doi:10.1093/comjnl/bxh000

Three Generalizations to a Generic Integrated Test Generation Method for Finite State Machines

Monika Kapus-Kolar

Jožef Stefan Institute, Ljubljana, Slovenia
Email: monika.kapus-kolar@ijs.si

In a previous paper, we proposed a generic test generation method for deterministic implementations of deterministic finite state machines. The method supports a wide class of testing strategies, multi-criteria optimization and integrated handling of all the usual optimization concerns. The present paper generalizes the method to an even wider class of strategies, to nondeterministic machines and to machines handling data.

Keywords: test sequence generation, nondeterministic finite state machine, extended finite state machine, generic method

Received August 21, 2007; revised July 28, 2008

1. INTRODUCTION

In the paper, we discuss *generation of tests for systems which can be abstractly described as a nondeterministic extended finite state machine* (nondeterministic extended FSM, nondeterministic EFSM, NEFSM), generalizing our study from [1] on test generation from a deterministic FSM (DFSM). Abstractly speaking, the problem of generating a test suite from a given system model is to find in the model a cost-optimal set of system runs collectively satisfying some specific properties. As precise property satisfaction and test cost optimization are conflicting concerns, test generation methods differ in how much emphasis they give to each of them.

As cost optimization is nearly always of primary importance and as individual system runs can be interpreted as walks in a graph, the more traditional methods tend to reduce test generation to various generic walk or walk-set optimization problems on graphs. Unfortunately, the methods each pursue only a very limited kind of test properties, the reason being that each kind of property requires a specific graph model of the test generation problem.

With the increasing importance of software quality, one needs methods more flexible than that. This is why test generation methods are increasingly moving towards model-checking methods (see, for example, [2]). The latter look for adequate system runs directly on the system model, but seldom excel in test cost optimization. The model-checking approach simplifies test generation particularly for data-handling systems, whose runs are typically so numerous that the corresponding walks in a graph model have to be represented in sheaves.

The method proposed in [1] and generalized in the present paper *synergistically combines the traditional and the model-checking approach to test generation*. Accepting *now an even wider class of declaratively specified testing strategies* and supporting *multi-criteria optimization*, it resembles the model-checking approach, but nevertheless *reduces the test generation problem to a generic optimization problem on a graph*, so that among the system runs identified as promising, an optimal set of runs can be computed efficiently, with domain-unspecific (and, hence, more widely available) tools exploiting special graph-theoretic knowledge. We also continue providing *guidelines for conducting the method in an approximate manner*, to cope with the enormous inherent complexity of the problem.

The described features give the method significant *practical value*, although one must be aware that the method is *generic*: It cannot be applied unless its parameters are instantiated. Above all, one has to specify, in the prescribed formal style, the desired testing strategy. For some of the best-known strategies, we in [1] actually provided such formalization, thereby turning the method into an immediately applicable test generation algorithm which, unlike earlier algorithms, *implements the adopted strategy optimally*, i.e., generates a test provably satisfying the

strategy to the maximum possible extent and at a minimum cost.

The method is important also *methodologically*: As it approaches the test generation problem in an entirely abstract way and handles all the usual optimization concerns in a precise and truly integrated manner, it represents a synthesis of most of the earlier methods for reducing test generation to generic graph-theoretic problems and *a standard against which more specific or approximate methods*, both old and new, *can be evaluated and compared*. Besides, we generalize the method to NEFSMs in a manner which is not strategy-specific. Thereby, we provide *a generic link between test generation methods for DFSMs and those for NEFSMs* and facilitate *dynamic selection of the strategy during adaptive testing*.

2. HOW TO READ THE PAPER

Our method is more of the model-checking kind and, hence, not defined in terms of traditional concepts. Consequently, for someone specializing in traditional test generation methods, either as their user or as their developer, reading the paper will not be easy, particularly because, to express our ideas both precisely and concisely, we were forced to use a lot of formal notation. Still, we encourage such readers to persists, because the paper is, like [1], intended primarily for them. By understanding it, they will gain a much wider perspective on the test construction process, particularly on all those exotic graphs which traditional test generation methods tend to construct as the problem model.

Although we in Section 3 summarize all the relevant notations and definitions from [1] and in Section 5 the method of [1] itself, an uninitiated reader is advised to first thoroughly study [1], in which we extensively explain the expected benefits of the proposed paradigm shift and on the example of many well-known testing strategies illustrate how the method relates to the traditional approach. Once the ideas are understood, reading the present paper should not be a problem, as it just extends the method to an increasingly general setting.

A reader familiar with the model-checking approach will also want to start with [1], not for the lack of insight, but to see all the details and proofs of the method. In the present paper, we explain the method just to the detail necessary to understand the proposed generalizations. We also provide no correctness proofs, as the generalizations are semantically just clever applications of the original method.

In the paper, we propose for the method a series of incremental generalizations, first in Section 6 to a wider class of testing strategies, then in Section 7 additionally to systems modelled as a nondeterministic FSM (NFSM) and finally, in Section 8, to systems modelled as an EFSM, which may as well be an NEFSM. As the generalizations form a cascade, it is important that for any abstract object introduced in Section 5, 6 or 7, the reader not only understands its meaning, but also remembers the formal name which it is given, because the name is used also in the subsequent sections.

Trying to integrate a multitude of concepts, we in the paper deal with abstract objects of many different kinds. Their definitions are for easy reference collected in Section 3, but the reader should better get acquainted with them when invited to do so in Section 4, where we informally introduce all the major concepts and procedures employed in the more formal Sections 5 to 8. The latter also provide a running example of test construction. Where a graph related to the example is, for illustration of a concept, referred to in an earlier section, one should just make the recommended observation and wait for further details till the formal sections.

3. NOTATIONS AND DEFINITIONS

3.1. Sequences

A sequence (or a vector) $\bar{o}$ of some objects o_i is an $o_1, \ldots, o_k$. In an extreme case, its size k is 0 or infinite. An empty sequence will be denoted by ε. Where we put an object o in a position where one by definition expects a sequence of objects of the class to which o belongs, it should be interpreted as a sequence with o the only component.

For a sequence $\bar{o}$, $cmps(\bar{o})$ denotes the set of its components, $cmp(\bar{o}, i)$ denotes its i-th component, $pfx(\bar{o}, i)$ its prefix of length i, and $sfx(\bar{o}, i)$ its suffix of length i. *Concatenation* $\bar{o} \cdot \bar{o}'$ of an $\bar{o} = o_1, \ldots, o_k$ and an $\bar{o}' = o'_1, \ldots, o'_{k'}$ is sequence $o_1, \ldots, o_k, o'_1, \ldots, o'_{k'}$.

3.2. Directed Graphs

A *directed graph (digraph)* G is defined by a tuple (V, E) in which V is a non-empty set of *vertices* and E is a set of directed *edges* between the vertices. If a vertex v_0 in V is designated as the root, G is *rooted* and more precisely defined by (V, v_0, E).

An edge e defined by a tuple $(v, v', l, \bar{c})$ goes from vertex v to vertex v'. l is a (possibly empty) sequence of some symbols and represents the *label* of e. An edge with an empty label will be called *unlabelled*. $\bar{c}$ is the *cost vector* representing the cost of traversing e, with different components corresponding to *different cost criteria*, where the size of the vector is the same for every edge in G. We assume that every component of a cost vector is a non-negative real. A $\bar{c}$ of the expected size with all components 0 will be denoted by $\bar{0}$.

For *comparing cost vectors*, we say that $\bar{c} < \bar{c}'$ if there is an i such that for every $1 \leq j < i$, $cmp(\bar{c}, j) = cmp(\bar{c}', j)$, and $cmp(\bar{c}, i) < cmp(\bar{c}', i)$, i.e., the costs listed earlier are considered absolutely more important than those listed later.

A sequence $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, l_i, \bar{c}_i)$ for $1 \leq i \leq k$, of consecutive edges of G is called a *walk* and has a *vertex sequence* $vrt(\bar{e}) = v_1, \ldots, v_{k+1}$, an *initial vertex* $init(\bar{e}) = v_1$, a *final vertex* $fin(\bar{e}) = v_{k+1}$, a *label* $lab(\bar{e}) = l_1 \cdot \ldots \cdot l_k$ and a *cost vector* $cost(\bar{e}) = \bar{c}_1 + \ldots + \bar{c}_k$. For the extreme case of $\bar{e} = \varepsilon$, where $init(\bar{e})$ and $fin(\bar{e})$ cannot be assessed as above, we assume that the vertex $init(\bar{e}) = fin(\bar{e})$ is evident from the context. If $init(\bar{e}) = fin(\bar{e})$, $\bar{e}$ is *closed*. A closed walk is a *tour*. If $init(\bar{e}) = v_0$, $\bar{e}$ is *initially rooted*. If $fin(\bar{e}) = v_0$, $\bar{e}$ is *finally rooted*. A tour is *rooted* if it is initially rooted. If $\bar{e} = \varepsilon$ or if v_1 to v_k are distinct, $\bar{e}$ is a *path*. A closed path with $k = 1$ is a *loop*.

If there is a path from v_0 to every v in V, G is *initially connected*. If there is a path from every v in V to every v' in V, G is *strongly connected*.

A digraph (V', E') is a *subgraph* of a digraph (V, E) if $V' \subseteq V$ and $E' \subseteq E$.

3.3. Finite State Machines

An FSM M is defined by a tuple (S, s_0, X, Y, T) in which S is a finite set of *states*, s_0 is the *root state* in S, X is a finite *input alphabet*, Y is a finite *output alphabet* and T is a finite set of *transitions* $(s, s', x/y, \bar{c})$ with s and s' in S, x in X, y in Y and $\bar{c}$ a cost vector, so that M can also be understood as a rooted digraph (S, s_0, T). If a $\tau = (s, s', z, \bar{c})$ in T is not the only member of T of the form $(\_, s', z, \_)$, we call it a *converging transition*. For example, the FSM in Figure 1 (Section 6), copied from [3], has converging transitions τ_1, τ_2, τ_6 and τ_7.

If T comprises an $(s, \_, x/\_, \_)$ for every s in S and x in X, M is *completely specified*. If there exist such functions δ, λ and γ that every τ in T is an $(s, \delta(s, x), x/\lambda(s, x), \gamma(s, x))$, M is *deterministic*. If T comprises no pair of transitions $(s, s', z, \bar{c})$ and $(s, s'', z, \bar{c}')$ with $s' \neq s''$, M is *observable*. For example, the FSM in Figure 1 is completely specified, but nondeterministic and unobservable.

A walk $\bar{\tau} = \tau_1, \ldots, \tau_k$, $\tau_i = (s_i, s_{i+1}, x_i/y_i, \bar{c}_i)$ for $1 \leq i \leq k$, in an FSM has an *initial state* $init(\bar{\tau}) = s_1$, a *final state* $fin(\bar{\tau}) = s_{k+1}$, a *label* $lab(\bar{\tau}) = z_1, \ldots, z_k = x_1/y_1, \ldots, x_k/y_k = \bar{x}/\bar{y}$ with *input part* $\bar{x} = in(\bar{\tau}) = in(lab(\bar{\tau})) = x_1, \ldots, x_k$ and *output part* $\bar{y} = out(\bar{\tau}) = out(lab(\bar{\tau})) = y_1, \ldots, y_k$, and a *cost vector* $cost(\bar{\tau}) = \bar{c}_1 + \ldots + \bar{c}_k$. For the extreme case of $\bar{\tau} = \varepsilon$, we assume that the state $init(\bar{\tau}) = fin(\bar{\tau})$ is evident from the context.

An FSM is *able to refuse an input sequence* $\bar{x}$ if for a prefix $\bar{x}' \cdot x$ of $\bar{x}$, it has an initially rooted walk $\bar{\tau}$ with $in(\bar{\tau}) = \bar{x}'$ for which there is no transition of the form $(fin(\bar{\tau}), \_, x/\_, \_)$.

A rooted tour $\bar{\tau}$ from an FSM M will be called a *deterministic rooted tour* (DRT), i.e., be a member of $Trs(M)$, if M is not able to refuse $in(\bar{\tau})$ and every initially rooted walk $\bar{\tau}'$ in M with $in(\bar{\tau}') = in(\bar{\tau})$ is a tour. For example, in the M in Figure 1, the tour "τ_4, τ_8, τ_2" is a DRT, whereas "τ_3, τ_2" is not.

In an FSM, the *forward exclusion set* $FE(\bar{z})$ of an input/output (I/O) sequence $\bar{z}$ is the set of all states s for which there is no walk $\bar{\tau}$ with $init(\bar{\tau}) = s$ and $lab(\bar{\tau}) = \bar{z}$. The *backward exclusion set* $BE(\bar{z})$ of a $\bar{z}$ is the set of all states s for which there is no walk $\bar{\tau}$ with $fin(\bar{\tau}) = s$ and $lab(\bar{\tau}) = \bar{z}$. For example, in the FSM in Figure 1, the sequence "$b/f, a/d$" has FE $\{s_2\}$ and BE $\{s_0\}$.

In an FSM, a sequence of consecutive transitions $\bar{\tau}$ is called a forward or a backward *state-recognition sequence* (SRS) when it is executed with an aim to distinguish the initial or the final state of the sequence, respectively, from the other states in the FSM. Its power is characterized by $FE(lab(\bar{\tau}))$ or $BE(lab(\bar{\tau}))$, respectively.

If for a set W of walks in a digraph, $in(lab(\bar{e}))$ is defined and the same for every $\bar{e}$ in W, it will be called $in(W)$.

3.4. Representing Digraph Walks in Sheaves

If a set of walks of interest in a digraph is very large, one might prefer to work with a more implicit representation of the set, a digraph in which there are walks interpreted as subsets of the set, in a manner covering all the members of the set and grouping them as convenient. Such concise representation is popular particularly for the set of all the initially rooted walks of a rooted digraph, because in the case that the digraph is initially connected, the set is its precise characterization.

For a walk $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, W_i, \bar{c}_i)$ with W_i a set of walks of a size $n_i > 0$ for $1 \leq i \leq k$, let $SH(\bar{e})$ denote the set of all walks $\bar{e}_1 \cdot \ldots \cdot \bar{e}_k$ with $\bar{e}_i$ in W_i for $1 \leq i \leq k$ and $fin(\bar{e}_i) = init(\bar{e}_{i+1})$ for $1 \leq i < k$, i.e., interpret $\bar{e}$ as a *sheaf* of walks of length $\sum_{1 \leq i \leq k} n_i$. For example, the walk "e'_{24}, e'_{21}" in the digraph in Figure 9 (Section 7.6) represents the walks "$e_{5,0}, e_{6,1}, e_4$" and "$e_{5,0}, e_{9,1}, e_8$" from the digraph in Figure 8 (Section 7.6).

We say that a digraph $G = (V, E)$ with an $Init \subseteq V$ designated as the set of its *entry points* and a $Fin \subseteq V$ as the set of its *exit points* qualifies as a *walk-sheaves graph* (WSG) *for a set W of finite walks* if $SH(\bar{e})$ is defined for every walk $\bar{e}$ in G and the union of all $SH(\bar{e})$ with $\bar{e}$ a walk in G with $init(\bar{e}) \in Init$ and $fin(\bar{e}) \in Fin$ is exactly W.

As an example, take the set of those walks in Figure 3 (Section 7.4) which consist exclusively of labelled edges, start in a vertex with an incoming unlabelled edge, terminate in a vertex with an outgoing unlabelled edge and represent, as the labels of their edges indicate, a DRT from the FSM in Figure 1. In the digraph in Figure 6 (Section 7.5), the labelled edges form a subgraph which is a WSG for the walk set, sheaving the DRTs according to the corresponding sequence of inputs and representing each sheaf as a walk starting in a vertex with an incoming unlabelled edge and

terminating in a vertex with an outgoing unlabelled edge.

We say that a digraph $G = (V, E)$ qualifies as a *WSG for an initially connected digraph* $G' = (V', v_0, E')$ if $SH(\bar{e})$ is defined for every walk $\bar{e}$ in G and in the union of all $SH(\bar{e})$ with $\bar{e}$ a walk in G, the members $\bar{e}'$ with $init(\bar{e}') = v_0$ are exactly the initially rooted walks of G'. For example, the digraph in Figure 10(ii) (Section 8) is a WSG for the FSM in Figure 1.

A walk set or an initially connected digraph can typically be represented by many different WSGs, where one of the preference criteria is whether individual vertices and edge labels in the WSG are concisely describable. In a typical WSG, not only edge labels, but also vertices, are sets (see Section 3.5), so that the usual trick for making them more concisely describable is to make them more symmetric, by extending them with elements whose presence is under the adopted interpretation of the WSG irrelevant.

3.5. Extended Finite State Machines

If an initially connected FSM models a system handling data, its states, inputs and outputs are vectors, each state denoting the current logical state of the system and the current value of its internal data variables, and each input or output denoting a logical input or output signal of the system, respectively, and the data it carries. If such an FSM is too large for explicit representation, it is usually represented by a WSG whose vertices group (to the desired extent) the states of the FSM according to the corresponding logical state and whose edges group (to the desired extent) the transitions of the FSM according to the corresponding logical input signal. Such a WSG is usually interpreted as a machine whose logical states and logical signals have been extended with data and is, hence, called an *extended FSM* (EFSM). There is more than one typical notation for EFSMs [4, 5, 6, 7] and this is one of the reasons why we rather interpret them abstractly, as WSGs with the following special properties:

(i) Vertices are non-intersecting non-empty subsets of the universe from which the states of the represented FSM are drawn.
(ii) For every edge e, $lab(e)$ is a non-empty subset of the universe from which the edges of the represented FSM are drawn and for every edge e' in $lab(e)$, $init(e')$ is in $init(e)$ and $fin(e')$ is in $fin(e)$.

As an example, take the EFSM in Figure 10(i) (Section 8). Its more abstract representation in Figure 10(ii) reveals that it is actually a WSG for the FSM in Figure 1. An EFSM is *deterministic* if the FSM it presumably represents is deterministic.

For an EFSM (V, E) representing an FSM with states drawn from a universe $\mathcal{S}$, we define a *state abstraction operator* $sa(t)$ which in any term t replaces every instance of an s from $\mathcal{S}$ with an instance of that vertex in V which comprises s, i.e., with $sa(s)$, where we assume that $\cup_{v \in V} v$ comprises every s of interest.

3.6. Predicates on Rooted Walks of Digraphs

When looking in a strongly connected digraph $G = (V, v_0, E)$ for a rooted tour which is according to the adopted criteria optimal, one has to evaluate various *predicates on rooted tours*, typically with the help of various *predicates on initially rooted walks* and *predicates on finally rooted walks*. For example, for a test tour (TT) $\bar{\tau}$ in an FSM, one typically wants that it traverses every transition τ of the FSM in a context in which it is recognized, e.g., that $\bar{\tau}$ is of the form $\bar{\tau}' \cdot \tau \cdot \bar{\tau}''$ with $\bar{\tau}'$ a backward SRS and $\bar{\tau}''$ a forward SRS. For every τ, one, hence, defines on the tour a predicate depending on a predicate on tour prefixes and a predicate on tour suffixes.

