1,A}} = t_{3,b}$
$\tau_{3,t_{2,A}} = t_{3,a}$

The transition $t_{3,A}$ in $\tau_{2,t_{1,A}}$ secures its synchronizability. The resulting G is given in Fig. 3.

For a given walk w in G whose first edge is not of a negative cost, let $ts(w)$ denote the TS obtained from $lab(w)$ by deleting every segment $t{-}t$ with t a transition of M. We define that a walk w of G represents a CTS, namely $ts(w)$, if it is an $ew'e'e_0$ with e an edge in E_{start}, w' a walk whose edge set is an acyclic subset of E_2 and e' an edge in E_{end}. In the following, such a walk is called a *checking walk* (CW).

To see that for every CW w, $ts(w)$ is indeed a CTS, observe the following:

1) $ts(w)$ is synchronizable and starts with a status transition starting in $init(M)$.
2) As those edges in w that are in E_2 form an acyclic subset of it, there exists such a permutation $t_1 \ldots t_{|otr(M)|}$ of the transitions in $otr(M)$ that for every $1 \leq i \leq |otr(M)|$, w has a segment $e_i w_i e'_i$ with e_i and edge in E_1 with $lab(e_i)$ a status transition, e'_i an edge in E_3 with $lab(e'_i)$ a $t_i t'_i$ with t'_i a status transition and w_i such a walk consisting of members of E_2 that every transition in $ts(w_i)$ belongs to the set $\{t_1, \ldots, t_{i-1}\}$.

Example 3. In the G of Example 2, one of the CWs, a w, is

$(v_0, v^1_{1,\alpha}, \epsilon, (0,0))w_{1,a}$
$(v^3_{2,\alpha}, v^1_{2,\alpha}, -t_{2,A}, (-1,-1))w_{2,a}$
$(v^3_{3,\alpha}, v^1_{3,\alpha}, -t_{3,A}, (-1,-1))w_{3,b}$
$(v^3_{1,\alpha}, v^1_{1,\alpha}, -t_{1,A}, (-1,-1))w_{1,b}$
$(v^3_{1,\alpha}, v^1_{1,\alpha}, -t_{1,A}, (-1,-1))w_{2,b}$
$(v^3_{2,\alpha}, v^1_{2,\alpha}, -t_{2,A}, (-1,-1))w_{3,a}$
$(v^3_{2,\alpha}, v'_0, \epsilon, (0,0))(v'_0, v_0, \epsilon, (0,0))$

with

$w_{1,a} = (v^1_{1,\alpha}, v^2_{1,\alpha}, t_{1,A}, (1,1))(v^2_{1,\alpha}, v^3_{2,\alpha}, t_{1,a}t_{2,A}, (1,2))$
$w_{2,a} = (v^1_{2,\alpha}, v^2_{2,\alpha}, t_{2,A}, (1,1))(v^2_{2,\alpha}, v^3_{3,\alpha}, t_{2,a}t_{3,A}, (1,2))$
$w_{3,b} = (v^1_{3,\alpha}, v^2_{3,\beta}, t_{3,A}, (1,1))(v^2_{3,\beta}, v^3_{1,\alpha}, t_{3,b}t_{1,A}, (1,2))$
$w_{1,b} = (v^1_{1,\alpha}, v^2_{1,\beta}, t_{1,A}, (1,1))(v^2_{1,\beta}, v^3_{1,\alpha}, t_{1,b}t_{1,A}, (1,2))$
$w_{2,b} = (v^1_{1,\alpha}, v^2_{1,\alpha}, t_{1,A}, (1,1))(v^2_{1,\alpha}, v^2_{2,\beta}, t_{1,a}, (0,1))$
$\quad (v^2_{2,\beta}, v^3_{2,\alpha}, t_{2,b}t_{2,A}, (1,2))$
$w_{3,a} = (v^1_{2,\alpha}, v^2_{2,\alpha}, t_{2,A}, (1,1))(v^2_{2,\alpha}, v^2_{3,\alpha}, t_{2,a}, (0,1))$
$\quad (v^2_{3,\alpha}, v^3_{2,\alpha}, t_{3,a}t_{2,A}, (1,2))$
$ts(w_{1,a}) = t_{1,A}t_{1,a}t_{2,A}$
$ts(w_{2,a}) = t_{2,A}t_{2,a}t_{3,A}$
$ts(w_{3,b}) = t_{3,A}t_{3,b}t_{1,A}$
$ts(w_{1,b}) = t_{1,A}t_{1,b}t_{1,A}$
$ts(w_{2,b}) = t_{1,A}t_{1,a}t_{2,b}t_{2,A}$
$ts(w_{3,a}) = t_{2,A}t_{2,a}t_{3,a}t_{2,A}$
$lab(w) = ts(w_{1,a}){-}t_{2,A}ts(w_{2,a}){-}t_{3,A}ts(w_{3,b}){-}t_{1,A}$
$\quad ts(w_{1,b}) - t_{1,A}ts(w_{2,b}) - t_{2,A}ts(w_{3,a})$
$\quad = t_{1,A}t_{1,a}t_{2,A} - t_{2,A}t_{2,A}t_{2,a}t_{3,A} - t_{3,A}$
$\quad t_{3,A}t_{3,b}t_{1,A} - t_{1,A}t_{1,A}t_{1,b}t_{1,A} - t_{1,A}$
$\quad t_{1,A}t_{1,a}t_{2,b}t_{2,A} - t_{2,A}t_{2,A}t_{2,a}t_{3,a}t_{2,A}$
$ts(w) = t_{1,A}t_{1,a}t_{2,A}t_{2,a}t_{3,A}t_{3,b}t_{1,A}t_{1,b}t_{1,A}t_{1,a}t_{2,b}$
$\quad t_{2,A}t_{2,a}t_{3,a}t_{2,A}$