Let $PT(G)$, $PI(G)$ and $PF(G)$, respectively, denote the set of the employed predicates of each of the three kinds. For a walk $\bar{e}$, let $PT(\bar{e})$, $PI(\bar{e})$ and $PF(\bar{e})$ denote the set of those predicates p in $PT(G)$, $PI(G)$ or $PF(G)$, respectively, for which $p(\bar{e})$ is true. We say [1] that the *predicate system* (PS) $(PT(G), PI(G), PF(G))$ is a *canonical PS* (CPS) if it respects the following restrictions making its predicates easily computable:

RESTRICTION 1. For every predicate p in $PT(G)$, there is a function θ_p such that for every rooted tour $\bar{e}$ in G,
$p(\bar{e}) = \exists \bar{e}', e, \bar{e}''.((\bar{e} = \bar{e}' \cdot e \cdot \bar{e}'') \wedge$
$\theta_p(e, PI(\bar{e}'), PF(e \cdot \bar{e}''), PI(\bar{e}' \cdot e), PF(\bar{e}'')))$,
so that p, if satisfied on $\bar{e}$, has a *witness edge* e, where the evidence for $p(\bar{e})$ is a combination of the properties of e itself, of its past and of its future.

RESTRICTION 2. There is a function β returning a subset of $PI(G)$ such that for every initially rooted walk $\bar{e} = \bar{e}' \cdot e$ in G, $PI(\bar{e}) = \beta(e, PI(\bar{e}'))$, so that all the predicates in $PI(G)$ *can be computed* for $\bar{e}$ recursively, *by a single forward traversal* of $\bar{e}$.

RESTRICTION 3. There is a function φ returning a subset of $PF(G)$ such that for every finally rooted walk $\bar{e} = e \cdot \bar{e}'$ in G, $PF(\bar{e}) = \varphi(e, PF(\bar{e}'))$, so that all the predicates in $PF(G)$ *can be computed* for $\bar{e}$ recursively, *by a single backward traversal* of $\bar{e}$.

To formally define a CPS, one has to specify its generators $PI(\varepsilon)$, $PF(\varepsilon)$, β, φ and θ_p, that are entirely problem-specific.

3.7. Generalized Canonical Predicate Systems

If for a digraph G, some predicates on its initially or finally rooted walks have already been evaluated, one might want to use the information for evaluating further predicates on initially or finally rooted walks, before finally employing all the computed information for evaluating the predicates which are of interest on the rooted tours of G. As in CPSs such cascade evaluation

of predicates is not defined, we in the following define a more general kind of PSs in which the members of $PI(G)$ and $PF(G)$ are grouped according to the stage at which their evaluation is foreseen. In such a PS, a predicate from $PI(G)$ supposed to be evaluated *not later than* during the i-th forward traversal of G will be said to belong to the subset $PI_i(G)$ of $PI(G)$, whereas a predicate from $PF(G)$ supposed to be evaluated *not later than* during the i-th backward traversal of G will be said to belong to the subset $PF_i(G)$ of $PF(G)$:

We say that a PS $(PT(G), PI(G), PF(G))$ is a *generalized CPS* (GCPS) if for some natural number $\omega \geq 1$, the *size* of the GCPS, there have been defined such subsets $\emptyset = PI_0(G) \subseteq PI_1(G) \subseteq \ldots \subseteq PI_\omega(G) = PI(G)$ of $PI(G)$ and subsets $\emptyset = PF_0(G) \subseteq PF_1(G) \subseteq \ldots \subseteq PF_\omega(G) = PF(G)$ of $PF(G)$ that the Restrictions 4, 5 and 6 given below are satisfied. A CPS is a special kind of a GCPS with $\omega = 1$.

Like their respective analogues Restrictions 1, 2 and 3, the three restrictions are intended for simplifying predicate evaluation. They generalize their analogues in the following ways:

(i) They apply to multi-stage predicate evaluation.
(ii) Within the context of a rooted tour $\bar{e} = \bar{e}' \cdot \bar{e}''$ in G, satisfaction of predicates from a $PI_i(G) \setminus PI_{i-1}(G)$ on $\bar{e}'$ is allowed to depend also on satisfaction of predicates from $PF_{i-1}(G)$ on $\bar{e}''$. Consequently, the set of the predicates from $PI_i(G)$ satisfied on $\bar{e}'$ in the context of $\bar{e}''$ is no longer denoted by $PI_i(\bar{e}')$, but by $PI_i(\bar{e}', \bar{e}'')$. In the case of $\bar{e}$ a TT and $\bar{e}'$ terminating with an edge e, one would, for example, be interested in predicates telling for $\bar{e}'$ how well $lab(e \cdot \bar{e}'')$ recognizes $init(e)$, that depending on how well $lab(\bar{e}'')$ recognizes $init(\bar{e}'')$ (for details, see Section 6.1).
(iii) Within the context of a rooted tour $\bar{e} = \bar{e}' \cdot \bar{e}''$ in G, satisfaction of predicates from a $PF_i(G) \setminus PF_{i-1}(G)$ on $\bar{e}''$ is allowed to depend also on satisfaction of predicates from $PI_i(G)$ on $\bar{e}'$. Consequently, the set of the predicates from $PF_i(G)$ satisfied on $\bar{e}''$ in the context of $\bar{e}'$ is no longer denoted by $PF_i(\bar{e}'')$, but by $PF_i(\bar{e}', \bar{e}'')$. In the case of $\bar{e}$ a TT and $\bar{e}''$ starting with an edge e, one would, for example, be interested in predicates telling for $\bar{e}''$ how well $lab(\bar{e}' \cdot e)$ recognizes $fin(e)$, that depending on how well $lab(\bar{e}')$ recognizes $fin(\bar{e}')$ (for details, see Section 6.1).

RESTRICTION 4. For every predicate p in $PT(G)$, there is a function θ_p such that for every rooted tour $\bar{e}$ in G,

$$p(\bar{e}) = \exists \bar{e}', e, \bar{e}''.((\bar{e} = \bar{e}' \cdot e \cdot \bar{e}'') \wedge \theta_p(e, PI(\bar{e}', e \cdot \bar{e}''), PF(\bar{e}', e \cdot \bar{e}''), PI(\bar{e}' \cdot e, \bar{e}''), PF(\bar{e}' \cdot e, \bar{e}'')))),$$

so that p, if satisfied on $\bar{e}$, has a *witness edge* e, where the evidence for $p(\bar{e})$ is a combination of the properties of e itself, of its past and of its future.

RESTRICTION 5. For every $1 \leq i \leq \omega$, there is a function β_i returning a subset of $(PI_i(G) \setminus PI_{i-1}(G))$ such that for every rooted tour $\bar{e} = \bar{e}' \cdot e \cdot \bar{e}''$ in G, $PI_i(\bar{e}' \cdot e, \bar{e}'') \setminus PI_{i-1}(\bar{e}' \cdot e, \bar{e}'') = \beta_i(e, PI_i(\bar{e}', e \cdot \bar{e}''), PF_{i-1}(\bar{e}', e \cdot \bar{e}''), PI_{i-1}(\bar{e}' \cdot e, \bar{e}''), PF_{i-1}(\bar{e}' \cdot e, \bar{e}''))$, where $PI_i(\varepsilon, \bar{e})$ is the same $PI_i(\varepsilon)$ for every $\bar{e}$, so that after all the predicates in $PI_{i-1}(G)$ have been computed for the prefixes of $\bar{e}$ and all the predicates in $PF_{i-1}(G)$ have been computed for the suffixes of $\bar{e}$, all the predicates in $PI_i(G) \setminus PI_{i-1}(G)$ *can be computed* for $\bar{e}' \cdot e$ in the context of $\bar{e}''$ recursively, *by a single forward traversal*.

RESTRICTION 6. For every $1 \leq i \leq \omega$, there is a function φ_i returning a subset of $(PF_i(G) \setminus PF_{i-1}(G))$ such that for every rooted tour $\bar{e} = \bar{e}' \cdot e \cdot \bar{e}''$ in G, $PF_i(\bar{e}', e \cdot \bar{e}'') \setminus PF_{i-1}(\bar{e}', e \cdot \bar{e}'') = \varphi_i(e, PI_i(\bar{e}', e \cdot \bar{e}''), PF_{i-1}(\bar{e}', e \cdot \bar{e}''), PI_i(\bar{e}' \cdot e, \bar{e}''), PF_i(\bar{e}' \cdot e, \bar{e}''))$, where $PF_i(\bar{e}, \varepsilon)$ is the same $PF_i(\varepsilon)$ for every $\bar{e}$, so that after all the predicates in $PI_i(G)$ have been computed for the prefixes of $\bar{e}$ and all the predicates in $PF_{i-1}(G)$ have been computed for the suffixes of $\bar{e}$, all the predicates in $PF_i(G) \setminus PF_{i-1}(G)$ *can be computed* for $e \cdot \bar{e}''$ in the context of $\bar{e}'$ recursively, *by a single backward traversal*.

To formally define a GCPS, one has to specify its generators $PI_i(\varepsilon)$, $PF_i(\varepsilon)$, β_i, φ_i and θ_p.

3.8. Predicate-Aware Unfolding of Digraphs

Assessment of a $PI(\bar{e}', \bar{e}'')$ or $PF(\bar{e}', \bar{e}'')$ for a rooted tour $\bar{e} = \bar{e}' \cdot \bar{e}''$ of a digraph $G = (V, v_0, E)$ with a GCPS $(PT(G), PI(G), PF(G))$ typically requires much more than just a brief glance at G. If a tour construction algorithm tends to refer to this information many times, it is reasonable to construct for G a model from which the information is directly evident. In [1], we, hence, defined $unf(G)$, the *predicate-aware unfolding* of G, a digraph representing the predicate-aware unfolding $unf(\bar{e})$ for each individual rooted tour $\bar{e}$ in G:

(i) For a rooted tour $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, l_i, \bar{c}_i)$ for $1 \leq i \leq k$, in G, $unf(\bar{e}) = e'_1, \ldots, e'_k$, $e'_i = ((v_i, PI(pfx(\bar{e}, i-1), sfx(\bar{e}, k-i+1)), PF(pfx(\bar{e}, i-1), sfx(\bar{e}, k-i+1))), (v_{i+1}, PI(pfx(\bar{e}, i), sfx(\bar{e}, k-i)), PF(pfx(\bar{e}, i), sfx(\bar{e}, k-i))), l_i, \bar{c}_i)$ for $1 \leq i \leq k$.
(ii) The vertices and edges of $unf(G)$ are those of unfoldings $unf(\bar{e})$ of rooted tours $\bar{e}$ in G.

Note the one-to-one correspondence between the rooted tours of G and the walks in $unf(G)$ starting in a $(v_0, PI(\varepsilon), PF)$ and ending in a $(v_0, PI, PF(\varepsilon))$. For such a walk $\bar{e}$ in $unf(G)$, the corresponding tour $\bar{e}'$ in G can be obtained by changing every step $((v, PI, PF), (v', PI', PF'), l, \bar{c})$ into $(v, v', l, \bar{c})$, whereas for every partition $\bar{e}_1 \cdot \bar{e}_2$ of $\bar{e}$ and the corresponding partition $\bar{e}'_1 \cdot \bar{e}'_2$ of $\bar{e}'$, $PI(\bar{e}'_1, \bar{e}'_2)$ and $PF(\bar{e}'_1, \bar{e}'_2)$ are directly evident from $fin(\bar{e}_1)$.

For example, suppose that for every state s_i and every suffix $\bar{\tau}$ of a rooted tour in the FSM M in Figure 1, one needs to know whether s_i is in $FE(lab(\bar{\tau}))$, i.e., whether a predicate p_i is satisfied on $\bar{\tau}$. For $PI(M) = \emptyset$ and $PF(M) = \{p_1, p_2, p_3\}$, $unf(M)$ can be found in Figure 3, as the subgraph consisting of the edges which are labelled, provided that every vertex name s_0^I is interpreted as $(s_0, \emptyset, \{p_i | i \in I\})$. In the subgraph, unfoldings of rooted tours from M start in any vertex of the form s_0^I and end in $s_0^{\{\}}$. For example, the tour "τ_5, τ_8, τ_2" is represented by the walk "e_{10}, e_8, e_3", in which the name of the second vertex indicates that the suffix "τ_8, τ_2" witnesses that the state reached after the first step of the tour is neither s_0 nor s_1. Indeed, the label "$b/e, b/f$" of the suffix is executable only from s_2.

3.9. Determinized Predicate-Aware Unfolding of FSMs

We say that $dunf(M)$, the *determinized predicate-aware unfolding* of an FSM $M = (S, s_0, T)$ with respect to the adopted GCPS, is a WSG representing $unf(\bar{\tau})$ for $\bar{\tau}$ in $Trs(M)$ in sheaves, grouped according to $in(\bar{\tau})$:

(i) The vertices and edges of $dunf(M)$ are those of the unfoldings $dunf(\bar{\tau})$ of $\bar{\tau}$ in $Trs(M)$, where for such a $\bar{\tau}$ with $in(\bar{\tau})$ an $\bar{x} = x_1, \ldots, x_k$, $dunf(\bar{\tau}) = e_1, \ldots, e_k$, $e_i = (\{init(e) | e \in l_i\}, \{fin(e) | e \in l_i\}, l_i, CE(l_i))$ with $l_i = \{cmp(unf(\bar{\tau}', i)) | (\bar{\tau}' \in Trs(M)) \wedge (in(\bar{\tau}') = \bar{x})\}$ for $1 \leq i \leq k$, where CE is the adopted cost-estimation function.

(ii) $Init$ and Fin of the WSG comprise those of its vertices which comprise exclusively elements of the form $(s_0, PI(\varepsilon), \_)$ or $(s_0, \_, PF(\varepsilon))$, respectively.

To see that $dunf(M)$ is indeed a WSG representing the considered walks from $unf(M)$ in the desired way, observe the following:

(i) For every $\bar{\tau}$ in $Trs(M)$, $SH(dunf(\bar{\tau}))$ is exactly $\{unf(\bar{\tau}') | (\bar{\tau}' \in Trs(M)) \wedge (in(\bar{\tau}') = in(\bar{\tau}))\}$.

(ii) The walks $\bar{e}$ in $dunf(M)$ with $init(\bar{e})$ in $Init$ and $fin(\bar{e})$ in Fin are exactly the unfoldings $dunf(\bar{\tau})$ of the walks $\bar{\tau}$ in $Trs(M)$.

As an example, take again the DRT "τ_5, τ_8, τ_2" in the M from Figure 1, where the other two DRTs with the input sequence "a, b, b" are "τ_3, τ_2, τ_1" and "τ_4, τ_8, τ_2", i.e., a $\bar{\tau}_2$ and a $\bar{\tau}_3$. Let the GCPS be as in Section 3.8. In the $unf(M)$ in Figure 3, $unf(\bar{\tau}_1)$, $unf(\bar{\tau}_2)$ and $unf(\bar{\tau}_3)$ are represented by the walks "e_{10}, e_8, e_3", "e_6, e_4, e_1" and "e_9, e_8, e_3", respectively. In $dunf(M)$, depicted in Figure 6 as the subgraph consisting of the edges which are labelled, $dunf(\bar{\tau}_1)$, which is also $dunf(\bar{\tau}_2)$ and $dunf(\bar{\tau}_3)$, is represented by the walk "$e'_{19}, e'_{21}, e'_{13}$".

3.10. Determinized Predicate-Aware Unfolding of EFSMs

We say that $dunf^*(Q)$, the *determinized predicate-aware unfolding* of an EFSM $Q = (V, \{s_0\}, E)$, a concise representation of an FSM $M = (S, s_0, T)$, is with respect to the adopted GCPS a WSG representing $dunf(\bar{\tau})$ for $\bar{\tau}$ in $Trs(M)$ in sheaves, grouped according to $sa(vrt(dunf(\bar{\tau})))$:

(i) The vertices and edges of $dunf^*(M)$ are those of the unfoldings $dunf^*(\bar{\tau})$ of $\bar{\tau}$ in $Trs(M)$, where for such a $\bar{\tau}$ with $sa(vrt(dunf(\bar{\tau})))$ a $\bar{v} = v_1, \ldots, v_k$, $dunf^*(\bar{\tau}) = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, l_i, CE(l_i))$ with $l_i = \{e | (\bar{\tau}' \in Trs(M)) \wedge (dunf(\bar{\tau}') = \ldots, e, \ldots) \wedge (sa(init(e)) = v_i) \wedge (sa(fin(e)) = v_{i+1})\}$ for $1 \leq i \leq k$, where CE is the adopted cost-estimation function.

(ii) $Init$ and Fin of the WSG comprise those of its vertices which comprise exclusively elements of the form $(\{s_0\}, PI(\varepsilon), \_)$ or $(\{s_0\}, \_, PF(\varepsilon))$, respectively.

To see that $dunf^*(M)$ is indeed a WSG representing the considered walks from $dunf(M)$ in the desired way, observe the following:

(i) For every $\bar{\tau}$ in $Trs(M)$, $SH(dunf^*(\bar{\tau}))$ is exactly $\{dunf(\bar{\tau}') | (\bar{\tau}' \in Trs(M)) \wedge (sa(vrt(\bar{\tau}')) = sa(vrt(\bar{\tau})))\}$.

(ii) The walks $\bar{e}$ in $dunf^*(M)$ with $init(\bar{e})$ in $Init$, $fin(\bar{e})$ in Fin and $SH(\bar{e})$ non-empty are exactly the unfoldings $dunf^*(\bar{\tau})$ of the walks $\bar{\tau}$ in $Trs(M)$.

As an example, take the DRT $\bar{\tau} = \tau_3, \tau_6, \tau_8, \tau_2$ in the M from Figure 1, the Q in Figure 10(ii) and the GCSP from Section 3.8. In Figure 3, $unf(\bar{\tau})$ is the walk e_{11}, e_{15}, e_8, e_3, which is in Figure 6 in the sheaf of the walk $dunf(\bar{\tau}) = e'_{16}, e'_{11}, e'_2, e'_1$, which is in $dunf^*(Q)$, depicted in Figure 12 (Section 8.2), in the sheaf of the walk $dunf^*(\bar{\tau}) = e''_{16}, e''_4, e''_2, e''_1$.

3.11. Some Tour Construction Problems on Digraphs

The directed *Rural Postman Problem* (RPP) is defined as follows [8]: Given a strongly connected digraph $G = (V, E)$ and an $\emptyset \subset E' \subseteq E$, find in G a minimum-cost tour $\bar{e}$ such that $E' \subseteq cmps(\bar{e})$.

The directed *Generalized RPP* (GRPP) is defined as follows [1]: Given G as for RPP, a non-empty set Γ of goals and for every g in Γ an $\emptyset \subset E_g \subseteq E$, find in G a minimum-cost tour $\bar{e}$ such that for every g in Γ, $cmps(\bar{e}) \cap E_g \neq \emptyset$. RPP corresponds to the special case with all E_g singleton.

The directed *Generalized GRPP* (G^2RPP) is defined as follows [1]: Given G, Γ and E_g as for GRPP, and a partition Ξ of Γ into non-empty, possibly intersecting subsets, find in G a minimum-cost tour $\bar{e}$ such that for some goal set ξ in Ξ, $cmps(\bar{e}) \cap E_g \neq \emptyset$ for every g in ξ. GRPP corresponds to the special case with Ξ singleton.

If for a G, one requires that the produced tour goes through its root v_0, so that it can be interpreted as starting in it, this must be specified as an additional goal g_0 in Γ. One enhances G with a zero-cost auxiliary

loop e_0 in v_0 and defines $E_{g_0} = \{e_0\}$. For G^{2}RPP, g_0 must be included in every goal set ξ in Ξ.

4. CONCEPTUAL THREADS OF THE PAPER

4.1. The Central Concept: Problem Model

Our work in [1] was motivated by a wish to polish the test generation method of [9] to perfection, so that it starts generating tests which implement the underlying testing strategy optimally. The algorithm of [9], like most traditional test generation methods primarily concerned with the cost of the test, looks for a test which, when applied to a system functioning as expected and currently in its root state, terminates by returning the system to the root. The problem addressed in [1] and the present paper is, hence, construction of a *test tour* (TT), although in the present paper with an understanding that for a nondeterministic system, one might at some point during testing want to change the rest of the test, to make use of the evidence collected so far.

Looking for a conceptual framework in which we could systematically identify possible improvements for [9], we became aware of the following:

(i) Behind any algorithm helping with system analysis, synthesis or presentation, there is a formal model of the problem being solved, which is more than just a formal model of the concerned system. *Once a sufficiently explicit and precise problem model is constructed, finding a solution to the problem is simply a search problem on the model.*

(ii) Search problems have the nice property that they tend to be generic, implying that at least for the simpler ones, one can reasonably expect that efficient tools for their solving already exist. It is, hence, advisable that the first attempt to any domain-specific problem is its reduction to a generic search problem. Even if it eventually turns out that the encountered search problem is a yet unknown one, such reduction is extremely useful, for it reveals the inherent nature of the domain-specific problem.

(iii) Even when working with an approximate problem model, it is important to have a clear idea of a precise model, i.e., of what vital information is being neglected, so that one can make an informed estimation of the extent to which solutions found on the employed model will be optimal.

Traditional TT construction methods actually tend to construct and exploit an explicit problem model, but those models are in general not precise and the methods provide no insight on what a precise model would be. With the method of [1], one can build a precise problem model for many well-known testing strategies, including that of [9] and one resulting in absolutely optimal tests.

4.2. System, Tests, Runs, System Model and Problem Model

In TT construction, one basically assumes that the *system* is exactly the FSM specified by the system model, although for an NFSM, one usually makes additional assumptions on how strongly and in what forms the specified nondeterminism can actually be exhibited by the system. For the universe in which to look for an optimal *test*, one basically assumes that it comprises any *sequence* (see Section 3.1) of inputs to which the FSM can react exclusively with a *run* consuming all the inputs, terminating in the root state and comprising no transition declared as forbidden.

The *system model* is syntactically an initially connected rooted *directed graph* (see Section 3.2). The digraph is either directly interpreted as the *finite state machine* (see Section 3.3) or is interpreted as a *walk-sheaves graph* for a digraph representing the FSM (see Section 3.4), i.e., as an *extended finite state machine* (see Section 3.5).

For every test, the system has one or, if it is nondeterministic, possibly more runs which it can activate in response. For every test run, the system model has one or, if it is an EFSM, possibly more rooted tours which are representations of the run. If the model is an EFSM, each of its rooted tours might be representing multiple alternative test runs.

The *problem model* is a strongly connected digraph consisting of a central subgraph, an auxiliary vertex acting as the root, an auxiliary loop in the root, auxiliary edges from the root to the entry points of the central subgraph, auxiliary edges from the exit points of the central subgraph to the root, and possibly also some auxiliary edges leading from an exit to an entry point of the central subgraph. *In the central subgraph, every walk from an entry to an exit point represents one or more test runs*, which are in the model, unlike in the system model, *annotated with additional information facilitating assessment of their optimality.*

Technically speaking, the central subgraph represents a set of annotated test runs either directly or as a WSG for the set (remember Section 3.4). Runs corresponding to the same test are always represented by the same walk and one also mimics, as much as possible and convenient, the sheaving of the corresponding walks in the system model, the latter to keep the problem model concise. Ideally, one represents all the candidate test runs and annotates them with all the available useful information. However, to keep the graph reasonably small, one typically represents just some runs, advisably those corresponding to some of those tests which seem the most promising, among them advisably also an emergency solution to the problem, typically a test previously generated by some less sophisticated, but consequently less complex method.

To select a test, one first selects a rooted tour in the problem model. If the model has been properly

constructed, the projection of the tour onto the edges of the central subgraph denotes a non-empty set of test runs and thereby a non-empty set of the corresponding tests, one of which is then systematically or at random selected as the generated test. *For some strategies, the tour is allowed to enter and leave the central subgraph more than once, with the traversed auxiliary edges denoting concatenation of the tests represented by the tour segments lying in the subgraph.*

For illustration, let us take a brief glance at the running example of Sections 6 to 8. Suppose that the system is the NFSM from Figure 1. If the system model is the FSM itself and the testing strategy is to check transitions by traversing them and checking their final state with SRSs of any kind, a possible problem model is that in Figure 6, in which the labelled edges, i.e., the edges of the central subgraph, each represent, as evident from Figure 3, a set of annotated copies of edges from Figure 1 (annotations are attached to edge endpoints). The entry points of the subgraph are those with an unlabelled (i.e. auxiliary) edge coming from the problem model root v_0, whereas its exit points are those with an outgoing auxiliary edge. In the problem model, representations of test runs are sheaved, to indicate that some tests have multiple corresponding runs. For example, the walk "$e'_{19}, e'_{21}, e'_{13}$" represents a sheaf comprising the walks "e_6, e_4, e_1", "e_9, e_8, e_3" and "e_{10}, e_8, e_3" from Figure 3, which are annotated copies of the rooted tours "τ_3, τ_2, τ_1", "τ_4, τ_8, τ_2" and "τ_5, τ_8, τ_2" from Figure 1, i.e., of the runs corresponding to the test "a, b, b".

Now suppose that the system FSM from Figure 1 is represented by a WSG, the NEFSM in Figure 10(ii). The central subgraph of the problem model then becomes, in its default version, the digraph in Figure 12, a WSG for the set of the walks of interest in the central subgraph of the more explicit problem model from Figure 6. The WSG mimics the TT sheaving introduced in Figure 10(ii). For example, as transitions τ_6 and τ_7 from Figure 1 are in Figure 10(ii) represented by the same edge, the corresponding e'_4, e'_6 and e'_{11} from Figure 6 are in Figure 12 also represented by the same edge. However, for convenience, we afterwards derive a slightly more explicit representation of the test universe, systematically partially unfolding the WSG first into the digraph in Figure 13 and finally into the central subgraph of the final version of the problem model, given in Figure 15.

4.3. Defining and Representing the Quality of Tests

The purpose of applying a test is to provoke a system run satisfying the specific requirements of the adopted testing strategy. We assume that the requirements are formalized by stating that the actualized run should satisfy one of the given *alternative sets of predicates*. The ideal is then to *find one of the cheapest tests among those which in the expected case provoke a run satisfying the requirements as much as possible*, where the expected case may be defined as anything between the best and the worst case. In any case, it is assumed that each of the predicates is such that whenever it is satisfied, the run has a step, i.e., a system transition executed in a specific context, acting as a *witness*, where there might be more than one witness per predicate. Some testing strategies also introduce *additional cost criteria*, to discourage transitions whose presence in the actualized run seems to reduce the discriminating power of the test, e.g., critical converging transitions (they complicate state recognition [10]).

The witnessing power of a step in a test run depends on specific properties of the transition itself, of the past transitions and of the future transitions. We, hence, assume that the *predicates of interest on rooted tours* in the system FSM are *formalized in terms of predicates on initially or finally rooted walks* in the FSM, in a form facilitating that the predicates are evaluated by traversing the tour a predefined number of times. In [1], we expected the employed predicates to form a *canonical predicate system* (see Section 3.6). With the generalization of the method proposed in Section 6, it suffices that they form a *generalized canonical predicate system* (see Section 3.7).

In the problem model, ideally one represents each test run of interest using its precise predicate-aware unfolding. In such an unfolding, every vertex is annotated with all the predicates of interest which are satisfied on the incoming part of the walk and with all those which are satisfied on the rest of the walk, so that for every step, the witnessing power can be computed by considering just its own properties and the annotations of its initial and its final vertex. In the annotations, one may also decide to *underestimate predicate satisfaction*, but only if the adopted PS is such that this cannot lead to overestimation of the witnessing power of the run, for this could lead to selection of a non-optimal test. The restriction applies also for considering concatenations of test runs represented in the central subgraph, because vertex annotations in the representations of the constituent runs often provide only approximate information on the satisfaction of predicates on various prefixes and suffixes of the overall run. This explains why construction of test runs by concatenation is for some testing strategies forbidden. Where it is not, one may in approximate solving of the problem conveniently concentrate on the representation of runs corresponding to tests with a limited goal, for those tests can be freely combined.

For a system model which is an FSM from which every forbidden transition has already been deleted, the precise problem model not combining representations of test runs into sheaves is assumed to have as its central subgraph the *predicate-aware unfolding of the FSM* (see Section 3.8). To construct the subgraph, one traverses increasingly more unfolded and annotated

versions of the FSM, alternately in the forward and in the backward direction, in each traversal generating an even more unfolded and annotated instance. More precisely, whenever a vertex is encountered after traversing a specific sequence of edges, some additional predicates on the sequence are evaluated and a copy of the vertex, additionally annotated with the newly computed information, is added to the new instance of the graph, together with a copy of the just traversed edge. Where different forward or backward walks to a vertex induce different annotations of the vertex copy, the vertex is virtually split. For example, the state s_1 of the FSM in Figure 1 is in its predicate-aware unfolding, the central subgraph of the digraph in Figure 3, split into vertices $s_1^{\{0\}}$ and $s_1^{\{2\}}$, because with respect to the adopted CPS, s_1 has two kinds of outgoing walks, characterized by $FE(z)$ of their label z being either $\{s_0\}$ or $\{s_2\}$.

If some test runs are to be represented in a sheaf, central subgraph construction requires that they are *traversed in parallel*, so that the generated copies of their segments can be sheaved immediately. Parallel traversal is employed also for walks which are already combined in a sheaf. When unfolding an NEFSM, it might even be necessary to parallelly traverse multiple sheaves, for it might be that each of them comprises some of the runs corresponding to a specific test and, hence, requiring sheaving. Still, the result of the algorithm is in all cases a graph representing the predicate-aware unfolding of every possible TT. If the system model is an NFSM or an (N)EFSM, the precise problem model, in its default version, is assumed to have as its central subgraph the *determinized predicate-aware unfolding of the system model* (see Sections 3.9 and 3.10, respectively), or a WSG equivalent to it (see the last paragraph of Section 3.4). To construct an approximate problem model, one during each traversal in central subgraph construction considers only the runs corresponding to the pre-selected candidate tests and/or underestimates satisfaction of predicates when they are evaluated.

If a rooted tour in the problem model represents a sheaf of test runs, it represents the sheaf in segments represented by individual edges of its central subgraph. In the early versions of the central subgraph, such a segment represents a single step of the system, but it need not be so in the later versions. Longer segments mean more explicit representation of test runs and, hence, better chances that the sum of the costs and the union of the witnessing capabilities assumed for the sequences of steps represented by individual segments are a good prediction of the cost and the witnessing power, respectively, of the run actualized in the case that the tour becomes the selected one and one of the corresponding tests is applied to the system. Thus, it sometimes pays to introduce for a walk in the central subgraph a *by-pass edge explicitly representing the sheaf of the test (sub)runs defined by the walk*. *Splitting of an edge* representing a sheaf into edges which each represent just a subset of the members of the sheaf might also improve prediction of costs and benefits. Most important, however, is to *eliminate every walk representing an empty sheaf*. To make such a walk sufficiently explicit for deletion, some *vertex splitting* might be necessary.

4.4. Solving the Generic Search Problem

In the problem model, each edge in the central subgraph, i.e., the corresponding sequence of steps of the actualized test run, is believed to have a specific cost and to provide witness of satisfaction for some specific predicates among those of interest on the TT. *The search problem is to find in the graph an optimal rooted tour which for a sufficient set of predicates passes over at least one witness for every predicate in the set.* In an unfortunate case, there is among the predicate sets which the adopted testing strategy considers sufficient no set with a witness for every member. Besides, even if witnesses for a sufficient set do exist, it might be impossible to construct a rooted tour visiting them all, for such a visit might require that the tour enters and leaves the central subgraph more than once, but the nature of the adopted PS might be such that this is unacceptable. Employing a more precise problem model might help, but sometimes it is unavoidable to resort to a less ambitious or substantially different testing strategy.

In [1], we demonstrated that, contrary to the common belief [11], the inherent nature of the TT construction problem is not that of the standard *Rural Postman Problem* (RPP) (see Section 3.11), but that of the *Generalized RPP* (GRPP) (see Section 3.11). For the testing strategies unnecessarily particular about specifics of the constructed test, the search problem to solve on the problem model might even be an instance of the *Generalized GRPP* (G^2RPP) (see Section 3.11), as such strategies typically define multiple alternative sets of predicates to satisfy.

The three generic search problems simply reduce to the well-known Travelling Salesman Problem (TSP) or its corresponding generalizations GTSP and G^2TSP [1], respectively, of which TSP and GTSP are already well investigated, while G^2TSP is a generic problem for further study.

Let us also mention that when writing [1], we were not aware of a previous initiative for reducing the TT generation problem to GRPP [12, 13, 14, 15]. Chen calls GRPP the Selecting Chinese Postman Problem and solves it by an approximate algorithm analogous to that of [11] for RPP. The idea of the algorithm is to *symmetrize the graph, so that the tour construction problem reduces to Euler tour construction.* Note that the strategy is applicable to any tour construction problem aiming at traversing all edges in a satisfactory

edge set, including G^{2}RPP: One just has to make an adequate choice of the objective pursued during the graph symmetrization. Anyhow, none of the Chen's TT generation methods is a generic method for integrated and precise handling of all the usual optimization concerns.

4.5. Adaptive Testing

In adaptive testing, the constructed test repeatedly undergoes a correction, after a part of the test has already been applied to the system, on which it has provoked a specific sequence of outputs. Computing a correction for the test, one, thus, takes care that in the problem model, *only test runs starting with the already observed I/O sequence are represented by rooted tours passing from the root vertex to the central subgraph just once*. To achieve that, one introduces a predicate telling whether a test run prefix is non-empty. In the problem model, this predicate partitions those vertices whose visit corresponds to a visit to the root state of the system according to whether or not the visit is the start of the test run. With some edge splitting, by-pass edges and edge deletion, one can then easily modify the central subgraph in such a way that, with adequate placement of auxiliary edges, the problem model has the desired form. Limiting the search of the model to tours passing from the root to the central subgraph just once will do the rest. We also recommend that *computation of corrections to the test always starts by updating the system model in compliance with the observed outputs and adapting the testing strategy accordingly.*

5. THE ORIGINAL METHOD AND A SLIGHT IMPROVEMENT

The method solves the following problem: Given

(i) an initially connected and deterministic specification FSM $M = (S, s_0, X, Y, T)$,
(ii) a set Φ_M of its faulty implementation FSMs, all defined on the same input and output alphabets as M, initially connected, deterministic and completely specified, where an M' is considered faulty if there is a sequence of inputs to which both M and M' are able to react, but with different sequences of outputs,
(iii) a list T' of those transitions in T which are considered forbidden and
(iv) a testing strategy formalizable in the intended manner,

produce a sequence of inputs which

(i) whenever applied to the root state of M, provokes no transition in T' and terminates by returning M to the root and
(ii) is, according to the adopted testing strategy, among the optimal tests for discriminating FSMs in Φ_M against M, i.e., satisfies the specific requirements of the strategy as much as possible and at a minimum cost, where the cost of a test is defined as the cost of applying it to the root of M.

First, one enhances the cost vectors of M's transitions with costs with respect to the *additional criteria* of the adopted testing strategy, if any, thereby enhancing M into an $M^+ = (S, s_0, T^+)$. One then prunes M^+ into an $M^* = (S^*, s_0, T^*)$, its maximal strongly connected and in s_0 rooted subgraph containing no transition corresponding to a transition in T'. It is appropriate to say that M^* is *the system model* in the sense of Section 4.2. This, however, does not preclude the derived problem model from referring to information available only in M. A fault-model-driven testing strategy will require reference also to Φ_M.

M^* is the FSM on which a test $\bar{x}^*$ is subsequently constructed as $in(\bar{\tau}^*)$ of a rooted tour $\bar{\tau}^*$. Hence, one completes the formalization of the testing strategy by defining for M^* an adequate CPS $(PT(M^*), PI(M^*), PF(M^*))$ and then constructs a digraph $G^* = (V^*, E^*)$, *the central subgraph of the problem model*, as $unf(M^*)$ (see Section 3.8) or an approximation of it.

For every edge $e = ((s, PI, PF), (s', PI', PF'), l, \bar{c})$ in E^*, one computes $PT(e)$, the set of all the predicates p from $PT(M^*)$ for which the corresponding step of any rooted tour in M^* is believed to be a *witness* of satisfaction on the tour, i.e., of all p with $\theta_p((s, s', l, \bar{c}), PI, PF, PI', PF')$ true. For a walk $\bar{e} = e_1, \ldots, e_k$ in G^*, the cumulative witnessing power $PT(\bar{e})$ can be computed as $\cup_{1 \le i \le k} PT(e_i)$.

G^* is enhanced into a $G = (V, v_0, E)$, *the problem model*, by introducing a dummy root v_0, an auxiliary loop in the root (see the last paragraph of Section 3.11), an auxiliary edge from v_0 to every vertex $(s_0, PI(\varepsilon), PF)$ in V^* (these is are the vertices qualifying as the start of a walk representing $\bar{\tau}^*$) and an auxiliary edge to v_0 from every vertex $(s_0, PI, PF(\varepsilon))$ in V^* (these are the vertices qualifying as the end of a walk representing $\bar{\tau}^*$). Normally, all the auxiliary edges are unlabelled and of cost $\bar{0}$.

One then finds in G *a minimum-cost tour* $\bar{e}^*$ comprising the root loop and a sufficient set of witnesses in E^*, and interprets it as a rooted tour whose projection $\bar{e}^+$ onto the edges in E^* corresponds to $\bar{\tau}^*$ in M^*. The search problem is a GRPP or a G^{2}RPP (or, in numerous specific cases, just an RPP), with individual predicates p in $PT(M^*)$ interpreted as goals g in Γ and individual $PT(e)$ as lists of those g for which e is in E_g. If some vital edges from $unf(M^*)$ are not sufficiently precisely represented in G^* or if the testing strategy is infeasible for M, no adequate $\bar{e}^*$ is found and one has to repeat the TT construction procedure with a better approximation of $unf(M^*)$ or a different strategy.