For each individual transition $t_{s,x}$ in $otr(M)$, $w_{s,x}$ is that segment of w that secures the checking of its implementation. Thanks to the negative-cost edges in w, the corresponding segments $ts(w_{s,x})$ of $ts(w)$ partially overlap. To see that $ts(w)$ is a CTS, observe that it is synchronizable, that it starts with $t_{1,A}$ and that

$ts(w_{1,a}) = t_{1,A}\tau_{1,a}t_{1,a}t_{2,A}$ with $\tau_{1,a} = \epsilon$
$ts(w_{2,a}) = t_{2,A}\tau_{2,a}t_{2,a}t_{3,A}$ with $\tau_{2,a} = \epsilon$

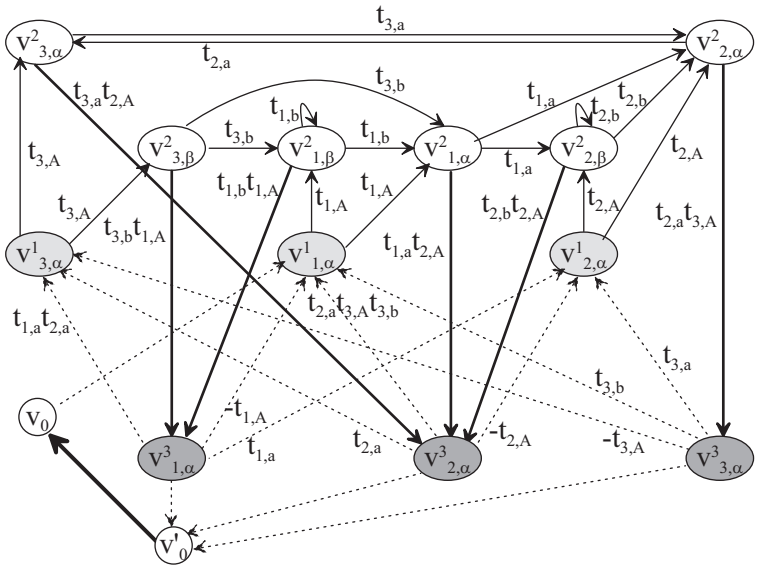

Figure 3. G for the example M, with the edges in $E_3 \uplus \{e_0\}$ bold and the edges in $E_{con} \uplus E_{start} \uplus E_{end}$ dashed.

$ts(w_{3,b}) = t_{3,A}\tau_{3,b}t_{3,b}t_{1,A}$ with $\tau_{3,b} = \epsilon$
$ts(w_{1,b}) = t_{1,A}\tau_{1,b}t_{1,b}t_{1,A}$ with $\tau_{1,b} = \epsilon$
$ts(w_{2,b}) = t_{1,A}\tau_{2,b}t_{2,b}t_{2,A}$ with $\tau_{2,b} = t_{1,a}$
$ts(w_{3,a}) = t_{2,A}\tau_{3,a}t_{3,a}t_{2,A}$ with $\tau_{3,a} = t_{2,a}$

5 Finding a Cheap Candidate Checking Sequence

A digraph is *edge-induced* if its vertices are exactly the starts and the ends of its edges. A digraph is *symmetric* if for every vertex, the number of the incoming edges is the same as the number of the outgoing edges. A digraph is *strongly connected* if from every vertex, there is a walk to every other vertex. A *component* of a digraph is a strongly connected sub-digraph which is not a part of any larger strongly connected sub-digraph.

Following an idea of [3], for finding a cheap (though not necessarily minimum-cost) CW we suggest the following procedure:

1) Construct one of the cheapest symmetric edge-induced digraphs (SEIDs) consisting of e_0, the edges in E_3 and any number of instances of the remaining edges of G. This is the most demanding step of the procedure, but can be accomplished in polynomial time [4].
2) Enhance the digraph into a strongly connected SEID G', using the following procedure for merging two components G_1 and G_2 of its current version:
   a) In G, select such a minimum-cost cycle $e_1w_1e'_1e''_1e_2w_2e'_2e''_2$ with e_1 and e_2 two edges in E_1, w_1 and w_2 two sub-walks whose every edge is in E_2, e'_1 and e'_2 two edges in E_3 and e''_1 and e''_2 two edges in E_{con} that every edge in $e_1w_1e'_1$ has an instance in G_1 and every edge in $e_2w_2e'_2$ has an instance in G_2.
   b) For every vertex in the cycle, add to G' a new instance.
3) Construct the CW as any Euler tour of G'. As G' is symmetric, simply initialize the current CW approximation to a zero-length walk starting and ending in v_0 and then repeatedly append to it one of the yet uncovered outgoing edges of its final vertex.