Particularly for conducting the method in an approximate manner, it is very *helpful* if the adopted testing strategy can be formalized in such a way

that the specified PS satisfies the following *additional restrictions*:

RESTRICTION 7. Every θ_p is such that for every transition τ in T^*, $PI_1 \subseteq PI_1' \subseteq PI(M^*)$, $PF_1 \subseteq PF_1' \subseteq PF(M^*)$, $PI_2 \subseteq PI_2' \subseteq PI(M^*)$ and $PF_2 \subseteq PF_2' \subseteq PF(M^*)$, $\theta_p(\tau, PI_1, PF_1, PI_2, PF_2) \Rightarrow \theta_p(\tau, PI_1', PF_1', PI_2', PF_2')$, so that underestimation of satisfaction of predicates from $PI(M^*)$ or $PF(M^*)$ can never lead to overestimation of satisfaction of a predicate from $PT(M^*)$.

RESTRICTION 8. β is such that for every transition τ in T^* and $PI \subseteq PI' \subseteq PI(M^*)$, $\beta(\tau, PI) \subseteq \beta(\tau, PI')$, so that underestimation of satisfaction of predicates from $PI(M^*)$ can never lead to overestimation in a subsequent assessment of satisfaction of such a predicate.

RESTRICTION 9. φ is such that for every transition τ in T^* and $PF \subseteq PF' \subseteq PF(M^*)$, $\varphi(\tau, PF) \subseteq \varphi(\tau, PF')$, so that underestimation of satisfaction of predicates from $PF(M^*)$ can never lead to overestimation in a subsequent assessment of satisfaction of such a predicate.

RESTRICTION 10. $PI(\varepsilon)$ is empty, so that substituting a $PI(\bar{\tau})$ with $PI(\varepsilon)$ can never be an overestimation of satisfaction of predicates from $PI(M^*)$.

RESTRICTION 11. $PF(\varepsilon)$ is empty, so that substituting a $PF(\bar{\tau})$ with $PF(\varepsilon)$ can never be an overestimation of satisfaction of predicates from $PF(M^*)$.

If *the first three restrictions* are met, one may safely underestimate the extent to which individual initially or finally rooted walks in M^* satisfy individual predicates in $PI(M^*)$ or $PF(M^*)$, respectively. If *the remaining two* are also met, one may also safely let $\bar{e}^*$ pass from v_0 to the G^* subgraph of G multiple times [1]. In both cases, *safely* means that the TT construction procedure will not overestimate the witnessing power of individual test steps.

If it is necessary to *discourage $\bar{e}^*$ from passing from v_0 to G^* multiple times*, one introduces for the edges of G an additional cost criterion of maximal importance according to which the edges outside G^* are more expensive. At this point, we observe that it suffices to make the edges passing from v_0 to G^* more expensive, so from now on, this will be the assumed approach. If $\bar{e}^*$ passes from v_0 to G^* multiple times in spite of those edges being expensive, this indicates the need for a more precise implementation of the adopted testing strategy or for a less ambitious strategy.

At this point, we propose *a slight improvement*: We observe that if Restrictions 7 to 9 are satisfied, it is safe to introduce an *additional auxiliary zero-cost edge* between any such pair of vertices (s_0, PI, PF) and (s_0, PI', PF') in G^* that $PI' \subseteq PI$ and $PF \subseteq PF'$, because a jump in $\bar{e}^+$ between such a pair of vertices can only lead to *under*estimation of the satisfaction of a predicate from $PT(M^*)$ on $\bar{\tau}^*$ [1]. Such an edge is recommended wherever it additionally facilitates consideration of a concatenation of rooted tours from M^* which is not represented in G^*.

In all our examples below, the ideal is to find a TT satisfying every predicate in $PT(M^*)$, where we assume that for every explicitly mentioned p in $PT(M^*)$, $PT(M^*)$ also comprises every abstraction of p which can be expressed without further enhancements of $PI(M^*)$ and $PF(M^*)$, so that if no TT satisfying p exists, the TT construction procedure automatically starts pursuing approximations of p [1].

6. GENERALIZATION TO GENERALIZED CANONICAL PREDICATE SYSTEMS

6.1. Motivation

Suppose that $PT(M^*)$ for every transition $\tau = (s, s', z, \bar{c})$ in T^* comprises a predicate $p_\tau(\bar{\tau})$ defined as
$\exists \bar{\tau}', \bar{\tau}''.((\bar{\tau} = \bar{\tau}' \cdot \tau \cdot \bar{\tau}'') \wedge$
$(\forall s'' \in S \setminus \{s\}.(p_{s''}(\bar{\tau}') \wedge p'_{s''}(\tau \cdot \bar{\tau}'))) \wedge$
$(\forall s'' \in S \setminus \{s'\}.(p_{s''}(\bar{\tau}' \cdot \tau) \wedge p'_{s''}(\bar{\tau}''))))$
with predicates $p_s(\bar{\tau})$ in $PI(M^*)$ for states s in S equivalent to $(s \in BE(lab(\bar{\tau})))$ and formalized as
$\exists \bar{\tau}', \tau'.((\bar{\tau} = \bar{\tau}' \cdot \tau') \wedge$
$\forall s' \in S.(\neg p_{s'}(\bar{\tau}') \Rightarrow \neg \exists (s', s, lab(\tau'), \_) \in T))$
and with predicates $p'_s(\bar{\tau})$ in $PF(M^*)$ for states s in S equivalent to $(s \in FE(lab(\bar{\tau})))$ and formalized as
$\exists \tau', \bar{\tau}'.((\bar{\tau} = \tau' \cdot \bar{\tau}') \wedge$
$\forall s' \in S.(\neg p'_{s'}(\bar{\tau}') \Rightarrow \neg \exists (s, s', lab(\tau'), \_) \in T))$.

Such a CPS formalizes transition-oriented testing in which the initial state of every transition is checked both by a backward and by a forward SRS, and so is its final state [1]. It might, however, be that a state-recognition capability which an I/O sequence has in the specification FSM M is not preserved in every M' in Φ_M. Hence, to increase the reliability of a state check, one might want to check not only the state, but also some states in its past and/or some states in its future.

Suppose that the above strategy for testing a transition τ is enhanced in such a way that the predecessor of its initial state and the successor of its final state are also double-checked. Predicates $p_\tau(\bar{\tau})$ in $PT(M^*)$ for transitions $\tau = (s, s', z, \bar{c})$ in T^* then become
$\exists \bar{\tau}', \bar{\tau}''.((\bar{\tau} = \bar{\tau}' \cdot \tau \cdot \bar{\tau}'') \wedge$
$(\forall s'' \in S \setminus \{s\}.(p_{s''}(\bar{\tau}') \wedge p'_{s''}(\tau \cdot \bar{\tau}''))) \wedge$
$(\forall s'' \in S.(p_{s'',-1}(\bar{\tau}') \wedge p'_{s'',-1}(\bar{\tau}', \tau \cdot \bar{\tau}''))) \wedge$
$(\forall s'' \in S \setminus \{s'\}.(p_{s''}(\bar{\tau}' \cdot \tau) \wedge p'_{s''}(\bar{\tau}''))) \wedge$
$(\forall s'' \in S.(p_{s'',+1}(\bar{\tau}' \cdot \tau, \bar{\tau}'') \wedge p'_{s'',+1}(\bar{\tau}''))))$
with predicates $p_{s,-1}(\bar{\tau}, \bar{\tau}'')$ (actually $p_{s,-1}(\bar{\tau})$) in $PI(M^*)$ providing information on the recognition, by the corresponding tour prefix, of the state just before (hence the -1) the splitting point of a tour $\bar{\tau} \cdot \bar{\tau}''$, for states s in S formalized as
$\exists \bar{\tau}', \tau'.((\bar{\tau} = \bar{\tau}' \cdot \tau') \wedge ((init(\tau') = s) \vee p_s(\bar{\tau}')))$,

	v_1		v_2		v_3		v_4		v_5		v_6		v_7		v_8		v_9		v_{10}
s	0	τ_3	1	τ_6	2	τ_7	2	τ_8	1	τ_2	0	τ_1	0	τ_4	2	τ_8	1	τ_2	0
p		τ_3	0	τ_6	01	τ_7	01	τ_8	02	τ_2	12	τ_1	12	τ_4	0	τ_8	02	τ_2	12
p'	12	τ_3	0	τ_6	0	τ_7	01	τ_8	2	τ_2	2	τ_1	12	τ_4	01	τ_8	2	τ_2	
p_{-1}		τ_3	0	τ_6	01	τ_7	012	τ_8	012	τ_2	012	τ_1	012	τ_4	012	τ_8	02	τ_2	012
p'_{-1}		τ_3	012	τ_6	01	τ_7	02	τ_8	012	τ_2	12	τ_1	02	τ_4	012	τ_8	012	τ_2	12
p_{+1}	01	τ_3	012	τ_6	012	τ_7	012	τ_8	012	τ_2	012	τ_1	02	τ_4	012	τ_8	012	τ_2	
p'_{+1}	01	τ_3	02	τ_6	012	τ_7	12	τ_8	02	τ_2	012	τ_1	012	τ_4	12	τ_8	0	τ_2	

TABLE 1. An abstract representation of the example predicate-aware tour unfolding from Section 6.1.

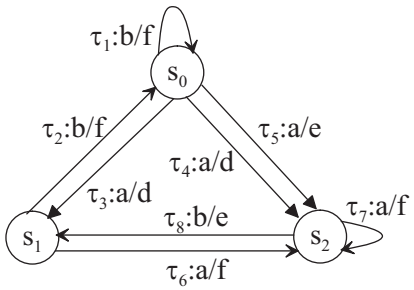

FIGURE 1. The M of the running example.

predicates $p'_{s,-1}(\bar{\tau}, \bar{\tau}'')$ in $PI(M^*)$ providing information on the recognition, by the corresponding tour suffix, of the state just before (hence the -1) the splitting point of a tour $\bar{\tau} \cdot \bar{\tau}''$, for states s in S formalized as

$\exists \bar{\tau}', \tau'.((\bar{\tau} = \bar{\tau}' \cdot \tau') \wedge ((init(\tau') = s) \vee p'_s(\tau' \cdot \bar{\tau}'')))$,

predicates $p_{s,+1}(\bar{\tau}'', \bar{\tau})$ in $PF(M^*)$ providing information on the recognition, by the corresponding tour prefix, of the state just after (hence the +1) the splitting point of a tour $\bar{\tau}'' \cdot \bar{\tau}$, for states s in S formalized as

$\exists \bar{\tau}', \tau'.((\bar{\tau} = \tau' \cdot \bar{\tau}') \wedge ((fin(\tau') = s) \vee p_s(\bar{\tau}'' \cdot \tau')))$

and predicates $p'_{s,+1}(\bar{\tau}'', \bar{\tau})$ (actually $p'_{s,+1}(\bar{\tau})$) in $PF(M^*)$ providing information on the recognition, by the corresponding tour suffix, of the state just after (hence the +1) the splitting point of a tour $\bar{\tau}'' \cdot \bar{\tau}$, for states s in S formalized as

$\exists \bar{\tau}', \tau'.((\bar{\tau} = \tau' \cdot \bar{\tau}') \wedge ((fin(\tau') = s) \vee p'_s(\bar{\tau}'))).$

The enhanced PS is not a CPS, but is nevertheless easily tractable, because it is a GCPS (see Section 3.7), with $\omega = 2$, $PI_1(M^*) = \{p_s | s \in S\}$, $PF_1(M^*) = \{p'_s | s \in S\}$, $PI_2(M^*) = \cup_{s \in S}\{p_{s,-1}, p'_{s,-1}\}$ and $PF_2(M^*) = \cup_{s \in S}\{p_{s,+1},\ p'_{s,+1}\}$. For illustration, we in Table 1 with respect to the GCPS unfold the routed tour $\bar{\tau} = \tau_3, \tau_6, \tau_7, \tau_8, \tau_2, \tau_1, \tau_4, \tau_8, \tau_2$ from the FSM in Figure 1, indicating for each predicate the states for which it is true in the specific vertex of the tour. For example, the "12" in the field (p'_{+1}, v_4) indicates that $p'_{s_i,+1}(pfx(\bar{\tau}, 3), sfx(\bar{\tau}, 6))$ is true for $i \in \{1, 2\}$.

6.2. Generalization of the Optional Restrictions

For now on, we assume that the adopted PS is a GCPS. Consequently, we have to generalize two of the five optional restrictions implementing the idea that underestimation of predicate satisfaction should better never lead to its overestimation, namely Restrictions 8 and 9, which, respectively, become:

RESTRICTION 12. For every $1 \leq i \leq \omega$, β_i is such that for every transition τ in T^*, $PI_i \subseteq PI''_i \subseteq PI_i(M^*)$, $PF_{i-1} \subseteq PF''_{i-1} \subseteq PF_{i-1}(M^*)$, $PI'_{i-1} \subseteq PI'''_{i-1} \subseteq PI_{i-1}(M^*)$ and $PF'_{i-1} \subseteq PF'''_{i-1} \subseteq PF_{i-1}(M^*)$ imply $\beta_i(\tau, PI_i, PF_{i-1}, PI'_{i-1}, PF'_{i-1}) \subseteq \beta_i(\tau, PI''_i, PF''_{i-1}, PI'''_{i-1}, PF'''_{i-1})$, so that underestimation of satisfaction of predicates from $PI_i(M^*)$ or $PF_{i-1}(M^*)$ can never lead to overestimation in a subsequent assessment of satisfaction of a predicate from $PI_i(M^*) \setminus PI_{i-1}(M^*)$.

RESTRICTION 13. For every $1 \leq i \leq \omega$, φ_i is such that for every transition τ in T^*, $PI_i \subseteq PI''_i \subseteq PI_i(M^*)$, $PF_{i-1} \subseteq PF''_{i-1} \subseteq PF_{i-1}(M^*)$, $PI'_i \subseteq PI'''_i \subseteq PI_i(M^*)$ and $PF'_i \subseteq PF'''_i \subseteq PF_i(M^*)$ imply $\varphi_i(\tau, PI_i, PF_{i-1}, PI'_i, PF'_i) \subseteq \varphi_i(\tau, PI''_i, PF''_{i-1}, PI'''_i, PF'''_i)$, so that underestimation of satisfaction of predicates from $PI_i(M^*)$ or $PF_i(M^*)$ can never lead to overestimation in a subsequent assessment of satisfaction of a predicate from $PF_i(M^*) \setminus PF_{i-1}(M^*)$.

6.3. Generalization of the Unfolding Algorithm

For a GCPS, computation of $unf(M^*)$ requires more than a single forward and a single backward traversal of M^*. An adequate algorithm is given in Figure 2.

The algorithm constructs $G^* = (V^*, E^*)$, the central subgraph of the problem model, taking as its initial version $M^* = (S^*, s_0, T^*)$, the system model, and then generating versions with more and more information on the satisfaction of the predicates in $PI(M^*)$ and $PF(M^*)$. The number of iterations of the main loop (lines 1-26) is ω, the size the GCPS.

The i-the iteration (lines 2-25) consists of two parts. In the first part (lines 2-15), one adds information on the satisfaction of the predicates in $PI_i(M^*) \setminus PI_{i-1}(M^*)$, if any. In the first iteration, execution of the body of the first part (lines 3-14) is mandatory, as the graph must be at least rewritten into the expected format. In the body (lines 17-24) of the second part (lines 16-25), one adds information on the satisfaction of the predicates in $PF_i(M^*) \setminus PF_{i-1}(M^*)$.

```
1  for i := 1 to ω do
2    if (i = 1) ∨ (PI_i(M*) \ PI_{i−1}(M*) ≠ ∅) then
3      if i = 1 then Open := {(s_0, PI_1(ε), ∅)} else Open := {(s_0, PI_i(ε), PF)|(s_0, PI_{i−1}(ε), PF) ∈ V*} endif;
4      V* := ∅; E' := ∅;
5      while Open ≠ ∅ do
6        Move an (s, PI, PF) from Open to V*;
7        if i = 1 then Edges := {((s, PI, PF), (s', β_1((s, s', x/y, c̄), PI, ∅, ∅, ∅), ∅), x/y, c̄)
8                                |(s, s', x/y, c̄) ∈ T*}
9        else Edges := {((s, PI, PF), (s', PI' ∪ β_i((s, s', x/y, c̄), PI, PF, PI', PF'), PF'), x/y, c̄)
10                      |((s, PI ∩ PI_{i−1}(M*), PF), (s', PI', PF'), x/y, c̄) ∈ E*}
11       endif;
12       Open := Open ∪ ({v'|∃(_, v', _, _) ∈ Edges} \ V*); E' := E' ∪ Edges
13     endwhile;
14     E* := E'
15   endif;
16   if PF_i(M*) \ PF_{i−1}(M*) ≠ ∅ then
17     Open := {(s_0, PI, PF_i(ε))|(s_0, PI, PF_{i−1}(ε)) ∈ V*}; V* := ∅; E' := ∅;
18     while Open ≠ ∅ do
19       Move an (s', PI', PF') from Open to V*;
20       Edges := {((s, PI, PF ∪ φ_i((s, s', x/y, c̄), PI, PF, PI', PF')), (s', PI', PF'), x/y, c̄)
21                 |((s, PI, PF), (s', PI', PF' ∩ PF_{i−1}(M*)), x/y, c̄) ∈ E*};
22       Open := Open ∪ ({v|∃(v, _, _, _) ∈ Edges} \ V*); E' := E' ∪ Edges
23     endwhile;
24     E* := E'
25   endif
26 endfor
```

FIGURE 2. Computation of $unf(M^*)$ for a GCPS of size ω.

Constructing a new version of G^*, i.e., executing the first or the second part of an iteration, one virtually traverses the rooted tours of M^*, but actually their representations in the old version of G^*. One, hence, reads the old E^*, whereas the generated members of the new E^* are, until line 14 or 24 is reached, temporarily collected in E'. The old V^* is of interest only in the beginning (line 3 or 17), when for every entry or exit vertex, respectively, of the old G^*, one generates the corresponding vertices of the new G^*, i.e., initializes *Open*, the current set of those new vertices from which further expansion of the new G^* is yet to be considered. After the initialization, V^* is, like E', emptied (line 4 or 17) and ready to accept newly generated vertices. Note that in the first iteration part, there are segments specific to the first iteration (lines 3,7,8). This is because in that iteration, one has to read from M^*, whose format is different from that of the later versions of G^*.

In each expansion step (lines 6-12 or 19-22), one

(i) moves a vertex from *Open* to the new G^* (line 6 or 19),
(ii) finds (line 8, 10 or 21) in the old G^* the corresponding vertex and its outgoing or incoming edges, respectively,
(iii) for each of the edges, generates (line 7, 9 or 20) a copy with its final or initial vertex, respectively, additionally annotated with the information on which of the currently considered predicates are satisfied on the corresponding initially or finally rooted walk in M^*, respectively, collecting the copies in *Edges*, and
(iv) puts the new edges into the new G^* and adds those of their final or initial vertices, respectively, which require further consideration to *Open* (line 12 or 22).

For illustration, consider again the tour unfolding in Table 1. In the initial version of G^*, i.e., in the FSM from Figure 1, vertices in $\{v_1, v_6, v_7, v_{10}\}$ denote the same vertex, as they denote the same state, and so do vertices in $\{v_2, v_5, v_9\}$ and vertices in $\{v_3, v_4, v_8\}$. After adding annotations for p_s in the first part of the first iteration, the equivalence classes are $\{v_1\}$, $\{v_6, v_7, v_{10}\}$, $\{v_2\}$, $\{v_5, v_9\}$, $\{v_3, v_4\}$ and $\{v_8\}$. After adding annotations for p'_s in the second part of the first iteration, only v_5 and v_9 are still in the same class, but only until after the first part of the second iteration, in which annotations for $p_{s,-1}$ and $p'_{s,-1}$ are added. Annotations for $p_{s,+1}$ and $p'_{s,+1}$ are added in the second part of the second iteration.

If $unf(M^*)$ is too large, one constructs for G^* an approximation of it in which only the rooted tours from M^* corresponding to the pre-selected candidate tests are represented and/or, if Restrictions 7, 12 and 13 are

satisfied, underestimates predicate satisfaction when a β_i or a φ_i is called.

7. GENERALIZATION TO NONDETERMINISTIC MACHINES

7.1. Nondeterministic FSMs and Their Implementations

From now on, we no longer assume that M or its implementations, in Φ_M or not, are deterministic. Still, we continue assuming that the system under test (SUT) is always ready for any member of the input alphabet of M and that it is considered *faulty* if among the input sequences to which M is able to react, there is also one to which the SUT might respond with a sequence of outputs with which M never could.

For any unforbidden walk $\bar{\tau}$ in M, one needs an estimation $prob(\bar{\tau})$ of its *probability*, i.e., of the probability that after a $\bar{z}$ which might have led M to $init(\bar{\tau})$ has been observed on the tested M', the reaction of M' to an additional $in(\bar{\tau})$, if not a incorrect, will be a $\bar{\tau}'$ corresponding to $\bar{\tau}$.

For a deterministic M, a $prob(\bar{\tau})$ is always 1. For a nondeterministic M, one typically expects only implementations with some specific properties, for example, only such which have not more than a certain number of states and which are deterministic (e.g. [16, 17, 18]) or whose responses at least exhibit a certain degree of fairness (e.g. [3, 19]). Such additional assumptions might imply that for some walks $\bar{\tau}$ in M, $prob(\bar{\tau}) = 0$.

Additional assumptions on the expected implementations represent specific knowledge on the tested M', and even more such knowledge is collected while the testing proceeds. One should, hence, favour testing strategies facilitating exploitation of specific knowledge, particularly for the latter phases of adaptive testing.

The above discussion leads us to the observation that the role of M in test construction is just auxiliary: Together with the associated walk probabilities and with T', the set of the forbidden transitions, it acts as a more or less precise collective model for Ψ, the adopted set of the expected SUTs (a superset of Φ_M, which should better be called just Φ), for the frequency of its members and for the frequency of their individual responses to individual input sequences, with an assumption that the correctness of a response coincides with its representation in the model and that the input sequences qualifying as complete tests are only those which in the model correspond to a DRT. It is appropriate to say that conception of such a model is already the first step of test construction, although we do not discuss the step in detail, assuming that the necessary algorithm is available as a part of the adopted testing strategy.

7.2. Functions for Predicting the Costs and the Benefits of the Actualized Responses

If M becomes nondeterministic, testing strategies get two additional parameters: the adopted *cost-prediction function* CP and the adopted *witnessing-prediction function* WP. For a specific test applied in a specific context, the two functions operate on the set of all the specified responses, where each of the responses is represented as a walk in (the adopted approximation of) $unf(M^*)$.

For a walk set W, $CP(W)$ predicts what the cost of the actualized member of W will be. $CP(W)$ should be a value between (inclusively) $min_{\bar{e}\in W} cost(\bar{e})$ and $max_{\bar{e}\in W} cost(\bar{e})$. Some typical choices for CP would be the worst-case cost, the best-case cost and the probability-weighted mean cost, where $prob(\bar{e})$ for an $\bar{e}$ in W is $prob(\bar{\tau})$ of the corresponding walk $\bar{\tau}$ in M.

For a walk set W, $WP(W)$ predicts which will be the predicates from $PT(M^*)$ for whose satisfaction the actualized member of W will provide witness. $WP(W)$ should be a set between (inclusively) $\cap_{\bar{e}\in W} PT(\bar{e})$ and $\cup_{\bar{e}\in W} PT(\bar{e})$. Again, one is free to decide how optimistic such a prediction should be.

How optimistic one wants to be depends on the specific situation. For example, if it has been decided that the particular round of testing would be the last one (in non-adaptive testing, this is always the case), WP would typically predict only the witnessing secured in every possible case, for this is the only witnessing one can rely on. Likewise, there is often an upper limit on the allowed test cost, implying that CP might also be required to be maximally pessimistic. Hence, maximum pessimism will be the default choice for both functions. However, particularly during the early rounds of adaptive testing, some optimism is often appropriate, for example, if there is a test goal satisfiable in the average, but not in every case.

7.3. Adaptive and Non-adaptive Testing

Adaptive testing proceeds in consecutive rounds, with non-adaptive testing denoting the case where there is deliberately a single round. For the i-th round, let $\bar{z}^\circ_{i-1} = \bar{x}^\circ_{i-1}/\bar{y}^\circ_{i-1}$ denote the I/O sequence observed in the past. For $i > 1$, the round starts by deleting from Ψ and Φ all FSMs possessing no initially rooted walk $\bar{\tau}$ with $lab(\bar{\tau}) = \bar{z}^\circ_{i-1}$, plus all the other FSMs whose believed probability of being the SUT has become sufficiently low (if one is just applying a pre-constructed set of tests, the deletion corresponds to deciding that for some of the tests, further application to the SUT would make no sense [20]). For the updated Ψ and Φ, one then selects a testing strategy, for the strategy is not necessarily the same in every round. For example, in the method of [18], the non-terminal rounds are intended for reducing Ψ to a deterministic M and its faulty implementations from the initial Φ whose members are also all assumed to be deterministic, while the last

round is supposed to use any strategy for checking the SUT against M.

The next step of the i-th round is for any i to construct a collective model of the expected SUTs (see Section 7.1). Then one constructs an optimal input sequence $\bar{x}_i^\dagger$ for satisfying the remaining test goals or sometimes just for securing that the SUT, provided that it is M itself, returns to its root, because if $i > 1$, M might have an initially rooted walk $\bar{\tau}$ with $lab(\bar{\tau}) = \bar{z}_{i-1}^\circ$ for which $fin(\bar{\tau})$ is not s_0. Actually, it suffices to construct a non-empty prefix $\bar{x}_i^*$ of $\bar{x}_i^\dagger$, e.g., just its first step, unless it has already been decided that the round should be the last, so that $\bar{x}_i^* = \bar{x}_i^\dagger$ is required. If construction of $\bar{x}_i^*$ fails, one might want to resort to a different testing strategy or just establish that further testing would make no sense.

Application of $\bar{x}_i^*$ must also be considered a part of the i-th round (if one is constructing a complete test procedure, this means that all the possible responses to $\bar{x}_i^*$ must be considered one by one). However, $\bar{x}_i^*$ sometimes does not have to be applied in its entirety. In particular, we assume that the testing process never extends beyond the first faulty response. If the round is not deliberately the last one, there are also two other legitimate reasons for premature termination of the round: If the observed responses start indicating that the predictions of the adopted functions CP and WP on the costs and benefits of applying $\bar{x}_i^*$ were unacceptably inaccurate, the round should terminate immediately, so that one can switch to a more appropriate testing strategy, for example, to less ambitious test goals, if the test is getting unexpectedly expensive, or to more ambitious goals, if the SUT has taken a direction unexpectedly convenient for the testing process. The precise meaning of "unacceptably inaccurate" in the previous sentence is a part of the adopted testing strategy, for it depends on how much one is willing to invest in test optimization.

It remains to discuss how to construct $\bar{x}_i^*$. In the following, we consider only the most demanding case, where $\bar{x}_i^*$ is the entire $\bar{x}_i^\dagger$.

7.4. Direct Test Construction

If M is nondeterministic, additional care is necessary already in the construction of M^*: If a transition $(s, s', x/y, \bar{c})$ from M^+ is for some reason not copied to M^*, this means that it is not safe to apply x in the state s of M^*. Hence, copying of any other transition of the form $(s, \_, x/\_, \_)$ is also forbidden. The probability of each individual walk in M^* is as for the corresponding walk in M.

Assuming the general case of the constructed test $\bar{x}^*$ required to be an $\bar{x}_{i-1}^\circ \cdot \bar{x}_i^\dagger$ for an i-th round of testing, with $\bar{x}_{i-1}^\circ$ the already applied part of the test, it is important to represent in G^* at least one DRT from M^* whose label has prefix $\bar{z}_{i-1}^\circ$, the I/O sequence observed as the response to $\bar{x}_{i-1}^\circ$. If such a DRT $\bar{\tau}$ is represented in G^*, so must be every other such DRT $\bar{\tau}'$ with $in(\bar{\tau}') = in(\bar{\tau})$.

Once G^* is enhanced into G, a possible next step is to look for $\bar{x}^*$ directly on G, trying to find an optimal non-empty set W^* of rooted tours whose members for an input sequence starting with $\bar{x}_{i-1}^\circ$ through one-to-one correspondence identify in M^* every corresponding DRT whose label starts with $\bar{z}_{i-1}^\circ$, where the estimated cost and witnessing power of the considered test are $CP(W^*)$ and $WP(W^*)$, respectively.

As the problem of finding in G such an optimal W^* can be considered a generic graph-theoretic problem, we do not discuss it in the paper. Instead, we in Sections 7.5 and 7.6 suggest how to, at least approximately, reduce TT construction to an instance of the more usual G^{2}RPP, as we have done for deterministic M.

For illustration, we construct a test for the M in Figure 1. In the figure, the nondeterministic transitions are, like the corresponding edges in Figures 3 and 8, marked with a triangle arrowhead. For the rest of the paper, we assume that all transitions of the FSM are unforbidden and of cost 1, that the testing strategy is to traverse every transition and check its final state individually against every other state, with an SRS of any kind, and that there are no additional cost criteria.

Formalizing the strategy, we define that $PT(M^*)$ for every transition $\tau = (s, s', z, \bar{c})$ in T^* and state $s'' \in S \setminus \{s'\}$ comprises a predicate $p_{\tau,s''}(\bar{\tau})$ defined as
$\exists \bar{\tau}', \bar{\tau}''.((\bar{\tau} = \bar{\tau}' \cdot \tau \cdot \bar{\tau}'') \wedge p'_{s''}(\bar{\tau}''))$,
where predicates p'_s in $PF(M^*)$ are as in Section 6.1, while $PI(M^*)$ is empty [1].

With no additional cost criteria, $M^* = M$. The G with $G^* = unf(M^*)$ is represented in Figure 3, in which edge labels identify the corresponding transitions in M and every vertex $(s_i, \emptyset, PF)$ is encoded as $s_i^{\{j|p'_{s_j} \in PF\}}$. For example, name $s_1^{\{2\}}$ tells that the vertex corresponds to the state s_1 and to the start of a sequence distinguishing the state from s_2. One would, hence, want to traverse edge e_5 to check transition τ_1, e_7 for τ_2, e_6 and e_{11} for τ_3, e_9 for τ_4, e_{10} for τ_5, e_{15} for τ_6, e_{17} for τ_7, and e_8 and e_{12} for τ_8. In other words, the ideal is to traverse all the edges which are drawn bold. However, in the vertex $s_0^{\{1,2\}}$, it is impossible to force M to take from s_0 a transition corresponding to a specific outgoing bold edge of the vertex. There is, hence, the problem of non-controllability.

The extent of the problem is more clearly evident from the alternative representation of $unf(M^*)$ given in Figure 4. In the figure, every edge e from $unf(M^*)$ is represented as a sequence of an input edge and an output edge, the latter annotated with the identifier of e from Figure 3. We see that it is possible to control whether an a applied to s_0 will activate an edge in the set $\{e_6, e_9, e_{10}\}$ or an edge in the set $\{e_{11}, e_{13}, e_{14}\}$ (that depends on whether the next input will be b or a, respectively), while the choice of a

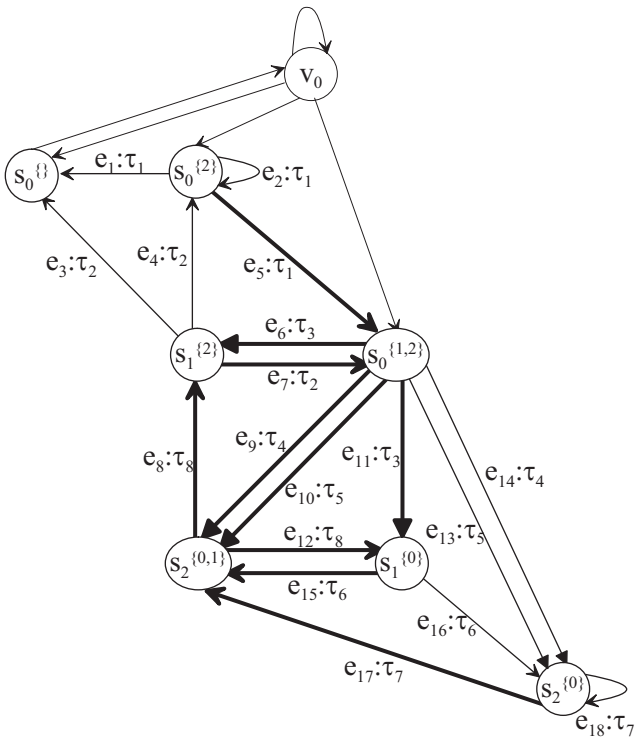

FIGURE 3. The $unf(M^*)$-based G constructed for the M in Figure 1.

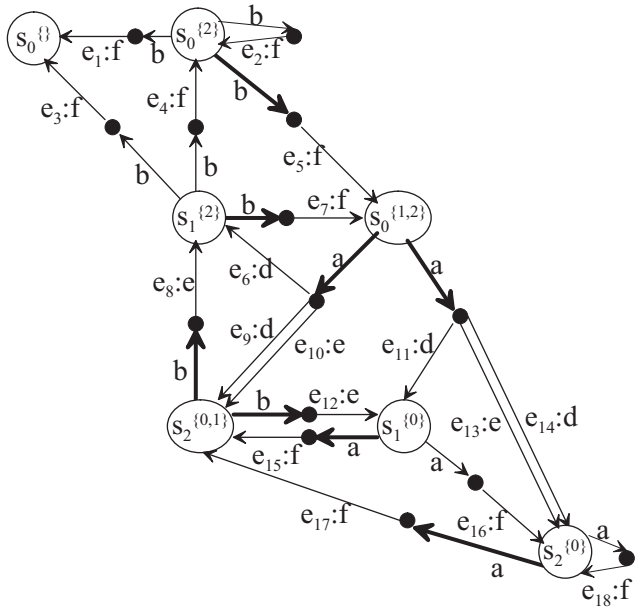

FIGURE 4. An alternative representation of the $unf(M^*)$ from Figure 3.

specific edge within such a set, e.g., of the desired edges e_6, e_9, e_{10} and e_{11}, is under the control of the SUT. Remember, however, that $PT(M^*)$ by default also comprises abstractions of its explicitly given members. In the particular case, it pays to apply in s_0 both an "$a, a, \ldots$" and an "$a, b, \ldots$", to collect evidence for $(p_{\tau_3,s_0} \vee p_{\tau_4,s_0} \vee p_{\tau_5,s_0})$ and $(p_{\tau_3,s_2} \vee (p_{\tau_4,s_0} \wedge p_{\tau_4,s_1}) \vee (p_{\tau_5,s_0} \wedge p_{\tau_5,s_1}))$, respectively. The target inputs are, hence, all those which Figure 3 shows in bold. An optimal test would be "$b, a, a, a, b, a, b, b, a, b, b$", in the G in Figure 3 corresponding to an optimal set of as many as 9 rooted tours.

7.5. WSG-Based Construction of Non-adaptive Tests

7.5.1. Constructing the Initial Version of the Central Subgraph of the Problem Model

To reduce the TT construction problem for a nondeterministic M to an instance of G^{2}RPP, one replaces the central subgraph G^* of G with a WSG $Q^\circ = (V^\circ, E^\circ)$ representing those walks from G^* which start in an $(s_0, PI(\varepsilon), PF)$ and end in an $(s_0, PI, PF(\varepsilon))$ in *sheaves of all the test runs possible for a specific input sequence*. One of the conveniences of the approach is that it can be easily generalized to test generation for EFSMs, because they are also WSGs (remember Section 3.5).

For a G^* precisely representing $unf(\bar{\tau})$ for every DRT $\bar{\tau}$ in M^*, the assumed first version $Q_1^\circ = (V_1^\circ, E_1^\circ)$ of Q° is $dunf(M^*)$ (see Section 3.9), for which a construction algorithm is given in Figure 5. If $dunf(M^*)$ is too large, one constructs for Q_1° an approximation of it in which only the DRTs from M^* corresponding to the pre-selected candidate tests are represented and/or, if Restrictions 7, 12 and 13 are satisfied, underestimates predicate satisfaction when a β_i or a φ_i is called.

The algorithm in Figure 5 is essentially that from Figure 2 (see Section 6.3), though with some modifications:

(i) The digraph constructed is not $G^* = (V^*, E^*)$, but $Q_1^\circ = (V_1^\circ, E_1^\circ)$.

(ii) In the first iteration, the body of the second part (lines 25-37) is also mandatory, to implement an early focusing on those rooted tours in M^* which are in $Trs(M^*)$, because for a nondeterministic M, there might exist also some which are not.

(iii) Initializing *Open* (lines 3-6 and 25-26), one takes every vertex representing a *set* of those vertices which the algorithm from Figure 2 would initially put on *Open*.

(iv) When retrieving copies of edges from M^* for further annotation, one looks in the *labels* of edges from the old Q_1° (line 18 or 34). The candidates for the initial or the final vertex, respectively, of a generated additionally annotated copy are all the members of the considered vertex from *Open* (line 13, 17 or 33), because in the case that the vertex is not a singleton set, this indicates that, because of M^* being nondeterministic, the corresponding state of M^* and the properties of the corresponding context might be any of the listed ones.