After the first step of the procedure, G' has the following properties:

1) It is symmetric, every edge is an instance of an edge of G and for every edge in E_3, there is at least one instance.
2) Exactly one edge is an instance of e_0. Hence, exactly one edge is an instance of an edge in E_{start} and exactly one edge is an instance of an edge in E_{end}, for otherwise the digraph would not be symmetric.
3) There is no cycle consisting exclusively of instances of edges in E_2, because for every such cycle, removal of its edges from G' would contradictorily result in an even cheaper SEID consisting of e_0, the edges in E_3 and any number of instances of the remaining edges of G.

In any subsequent merging of two components, all the properties of G' are preserved, because no edge is removed, every new edge is an instance of an edge in $E_1 \uplus E_2 \uplus E_3 \uplus E_{con}$, the new edges form a cycle and the set of those edges in E_2 of which G' comprises an instance is preserved. Hence, the final version of G' also possesses the properties, which indeed makes the subsequently constructed tour w a CW. As an additional

improvement of the corresponding CTS $ts(w)$, replace its first element with one of the cheapest transitions t in $str(M)$ with $init(t) = init(M)$.

6 DISCUSSION

In the usual approach to the construction of a CS without controllability problems [5], the first step is to secure the checking of a sufficient set of *locally distinguishing sequences* (LDSs) and a sufficient set of *transfer IOSs* (TIOSs). Without this step, the LDSs and the TIOSs, employed in the subsequently constructed *transition implementation tests* (TITs), cannot be trusted. In the considered special setting, a sufficient set of reliable LDSs and TIOSs is available by definition, as every SRQ is a (length one) reliable LDS and the label of every transition in $sts(M)$ is a (length one) reliable TIOSs.

As another complication in the usual approach, the conceived TITs sometimes fail to completely check the implementation of the transition labels, so that additional tests are necessary [6]. In the considered special setting, we have not encountered the problem, in spite of relying on a straightforward generalization of the usual approach for the single-port case [7].

For the generalization, we had to find a way to adequately represent in the auxiliary digraph G only the synchronizable TSs of M. Interestingly, we discovered that the topology of the central (i.e. E_2) part of G cannot be simply that of the relevant part of the canonical representation $\chi_{min}(M)$ [8] of $sts(M)$, for then the *dependency relation* between the TITs employed in the constructed TS can have *cycles* even if the employed part of E_2 is acyclic. We prevented this by also securing $init(e) = init(e')$ for every two edges e and e' in $E_2 \uplus E_3$ whose labels start with the same transition in $otr(M)$. The hint seems useful also for the multi-port case without the status-reporting capability, for which a method with thorough global optimization of the CS has yet to be developed.

REFERENCES

[1] R.M. Hierons, "Using status messages in the distributed test architecture," *Inf. Soft. Tech.*, vol. 51, no. 7, pp. 1123-1130, 2009.
[2] H. Dan, R.M. Hierons, "Controllability problems in MSC-based testing," *Computer Journal*, vol. 55, no. 11, pp. 1270-1287, 2012.
[3] R.M. Hierons, H. Ural, "Optimizing the length of checking sequences," *IEEE Trans. Comput.*, vol. 55, no. 5, pp. 618-629, 2006.
[4] A.V. Aho, A.T. Dahbura, D. Lee, M.U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours," *IEEE Trans. Comm.*, vol. 39, no. 11, pp. 1604-1615, 1991.
[5] R.M. Hierons, H. Ural, "Checking sequences for distributed test architectures," *Dist. Comput.*, vol. 21, no. 3, pp. 223-238, 2008.
[6] J. Chen, R.M. Hierons, H. Ural, "Overcoming observability problems in distributed test architectures," *Inf. Proc. Let.*, vol. 98, no. 5, pp. 177-182, 2006.
[7] H. Ural, X. Wu, F. Zhang, "On minimizing the lengths of checking sequences," *IEEE Trans. Comput.*, vol. 46, no. 1, pp. 93-99, 1997.
[8] R.M. Hierons, "Canonical finite state machines for distributed systems," *Theor. Comput. Sci.*, vol. 411, no. 2, pp. 566-580, 2010.

Monika Kapus-Kolar received her B.Sc. degree in electrical engineering from the University of Maribor, Slovenia, and her M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia. Since 1981 she has been with the Jožef Stefan Institute, Ljubljana, where she is currently a researcher at the Department of Communication Systems. Her current research interests include formal specification techniques and methods for the development of real-time, concurrent and reactive systems.