(v) In every expansion step of the first traversal, every considered input x must be checked for being

1 **for** $i := 1$ **to** ω **do**
2 **if** $(i = 1) \vee (PI_i(M^*) \setminus PI_{i-1}(M^*) \neq \emptyset)$ **then**
3 **if** $i = 1$ **then** $Open := \{\{(s_0, PI_1(\varepsilon), \emptyset)\}\}$
4 **else** $Open := \{\{(s_0, PI_i(\varepsilon), PF) | (s_0, PI_{i-1}(\varepsilon), PF) \in v\}$
5 $|(v \in V_1^\circ) \wedge \neg\exists(s, PI, \_) \in v.((s \neq s_0) \vee (PI \neq PI_{i-1}(\varepsilon)))\}$
6 **endif**;
7 $V_1^\circ := \emptyset; E' := \emptyset;$
8 **while** $Open \neq \emptyset$ **do**
9 Move a v from $Open$ to V_1°;
10 **if** $i = 1$ **then** $Edges := \{(v, \{v' | \exists(\_, v', \_, \_) \in Label\}, Label, CP(Label))$
11 $|\exists x.((\forall(s, \_, \_) \in v.\exists(s, \_, x/\_, \_) \in T^*) \wedge$
12 $(Label = \{((s, PI, \emptyset), (s', \beta_1((s, s', x/y, \bar{c}), PI, \emptyset, \emptyset, \emptyset), \emptyset), x/y, \bar{c})$
13 $|((s, PI, \emptyset) \in v) \wedge ((s, s', x/y, \bar{c}) \in T^*)\}))\}$
14 **else** $Edges := \{(v, \{v' | \exists(\_, v', \_, \_) \in Label\}, Label, CP(Label))$
15 $|\exists e \in E_1^\circ.((init(e) = \{(s, PI \cap PI_{i-1}(M^*), PF) | (s, PI, PF) \in v\}) \wedge$
16 $(Label = \{((s, PI, PF), (s', PI' \cup \beta_i((s, s', x/y, \bar{c}), PI, PF, PI', PF'), PF'), x/y, \bar{c})$
17 $|((s, PI, PF) \in v) \wedge$
18 $(((s, PI \cap PI_{i-1}(M^*), PF), (s', PI', PF'), x/y, \bar{c}) \in lab(e))\}))\}$
19 **endif**;
20 $Open := Open \cup (\{v' | \exists(\_, v', \_, \_) \in Edges\} \setminus V_1^\circ); E' := E' \cup Edges$
21 **endwhile**
22 $E_1^\circ := E'$
23 **endif**;
24 **if** $(i = 1) \vee (PF_i(M^*) \setminus PF_{i-1}(M^*) \neq \emptyset)$ **then**
25 $Open := \{\{(s_0, PI, PF_i(\varepsilon)) | (s_0, PI, PF_{i-1}(\varepsilon)) \in v'\}$
26 $|(v' \in V_1^\circ) \wedge \neg\exists(s, \_, PF) \in v'.((s \neq s_0) \vee (PF \neq PF_{i-1}(\varepsilon)))\};$
27 $V_1^\circ := \emptyset; E' := \emptyset;$
28 **while** $Open \neq \emptyset$ **do**
29 Move a v' from $Open$ to V_1°;
30 $Edges := \{(\{v | \exists(v, \_, \_, \_) \in Label\}, v', Label, CP(Label))$
31 $|\exists e \in E_1^\circ.((fin(e) = \{(s', PI', PF' \cap PF_{i-1}(M^*)) | (s', PI', PF') \in v'\}) \wedge$
32 $(Label = \{((s, PI, PF \cup \varphi_i((s, s', x/y, \bar{c}), PI, PF, PI', PF')), (s', PI', PF'), x/y, \bar{c})$
33 $|((s', PI', PF') \in v') \wedge$
34 $(((s, PI, PF), (s', PI', PF' \cap PF_{i-1}(M^*)), x/y, \bar{c}) \in lab(e))\}))\};$
35 $Open := Open \cup (\{v | \exists(v, \_, \_, \_) \in Edges\} \setminus V_1^\circ); E' := E' \cup Edges$
36 **endwhile**;
37 $E_1^\circ := E'$
38 **endif**
39 **endfor**

FIGURE 5. Computation of $dunf(M^*)$ for a GCPS of size ω.

applicable in every state of M^* corresponding to the considered vertex from *Open* (line 11). For every legal x, the generated additionally annotated copies of the corresponding edges from M^* are packed into the label of a new edge for Q_1° (lines 12-13). When computing *Edges* in the later traversals (lines 14-18 or 30-34), annotated edge copies are already known to be legal and properly sheaved, implying that the only task is to generate additional annotations.

For the example TT construction problem from Section 7.4, $dunf(M^*)$ is given in Figure 6, as the subgraph consisting of the edges which are labelled. In the subgraph, each vertex or edge label is a set of vertices or edges, respectively, from the corresponding $unf(M^*)$ subgraph of the graph in Figure 3. In $dunf(M^*)$, the non-controllability problem described in Section 7.4 reflects in the fact that the problematic edges e_6, e_9, e_{10} and e_{11} from $unf(M^*)$ occur exclusively in edge labels which are not singleton sets, namely in the labels of the edges e'_{16}, e'_{19}, e'_{20} and e'_{22}.

7.5.2. The Rest of the Procedure

(The adopted approximation of) $dunf(M^*)$ is typically not the final version of Q°, for one typically has to take some additional *measures for circumventing the problem of the non-compositionality of cost and witnessing predictions*: If Q° remains Q_1°, the cost which the G^{2}RPP solver activated on the resulting G believes for a

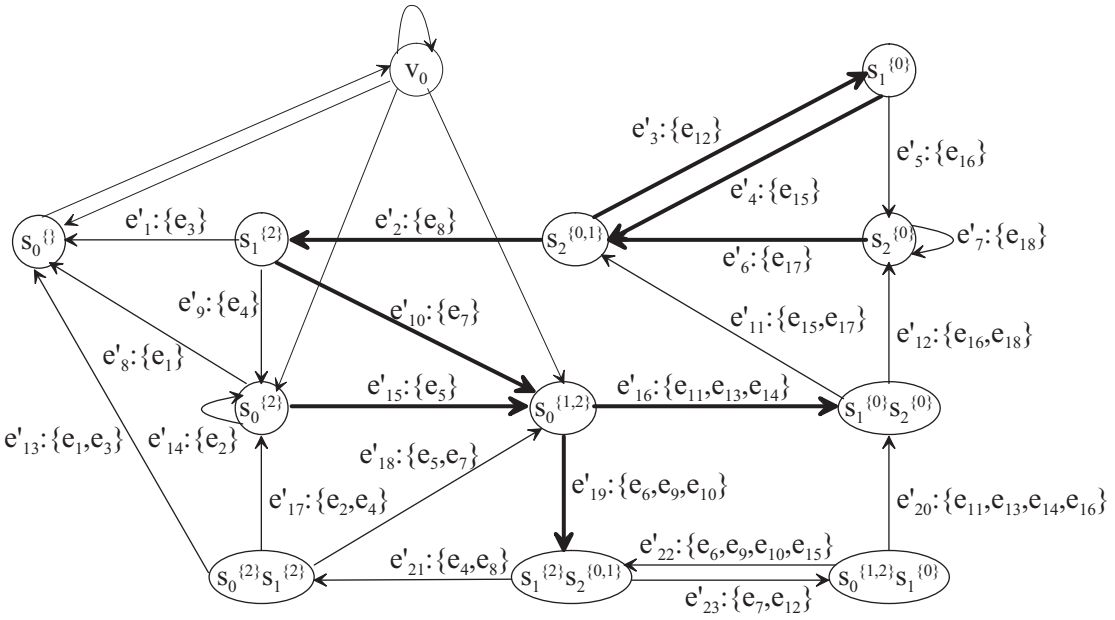

FIGURE 6. The $dunf(M^*)$-based G constructed for the M in Figure 1.

walk $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, E_i, CP(E_i))$ for $1 \leq i \leq k$, in Q° is $\sum_{1 \leq i \leq k} CP(E_i)$, which is often different from the desired estimation $CP(SH(\bar{e}))$. Likewise, $PT(\bar{e})$, the witnessing capability of $\bar{e}$ believed during the tour construction, is $\cup_{1 \leq i \leq k} WP(E_i)$, which is often different from the desired estimation $WP(SH(\bar{e}))$.

For illustration, take some edges e_1 to e_5 of cost 1, 2, 3, 4 and 2, respectively, and walks "e_1, e_4", "e_2, e_4" and "e_3, e_5", of cost 5, 6 and 5, respectively. Assuming that $init(e_4) \neq init(e_5)$, the walks can be represented in a sheaf, by a walk whose two edges represent sets $\{e_1, e_2, e_3\}$ and $\{e_4, e_5\}$, respectively. Estimation of the best case cost, of the worst case cost and of the average cost are all problematic:

$min\{1,2,3\} + min\{4,2\} = 1 + 2 = 3$
$min\{5,6,5\} = 5$
$max\{1,2,3\} + max\{4,2\} = 3 + 4 = 7$
$max\{5,6,5\} = 6$
$mean\{1,2,3\} + mean\{4,2\} = 2 + 3 = 5$
$mean\{5,6,5\} = 5.\dot{3}$

For a walk $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, E_i, CP(E_i))$ for $1 \leq i \leq k$, in Q°, the problem can be solved by introducing a *by-pass edge* $e_{\bar{e}} = (v_1, v_{k+1}, SH(\bar{e}), CP(SH(\bar{e})))$ with $PT(e_{\bar{e}}) = WP(SH(\bar{e}))$, where any edge of $\bar{e}$ which consequently becomes irrelevant for G may be deleted. Hence, in the next step, one enhances Q° with as many by-pass edges as desirable and feasible. The shorter the sequences of consecutive nondeterministic vertices in the walks of M^* are, the more likely it will be possible to introduce all the useful by-pass edges. If the adopted CP and WP weight the contribution of each individual walk with the walk probability, the step serves also for *neutralizing the presence of those walks which are believed to be impossible*.

For illustration, let us return to the $dunf(M^*)$ in Figure 6. For the explicitly defined predicates in $PT(M^*)$ and the default pessimistic WP, the problematic edges e'_{16}, e'_{19}, e'_{20} and e'_{22} all have zero witnessing capability and it does not seem that this could be improved by combining the edges with their neighbours.

As an example where a by-pass edge can help, take the M in Figure 7(i), with all transitions unforbidden and of the cost indicated in the figure. Suppose that $M^* = M$ and that $PT(M^*)$ comprises just one predicate, a $p(\bar{\tau})$ true if $\bar{\tau}$ comprises a τ with $out(\tau) = d$. $PI(M^*)$ and $PF(M^*)$ are, hence, empty. In Figure 7(ii), the subgraph consisting of the labelled edges is $dunf(M^*)$, where for every edge, we indicate only its input and its expected cost. As only the b edge is a predictable witness for p, setting Q° to $dunf(M^*)$ is inappropriate, as one then fails to observe that "a, a", and not "b", is the cheapest satisfactory test. As illustrated in Figure 7(iii), the problem can be solved by a by-pass edge revealing that of two consecutive a steps, there is always one witnessing for p and one whose cost is just 1.

As we have just seen, for every edge $e = (v, v', l, CP(l))$ in Q°, $in(l)$ can be used as a *shorthand* for l, because the latter is the set of all $((s, PI, PF), (s', PI', PF'), in(l)/\bar{y}, \bar{c})$ with (s, PI, PF) in v, (s', PI', PF') in v', and $(s, s', in(l)/\bar{y}, \bar{c})$ corresponding to a walk in M^* for which the end-point annotations (PI, PF) and (PI', PF') are under the adopted predicate-satisfaction underestimation policy consistent

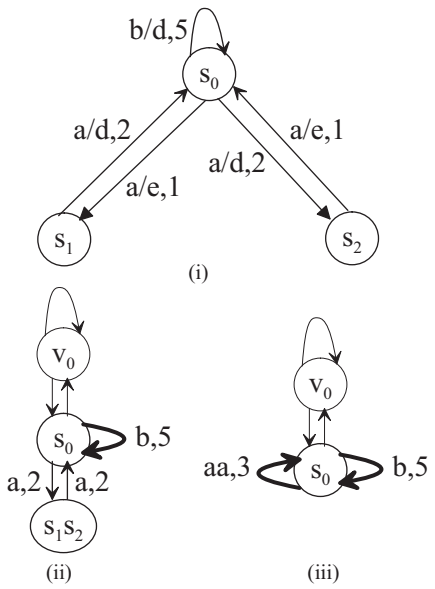

FIGURE 7. An example illustrating the use of by-pass edges: (i) the M, (ii) the $dunf(M^*)$-based first version of G and (iii) the final version of G. The bold edges are those identified as witnesses for the only member of $PT(M^*)$.

with respect to the functions β_i and φ_i. Moreover, as $cost(e)$ and $PT(e)$ are also derived properties, $(v, v', in(l))$ within the given context qualifies as a shorthand for e, just as if M was deterministic and G was $unf(M^*)$-based. Hence, TT construction can in principle continue in the usual way, though with the following adaptations:

When enhancing Q° into G, one introduces an auxiliary edge to or from a vertex v in V° exactly where such an edge is safe and desired for every vertex in the underlying G^* for which v comprises a copy: The auxiliary edges from v_0 to Q° end in those v in V° whose members are all of the form $(s_0, PI(\varepsilon), \_)$, while the auxiliary edges from Q° to v_0 start in those v in V° whose members are all of the form $(s_0, \_, PF(\varepsilon))$. If Restrictions 7, 12 and 13 are satisfied, it is also safe to introduce an auxiliary zero-cost edge between any pair of vertices v and v' in Q° for which every pair of an (s, PI, PF) in v and an (s', PI', PF') in v' satisfies $s = s' = s_0$, $PI' \subseteq PI$ and $PF \subseteq PF'$. Once a rooted tour in G and its projection $\bar{e}^\circ = e_1, \ldots, e_k$ onto the edges of the central subgraph are constructed, the corresponding test $\bar{x}^*$ is $in(lab(e_1)) \cdot \ldots \cdot in(lab(e_k))$.

For the example TT construction problem from Section 7.4, the $dunf(M^*)$-based G is the graph in Figure 6. There are many edges representing an individual target edge from $unf(M^*)$. Besides, it pays to traverse e'_{16} and e'_{19}, because they witness for the derived abstract predicates $(p_{\tau_3,s_0} \vee p_{\tau_4,s_0} \vee p_{\tau_5,s_0})$ and $(p_{\tau_3,s_2} \vee (p_{\tau_4,s_0} \wedge p_{\tau_4,s_1}) \vee (p_{\tau_5,s_0} \wedge p_{\tau_5,s_1}))$, respectively. The target edges in the G are, hence, all those drawn bold. An optimal $\bar{e}^\circ$ would be "$e'_{15}, e'_{16}, e'_{12}, e'_6, e'_3, e'_4, e'_2, e'_{10}, e'_{19}, e'_{21}, e'_{13}$", corresponding exactly to the test constructed in Section 7.4.

7.6. WSG-Based Construction of Test Continuations

7.6.1. Constructing the Initial Version of the Central Subgraph of the Problem Model

If the constructed test $\bar{x}^*$ is required to be an $\bar{x}^\circ_{i-1} \cdot \bar{x}^\dagger_i$ for a non-initial i-th round of adaptive testing, with $\bar{x}^\circ_{i-1}$ the already applied part of the test and the I/O sequence produced in response a $\bar{z}^\circ_{i-1}$, one in $PI(M^*)$ needs a special-purpose predicate $p^+(\bar{\tau})$ defined as

$\exists \bar{\tau}', \tau.(\bar{\tau} = \bar{\tau}' \cdot \tau)$,

whose satisfaction must never be underestimated. The idea is to secure that for any DRT $\bar{\tau}$ in M^*, $init(unf(\bar{\tau}))$ becomes the only vertex on $unf(\bar{\tau})$ of the form $(s_0, PI, \_)$ with p^+ not in PI and that only approximations of $unf(\bar{\tau})$ inheriting the property are employed. With this arrangement, no vertex of the constructed $dunf(M^*)$-based central subgraph Q° of G ever has both a member corresponding to the start of a DRT and a member corresponding to a non-initial vertex of a DRT. In other words, DRT starts are in Q° clearly represented as the set $Start(Q^\circ)$ of those vertices whose members are all of the form $(s_0, PI, \_)$ with p^+ not in PI, actually of the form $(s_0, PI(\varepsilon), \_)$, as it is assumed that no predicate from $PI(M^*)$ is ever underestimated on ε. By the definition of p^+, the vertices in $Start(Q^\circ)$ are without incoming edges.

When constructing Q°_1, the initial version of Q°, one must be aware that only DRTs with $\bar{z}^\circ_{i-1}$ a prefix of their label are of interest. At least one such DRT must be represented in Q°_1. Representation of DRTs which are not of the kind is irrelevant, implying that Q°_1 may as well be precisely $dunf(M^*)$.

For illustration, let us return to the example TT construction problem from Section 7.4. Now we want to test adaptively, sticking to the same strategy as before, but expecting only deterministic implementations of the M in Figure 1, with at most three states. As $\bar{x}^\dagger_1$, we adopt the test derived in Section 7.5.2, but try to further improve it after having applied its prefix $\bar{x}^\circ_1 = b, a$.

Suppose that the $\bar{y}^\circ_1$ which has been observed in response is "f, d". Hence, we in the second round of testing assume that M is as before, but without the transition τ_5, because the response indicates that τ_3 or τ_4 has been implemented instead. With M^* the reduced M and with $PI(M^*)$ enhanced with p^+ as its only member, $unf(M^*)$ is as in Figure 8. The corresponding $dunf(M^*)$, which we adopt as Q°_1, is given in Figure 9, as the subgraph comprising every vertex except v_0 and every edge which is labelled, but not bold. For edges e'_1 to e'_{23}, the name identifies the corresponding edge in Figure 6.

In both figures, triplets (s_i, PI, PF) are encoded as $s_i^{q,\{j|p'_{s_j} \in PF\}}$, where q is 1 or 0, indicating, respectively,

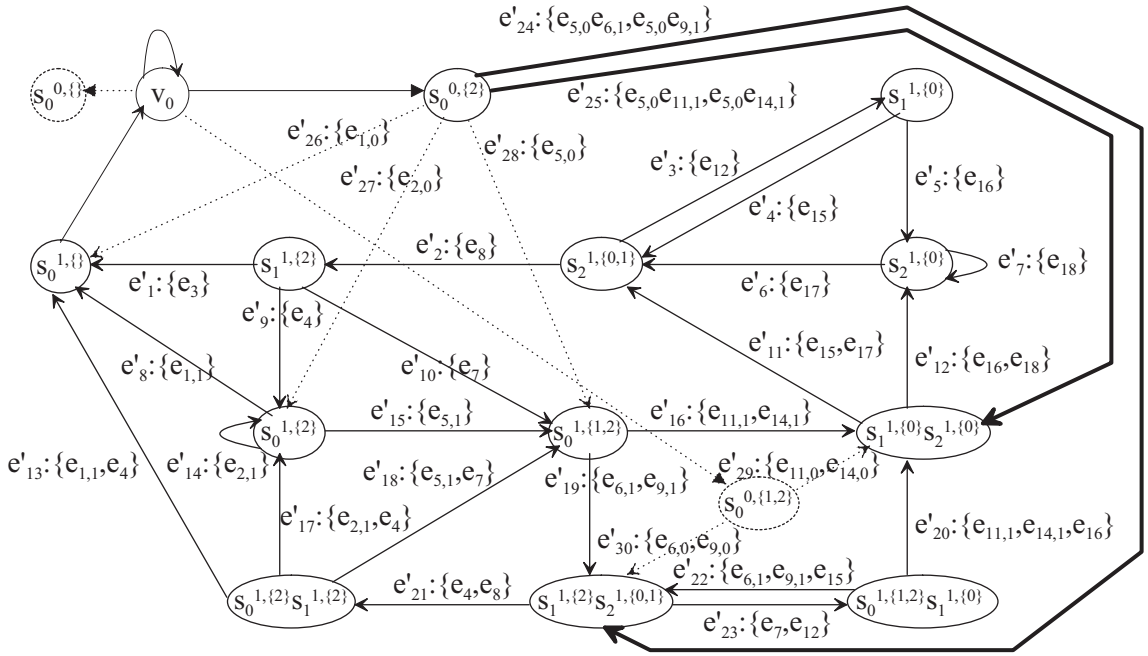

FIGURE 9. Combined representation of the Q_1° and the resulting G constructed for the example test continuation construction problem.

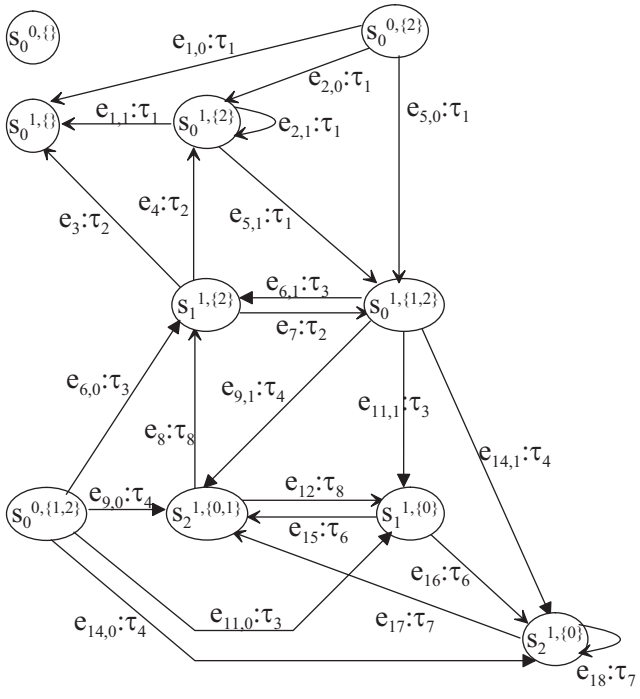

FIGURE 8. The $unf(M^*)$ from Figure 3 enhanced with explicit representation of test run starts, but without the walks irrelevant for the assumed non-initial round of adaptive testing.

that p^+ is or is not in PI. In the edge names in Figure 8, the first subscript identifies the corresponding edge in Figure 3, while the second subscript, if it exists, is identical to the first superscript in the name of the initial vertex of the edge. In the Q_1° subgraph of Figure 9, the members of $Start(Q^\circ)$ are the vertices comprising just $s_0^{0,\{\}}$, $s_0^{0,\{2\}}$ or $s_0^{0,\{1,2\}}$.

7.6.2. The Rest of the Procedure

To obtain Q_2°, the second approximation of Q°, one introduces for every walk $\bar{e}$ in Q_1° with $init(\bar{e})$ in $Start(Q^\circ)$ and with $in(SH(\bar{e})) = \bar{x}_{i-1}^\circ$ a by-pass edge $(init(\bar{e}), fin(\bar{e}), l, CP(l))$ with l the set of all $\bar{e}'$ in $SH(\bar{e})$ with $lab(\bar{e}') = \bar{z}_{i-1}^\circ$. The introduced by-pass edges form $Past_1(Q^\circ)$, the initial version of the set $Past(Q^\circ)$ of the edges individually modelling the past execution of $\bar{z}_{i-1}^\circ$. In Figure 9, such by-pass edges are e'_{24} and e'_{25}, for the walks "e'_{28}, e'_{19}" and "e'_{28}, e'_{16}", respectively. The two walks both represent a sheaf of the walks "τ_1, τ_3" and "τ_1, τ_4" from M^*, but in a different context, assuming that the next input is b or a, respectively.

To obtain Q_3°, the third approximation of Q°, one deletes from Q° every edge e qualifying neither as the first element of the projection $\bar{e}^\circ$ of the subsequently constructed tour $\bar{e}^*$ onto the edges of Q° (for that, it would have to be in $Past(Q^\circ)$) nor as a non-initial element of $\bar{e}^\circ$ (for that, $init(e)$ would have to be outside $Start(Q^\circ)$). In Figure 9, such edges are e'_{26} to e'_{30}.

To obtain Q_4°, the fourth approximation of Q°, one enhances Q° with as many additional by-pass edges $(v_1, v_{k+1}, SH(\bar{e}), CP(SH(\bar{e})))$ for walks $\bar{e} = e_1, \ldots e_k$, $e_i = (v_i, v_{i+1}, W_i, \bar{c}_i)$ for $1 \leq i \leq k$, as convenient. Any by-pass edge introduced for a walk comprising an edge from $Past(Q^\circ)$ also becomes a member of $Past(Q^\circ)$,

because the fact that vertices from $Start(Q^\circ)$ have no incoming edges implies that the edge from $Past(Q^\circ)$ is the first edge of the walk and, hence, the by-pass edge comprises the past. In Figure 9, there are no additional by-pass edges, implying that $Past(Q^\circ)$ comprises just e'_{24} and e'_{25}.

Q°_4 is then enhanced into the first approximation of G: One introduces a root v_0 and the usual auxiliary edges, except for those starting in $Start(Q^\circ)$, to indicate that $\bar{e}^*$ should not exit Q° immediately upon entering it. In Figure 9, the auxiliary edges are the unlabelled ones. As for the edge costs, one follows the usual policy for discouraging $\bar{e}^*$ from passing from v_0 to Q° multiple times. In Figure 9, the extremely expensive auxiliary edges are those with a triangle arrowhead. To finalize G, and thereby its central subgraph Q°, one then deletes all edges and vertices not lying on a rooted tour. In Figure 9, the additionally deleted elements are the dotted auxiliary edges and the dotted vertices.

The rooted tour $\bar{e}^*$ constructed on G is acceptable only if it enters $Start(Q^\circ)$ exactly once, i.e., if it comprises exactly one edge from $Past(Q^\circ)$, which is then interpreted as the first edge of the projection $\bar{e}^\circ$, an edge whose label comprises exclusively walks whose label starts with $\bar{z}^\circ_{i-1}$, as required. To force $\bar{e}^*$ to actually traverse an edge from $Past(Q^\circ)$ even if the only purpose of the test continuation being constructed is to return M^* to the root, we introduce into $PT(M^*)$ a special-purpose predicate $p^*(\bar{\tau})$ equivalent to $\bar{\tau} \neq \varepsilon$ and defined as

$\exists \bar{\tau}', \tau, \bar{\tau}''.(\bar{\tau} = \bar{\tau}' \cdot \tau \cdot \bar{\tau}'')$,

i.e., a predicate for which every edge in Q° is a witness.

In Figure 9, it, like in Figure 6, pays to traverse e'_{16} and e'_{19}, this time because they witness for predicates $(p_{\tau_3,s_0} \vee p_{\tau_4,s_0})$ and $(p_{\tau_3,s_2} \vee (p_{\tau_4,s_0} \wedge p_{\tau_4,s_1}))$, respectively. However, this evidence can be collected also by traversing e'_{24} or e'_{25}, respectively. As e'_{24} or e'_{25} can only be reached over a very expensive auxiliary edge, $\bar{e}^*$ will traverse just e'_{24} or just e'_{25}, and in addition rely on e'_{19} or e'_{16}, respectively. As the other edges to be traversed are e'_2, e'_3, e'_4, e'_6 and e'_{10}, one of the optimal $\bar{e}^\circ$ happens to be "$e'_{25}, e'_{12}, e'_6, e'_3, e'_4, e'_2, e'_{10}, e'_{19}, e'_{21}, e'_{13}$", corresponding to the originally planned test, where e'_{25} represents the already executed part. Obviously, at least for the adopted testing strategy, the knowledge that τ_5 has not been implemented is not a sufficient reason for changing the rest of the test.

8. GENERALIZATION TO EXTENDED FINITE STATE MACHINES

8.1. Introduction

Attempting EFSM-based testing, we in the following assume that the specification FSM $M = (S, s_0, T)$ for the particular testing round, with elements of S and T drawn from universes $\mathcal{S}$ and $\mathcal{T}$, respectively, is given by a rooted WSG $Q^\bullet = (V^\bullet, \{s_0\}, E^\bullet)$. This is in accordance with our view that an EFSM is just a WSG for an initially connected FSM (see Section 3.5). With this view, we, like [5], limit our attention to EFSMs without spontaneous transitions or quiescence, but compensate that by allowing nondeterminism in responses to external stimuli, as we have done for FSMs. For the purpose of testing, the approach is entirely adequate, for even if the SUT to some test produces no visible response, the tester will after some time conclude that no visible response is to be expected, i.e., virtually *observe* the non-response by observing that a timer has expired [21].

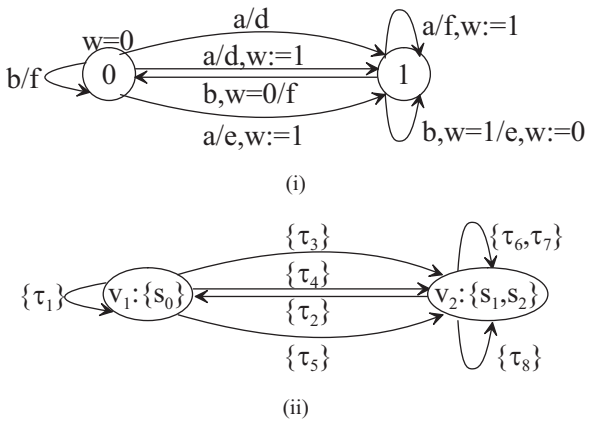

FIGURE 10. The $Q^\bullet$ of the running example: (i) in a typical notation, (ii) in the abstract notation.

By EFSM-based test construction we mean that a test for M is derived without first extracting M from $Q^\bullet$, where it is assumed that $Q^\bullet$ represents M in a manner sufficiently explicit that any property of M to which functions β_i, φ_i and θ_p of the adopted GCPS refer can be easily assessed directly on $Q^\bullet$. To satisfy the assumption, it might in some cases be more acceptable to adapt the GCPS than to further unfold $Q^\bullet$. For example, predicates p'_s from Section 6.1 originally refer to T, while on $Q^\bullet$, they could instead refer to the union of its edge labels, even if one would thereby undesirably suspect executability of some actually unexecutable transitions, for the union is often a true superset of T.

For illustration, let us return to the example TT construction problem from Section 7.4. For the M in Figure 1, avoiding explicit representation of the state space, of course, makes no sense and brings only unnecessary complications of the procedures and the generated models. Still, we will be able to show how a symmetry from $Q^\bullet$ induces a symmetry in the problem model G. For a realistically large M, such additional symmetries in G could be crucial for its tractability.

Let the states s_0, s_1 and s_2 of the adopted M be encoded as vectors $(0,0)$, $(1,0)$ and $(1,1)$, respectively, with the two binary vector components interpreted as the logical state of the system and the current value of an internal binary variable w, respectively. With this interpretation, the state universe $\mathcal{S}$ comprises

```
1  for i := 1 to ω do
2    if (i = 1) ∨ (PI_i(M*) \ PI_{i−1}(M*) ≠ ∅) then
3      if i = 1 then Open := {{({s_0}, PI_1(ε), ∅)}}
4      else Open := {{({s_0}, PI_i(ε), PF)|({s_0}, PI_{i−1}(ε), PF) ∈ v}
5                    |(v ∈ V_1^◇) ∧ ¬∃(S', PI, _) ∈ v.((S' ≠ {s_0}) ∨ (PI ≠ PI_{i−1}(ε)))}
6      endif;
7      V_1^◇ := ∅; E' := ∅;
8      while Open ≠ ∅ do
9        Move a v from Open to V_1^◇;
10       if i = 1 then Edges := {(v'', {v'|∃(_, v', _, _) ∈ Label}, Label, CP(Label))
11                              |(v'' ∈ S*) ∧ (sa(v'') = v)∧
12                              ∃x.((∀(s, _, _) ∈ v''.∃(sa(s), _, l, _) ∈ E*.((s, _, x/_, _) ∈ l))∧
13                                  (Label = {((s, PI, ∅), (s', β_1((s, s', x/y, c̄), PI, ∅, ∅, ∅), ∅), x/y, c̄)
14                                           |((s, PI, ∅) ∈ v'') ∧ ∃(sa(s), _, l, _) ∈ E*.((s, s', x/y, c̄) ∈ l)})}
15       else Edges := {(v'', {v'|∃(_, v', _, _) ∈ Label}, Label, CP(Label))
16                      |(v'' ∈ S*) ∧ (sa(v'') = v)∧
17                      ∃e ∈ E_1^◇, e' ∈ lab(e).
18                        ((init(e') = {(s, PI ∩ PI_{i−1}(M*), PF)|(s, PI, PF) ∈ v''})∧
19                         (Label = {((s, PI, PF), (s', PI' ∪ β_i((s, s', x/y, c̄), PI, PF, PI', PF'), PF'), x/y, c̄)
20                                  |(s, PI, PF) ∈ v'')∧
21                                  (((s, PI ∩ PI_{i−1}(M*), PF), (s', PI', PF'), x/y, c̄) ∈ lab(e'))}))}
22       endif; Open := Open ∪ ({sa(fin(e))|e ∈ Edges} \ V_1^◇);
23       E' := E' ∪ {(v, v', Label, CP+(Label))|(Label = {e|(e ∈ Edges) ∧ (sa(fin(e)) = v')}) ∧ (Label ≠ ∅)}
24     endwhile;
25     E_1^◇ := E'
26   endif;
27   if (i = 1) ∨ (PF_i(M*) \ PF_{i−1}(M*) ≠ ∅) then
28     Open := {{({s_0}, PI, PF_i(ε))|({s_0}, PI, PF_{i−1}(ε)) ∈ v'}
29              |(v' ∈ V_1^◇) ∧ ¬∃(S', _, PF) ∈ v'.((S' ≠ {s_0}) ∨ (PF ≠ PF_{i−1}(ε)))};
30     V_1^◇ := ∅; E' := ∅;
31     while Open ≠ ∅ do
32       Move a v' from Open to V_1^◇;
33       Edges := {({v|∃(v, _, _, _) ∈ Label}, v'', Label, CP(Label))
34                 |(v'' ∈ S*) ∧ (sa(v'') = v')∧
35                 ∃e ∈ E_1^◇, e' ∈ lab(e).
36                   ((fin(e) = {(s', PI', PF' ∩ PF_{i−1}(M*))|(s', PI', PF') ∈ v''})∧
37                    (Label = {((s, PI, PF ∪ φ_i((s, s', x/y, c̄), PI, PF, PI', PF')), (s', PI', PF'), x/y, c̄)
38                              |((s', PI', PF') ∈ v'')∧
39                              (((s, PI, PF), (s', PI', PF' ∩ PF_{i−1}(M*)), x/y, c̄) ∈ lab(e'))}))};
40       Open := Open ∪ ({sa(init(e))|e ∈ Edges)} \ V_1^◇);
41       E' := E' ∪ {(v, v', Label, CP+(Label))|(Label = {e|(e ∈ Edges) ∧ (sa(init(e)) = v)}) ∧ (Label ≠ ∅)}
42     endwhile;
43     E_1^◇ := E'
44   endif
45 endfor
```

FIGURE 11. Computation of $dunf^{\Box}(Q^{\star})$ for a GCPS of size ω.

one more state, an s_3 encoded as $(0, 1)$, whereas the assumed $Q^{\bullet}$ is as in Figure 10. In Figure 10(i), $Q^{\bullet}$ is represented in a typical notation, with vertices corresponding to (possibly additionally constrained) logical states and edges labelled with an input (possibly with a precondition), with the resulting output and the resulting variable update (if any). In Figure 10(ii), $Q^{\bullet}$ is represented in the adopted abstract notation.

8.2. EFSM-Based Construction of Non-adaptive Tests

The procedure is an adaptation of that from Section 7.5.

8.2.1. Constructing the Initial Version of the Central Subgraph of the Problem Model

Instead of M^+, one constructs its WSG $Q^+ = (V^+, \{s_0\}, E^+)$, an EFSM obtained from $Q^{\bullet}$ by introducing

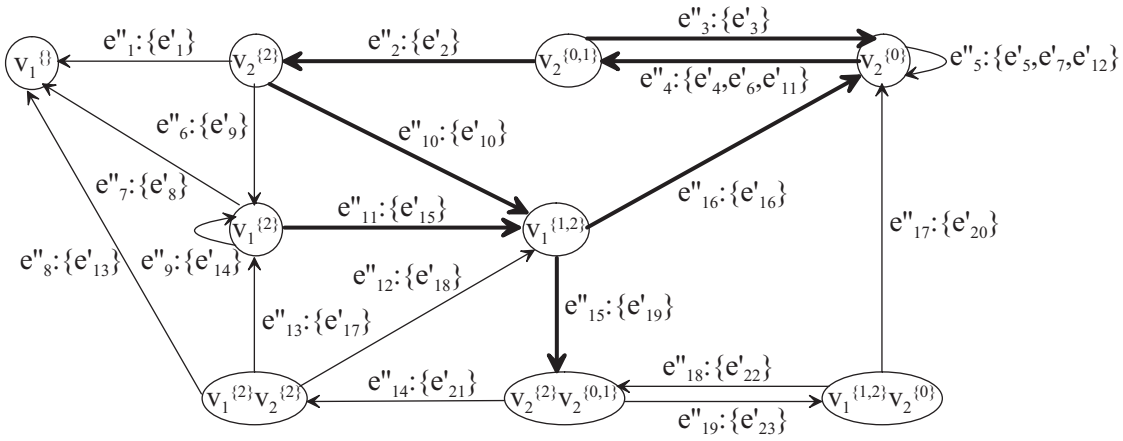

FIGURE 12. The $dunf^{\square}(Q^{\star})$ constructed for the $Q^{\bullet}$ in Figure 10.

the costs according to the additional cost criteria for every instance of a transition from T in the label of an edge in $E^{\bullet}$, thereby influencing also the cost vector of the edge. For our example, Q^{+} is simply $Q^{\bullet}$.

Instead of M^{*}, one constructs its WSG $Q^{\star} = (V^{\star}, \{s_0\}, E^{\star})$, an EFSM obtained from Q^{+} by

(i) deleting from its edge labels every τ corresponding to a transition in T' and every τ for which a τ' with $init(\tau) = init(\tau')$ and $in(\tau) = in(\tau')$ has already been deleted from an edge label and
(ii) deleting every edge which has an empty label and every edge and vertex which is no longer a part of a rooted tour.

For our example, $Q^{\star}$ is simply Q^{+}, i.e. $Q^{\bullet}$.

Instead of Q°, one constructs as the central subgraph of G a WSG $Q^{\diamond} = (V^{\diamond}, E^{\diamond})$ in which all the walks from Q° which represent a sheaf of walks from G^{*}, i.e., all the walks whose starting vertex contains exclusively members of the form $(s_0, PI(\varepsilon), \_)$ and ending vertex exclusively members of the form $(s_0, \_, PF(\varepsilon))$, are represented in sheaves, where the meta-sheaving imitates the sheaving of the corresponding walks from M^{*} in $Q^{\star}$. The idea is to make the central subgraph of G resemble $Q^{\star}$, so that it can be constructed directly from it.

As the first version $Q_1^{\diamond} = (V_1^{\diamond}, E_1^{\diamond})$ of $Q^{\diamond}$, one constructs a WSG for the walks of interest in Q_1°, the first version of Q°. For a G^{*} precisely representing $unf(\bar{\tau})$ for every DRT $\bar{\tau}$ in M^{*}, the assumed Q_1° is the $dunf^{\square}(Q^{\star})$ defined by the construction procedure in Figure 11. Ideally, $dunf^{\square}(Q^{\star})$ would be $dunf^{*}(Q^{\star})$ (see Section 3.10), but for easier construction, it is rather a more symmetric WSG equivalent to it (see the last paragraph of Section 3.4). If it is too large, one constructs for Q_1° an approximation of it in which only the DRTs from M^{*} corresponding to the pre-selected candidate tests are represented and/or, if Restrictions 7, 12 and 13 are satisfied, underestimates predicate satisfaction when a β_i or a φ_i is called.

For our example, $dunf^{\square}(Q^{\star})$ is given in Figure 12, in which edge labels refer to edges in Figure 6. The bold edges are those whose label comprises at least one edge from $dunf(M^{*})$ which, as known from Section 7.5.2, deserves traversal.

The algorithm in Figure 11 is essentially that from Figure 5 (see Section 7.5.1), though with some modifications:

(i) The digraph constructed is not $Q_1^{\circ} = (V_1^{\circ}, E_1^{\circ})$, but $Q_1^{\diamond} = (V_1^{\diamond}, E_1^{\diamond})$.
(ii) One refers to the universe $\mathcal{S}^{*}$ from which vertices of walks $dunf(\bar{\tau})$ are drawn, i.e., the universe of non-empty sets of those triplets (s, PI, PF) with s in $\mathcal{S}$, $PI \subseteq PI(M^{*})$ and $PF \subseteq PF(M^{*})$ which could not be ruled out as infeasible. For example, triplets with the predicate p_s' from Section 6.1 in PF are infeasible, because no $lab(\bar{\tau})$ can witnesses that $init(\bar{\tau})$ is not what it is. When constructing the $dunf^{\square}(Q^{\star})$ in Figure 12, we actually employed the knowledge. Consequently, the digraph happens to be precisely $dunf^{*}(Q^{\star})$.
(iii) Initializing *Open* (lines 3-6 and 28-29), one takes every vertex which is $sa(v)$ for a vertex v which the algorithm from Figure 5 would initially put on *Open*.
(iv) When retrieving copies of edges from M^{*} for further annotation, one looks in the *labels of the members of the labels* of edges from the old $Q_1^{\diamond}$ (line 21 or 39). The candidates for the initial or the final vertex, respectively, of a generated additionally annotated copy are all the members of those v'' from $\mathcal{S}^{*}$ for which $sa(v'')$ is the considered vertex from *Open* (line 14, 20 or 38), i.e., of all those v'' which the algorithm from Figure 5 would consider plus possibly of some whose consideration is irrelevant, because the vertex from *Open* neither has nor will ever obtain an incoming or an outgoing edge, respectively, with a label comprising an edge with the final or the initial vertex, respectively,

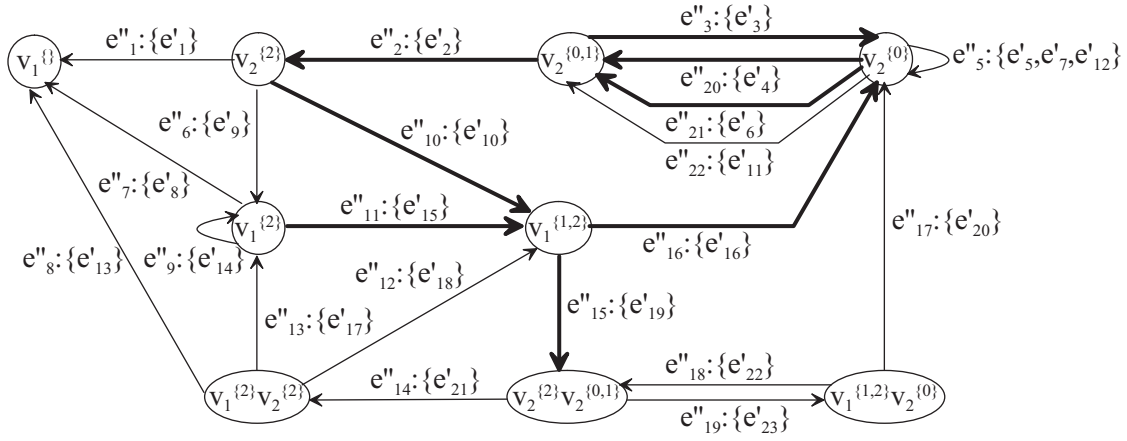

FIGURE 13. An intermediate form of the $Q^\diamond$ constructed for the $Q^\bullet$ in Figure 10.

$InEdgs := \{e|(e \in E^\diamond) \wedge (fin(e) = v)\}; OutEdgs := \{e|(e \in E^\diamond) \wedge (init(e) = v)\}; E^\diamond := E^\diamond \setminus (InEdgs \cup OutEdgs);$
$EdgSets := \{\{e'|(e' \in OutEdgs) \wedge \exists e''' \in lab(e').(fin(e'') = init(e'''))\}|\exists e \in InEdgs.(e'' \in lab(e))\};$
$V^\diamond := V^\diamond \setminus \{v\}$; **for** $\forall E' \in EdgsSets$ **do** Add to $V^\diamond$ a new vertex $v_{E'}$ **endfor**;
$E^\diamond := E^\diamond \cup \{(v_{E'}, fin(e), lab(e), cost(e))|(E' \in EdgSets) \wedge (e \in (E' \setminus InEdgs))\};$
$InEdgs := (InEdgs \setminus OutEdgs) \cup \{(v_{E'}, fin(e), lab(e), cost(e))|(E' \in EdgSets) \wedge (e \in (E' \cap InEdgs))\};$
$E^\diamond := E^\diamond \cup \{(init(e), v_{E'}, Label, CP^+(Label))$
$|(e \in InEdgs) \wedge (E' \in EdgSets) \wedge$
$(Label = \{e''|(e'' \in lab(e)) \wedge (\{e'|(e' \in OutEdgs) \wedge \exists e''' \in lab(e').(fin(e'') = init(e'''))\} = E')\}) \wedge$
$(Label \neq \emptyset)\}$

FIGURE 14. Splitting a problematic vertex v in $Q^\diamond$.

equal to v''. Such liberal policy of candidate selection has been adopted because it is easy to implement.

(v) Instead of, respectively, the final or the initial vertices v of the members of *Edges*, one moves to *Open* abstractions $sa(v)$ of those vertices (line 22 or 40).

(vi) When the members of *Edges* are moved to the new $Q_1^\diamond$ (line 23 or 41), they are sheaved according to the abstractions $sa(v)$ and $sa(v')$ of their initial and final vertices v and v', where the cost-prediction function employed for the meta-sheaving is a CP^+. For a walk set W, $CP^+(W)$ should be a value between (inclusively) $min_{\bar{e} \in W} cost(\bar{e})$ and $max_{\bar{e} \in W} cost(\bar{e})$, where the latter is also the recommended default.

(vii) $PT(e)$ for edges e in *Edges* remains $WP(lab(e))$. For edges e in $E^\diamond$, $PT(e)$ is a $WP^+(lab(e))$ with WP^+ the witnessing-prediction function employed for the meta-sheaving. For a walk set W, $WP^+(W)$ should be a value between (inclusively) $\cap_{\bar{e} \in W} PT(\bar{e})$ and $\cup_{\bar{e} \in W} PT(\bar{e})$, where the former is also the recommended default.

8.2.2. The Rest of the Procedure

To obtain the final version of $Q^\diamond$, one applies, wherever mandatory or promising, transformations of the following kinds: edge splitting, vertex splitting, walk by-passing, edge deletion and vertex deletion. In particular, one has to delete every edge and vertex which could not lie on a rooted tour of the subsequently constructed G.

Splitting of an edge $e = (v, v', l, CP^+(l))$ in the current version of $Q^\diamond$ into a pair of edges $e_1 = (v_1, v_1', l_1, CP^+(l_1))$ and $e_2 = (v_2, v_2', l_2, CP^+(l_2))$ with $\{l_1, l_2\}$ a partition of l facilitates more independent consideration of the elements of l_1 and the elements of l_2. This leads to a more individualized and, hence, more accurate estimation of their cost and witnessing power and facilitates their independent combination with other elements of the graph. Returning to our example and its $Q_1^\diamond$ in Figure 12, we split e_4'' into e_{20}'', e_{21}'' and e_{22}'', because it is desirable that the interesting members e_4' and e_6' of its label become isolated. The resulting version of $Q^\diamond$ is given in Figure 13.

Vertex splitting is intended exclusively for the mandatory elimination of walks $\bar{e}$ with $SH(\bar{e})$ empty. The proposed brute-force method for preventing such walks is elimination of every vertex v with such an incoming edge e and such an outgoing edge e' that for an e'' in $lab(e)$, $lab(e')$ comprises no e''' with $fin(e'') = init(e''')$. The vertex-splitting transformation from Figure 14, analogous to the first transformation for the elimination of conditional transitions given in [4], is the

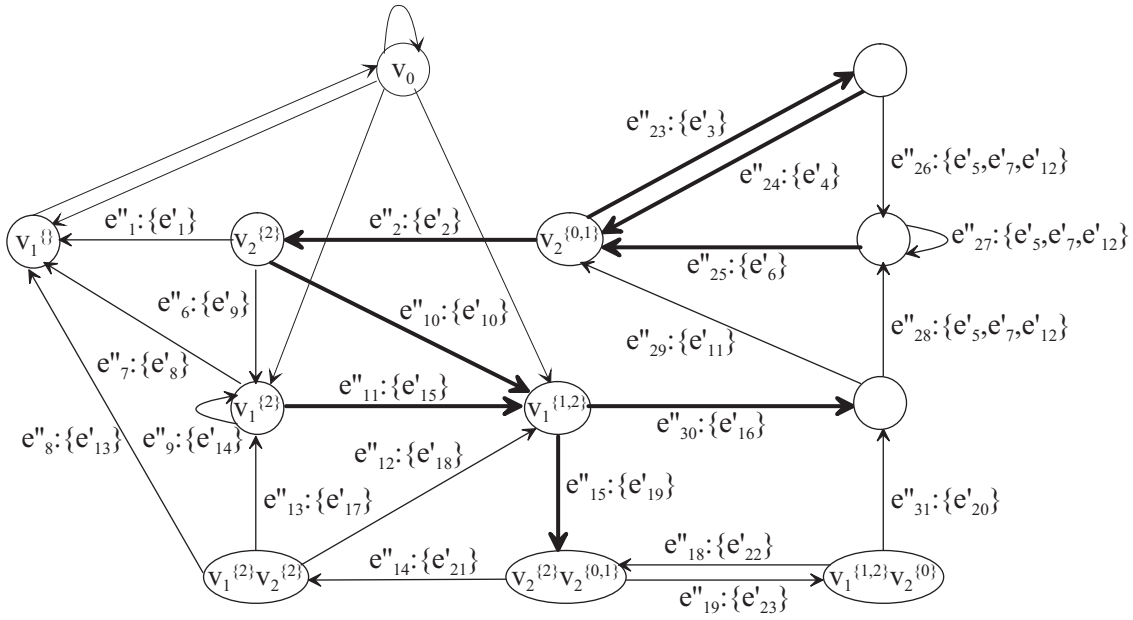

FIGURE 15. The $Q^\diamond$-based G constructed for the $Q^\bullet$ in Figure 10.

first method of choice for eliminating such a v. Note that such a v always comprises a (v', PI, PF) for which the vertex v' from $Q^\bullet$ is not a singleton set and, hence, not $\{s_0\}$, for otherwise v could not be problematic in the described sense. Such a v, hence, never qualifies as a point where a rooted tour in G could (re-)enter or (re-)exit $Q^\diamond$, and neither do the vertices into which it is split. In other words, as vertex splitting is the only transformation generating new vertices during $Q^\diamond$ finalization, every candidate for an entry or an exit point of $Q^\diamond$ is conveniently present already in $Q_1^\diamond$.

In Figure 13, there is also a problematic vertex, namely $v_2^{\{0\}}$. For example, in Figure 6 we see that e'_3 and e'_6 are non-adjacent, implying that the walk "e''_3, e''_{21}" is infeasible. So we split the vertex into three vertices. In Figure 15, whose central subgraph is the final version of the constructed $Q^\diamond$, these are the vertices which are unlabelled. The $Q^\diamond$ happens to be isomorphic to $dunf(M^*)$ and the resulting G happens to be isomorphic to the one in Figure 6. However, if we observe edges e'_5, e'_7 and e'_{12} from Figure 6 and the corresponding edges e''_{26}, e''_{27} and e''_{28} in Figure 15, we see that the edges now have the same label. The new symmetry has been inherited from $Q^\bullet$, in which τ_6 and τ_7, the corresponding transitions from M, are represented by the same edge.

Introducing a by-pass edge $e_{\bar{e}}$ for a walk $\bar{e} = e_1, \ldots, e_k$, $e_i = (v_i, v_{i+1}, E_i, CP^+(E_i))$ for $1 \leq i \leq k$, in the current version of $Q^\diamond$ corresponds to introducing a by-pass for every walk from the corresponding version of Q° which is in $SH(\bar{e})$ and, hence, makes sense only if $SH(\bar{e})$ is non-empty. In that case, $e_{\bar{e}}$ is $(v_1, v_{k+1}, l_{\bar{e}}, CP^+(l_{\bar{e}}))$ with $l_{\bar{e}}$ the set of all $e_{\bar{e}'} = (init(\bar{e}'), fin(\bar{e}'), SH(\bar{e}'), CP(SH(\bar{e}')))$, with $PT(e_{\bar{e}'}) = WP(SH(\bar{e}'))$, for $\bar{e}'$ in $SH(\bar{e})$. Every edge of $\bar{e}$ which with the introduction of $e_{\bar{e}}$ becomes irrelevant for G may be deleted.

In many cases, edge by-passing is recognized as an alternative for vertex splitting, which, if too extensive, makes $Q^\diamond$ grow unacceptably large. In other words, instead of splitting a problematic vertex v, one might rather choose to by-pass it. In that case, one becomes interested in walks $\bar{e} = e_1, \ldots, e_k$ with $k \geq 2$, $init(e_1) \neq v$, $fin(e_k) \neq v$ and $fin(e_i) = v$ for $1 \leq i < k$, introducing a by-pass edge for some of the promising ones and then deleting all the incoming edges of v and all the consequently irrelevant parts of $Q^\diamond$. When this transformation is applied to a loop partially unfolded through vertex splitting, the combined transformation is analogous to the second transformation for the elimination of conditional transitions given in [4].

When enhancing $Q^\diamond$ into G, one introduces an auxiliary edge to or from a vertex v in $V^\diamond$ exactly where such an edge is safe and desired for every vertex v' in the underlying Q° for which v virtually comprises a copy, by being $sa(v')$: The auxiliary edges from v_0 to $Q^\diamond$ end in those v in $(V^\diamond \cap V_1^\diamond)$ whose members are all of the form $(\{s_0\}, PI(\varepsilon), \_)$, while the auxiliary edges from $Q^\diamond$ to v_0 start in those v in $(V^\diamond \cap V_1^\diamond)$ whose members are all of the form $(\{s_0\}, \_, PF(\varepsilon))$. If Restrictions 7, 12 and 13 are satisfied, it is also safe to introduce an auxiliary zero-cost edge between any pair of vertices v and v' in $(V^\diamond \cap V_1^\diamond)$ for which every pair of an (S_1, PI, PF) in v and an (S_2, PI', PF') in v' satisfies $S_1 = S_2 = \{s_0\}$, $PI' \subseteq PI$ and $PF \subseteq PF'$.

Construction of $\bar{e}^*$, an optimal rooted tour in G, proceeds as if the central subgraph of G was Q°. Once $\bar{e}^*$ is constructed, its projection $\bar{e}^\diamond$ onto the edges of

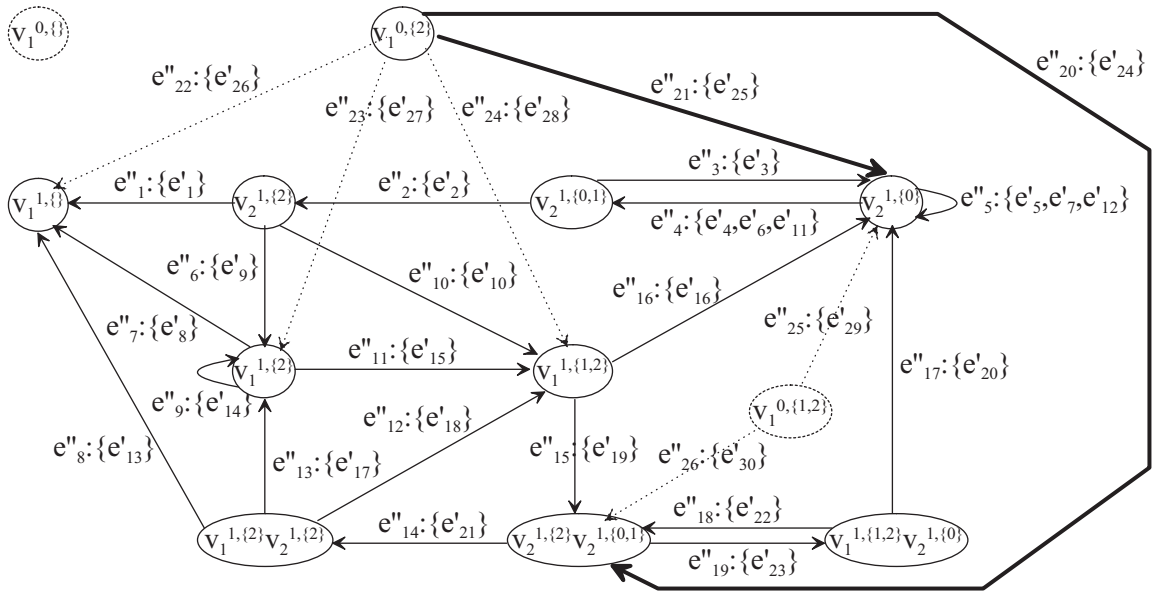

FIGURE 16. The $Q_2^\diamond$ constructed for the example test continuation construction problem.

$Q^\diamond$ is in general a concatenation $\bar{e}_1 \cdot \ldots \cdot \bar{e}_k$ of walks in $Q^\diamond$, with each $\bar{e}_i$ representing a sheaf of walks in Q° starting in a vertex in which a rooted tour in a Q°-based G could (re-)enter Q° and ending in a vertex in which such a tour could (re-)exit Q°. The candidates for $\bar{e}^\circ$, the selected concatenation of walks in Q°, are, hence, all $\bar{e}'_1 \cdot \ldots \cdot \bar{e}'_k$ with $\bar{e}'_i$ in $SH(\bar{e}_i)$ for $1 \leq i \leq k$. It is, however, recommended to first make for every $SH(\bar{e}_i)$ a more detailed assessment of the cost and the witnessing power of its individual members, whose estimated value has so far been that of the sheaf. Once an $\bar{e}^\circ = e_1, \ldots, e_{k'}$ is selected, the corresponding test $\bar{x}^*$ is $in(lab(e_1)) \cdot \ldots \cdot in(lab(e_{k'}))$.

One of the optimal tours in the $Q^\diamond$-based G in Figure 15 is the one for which $\bar{e}^\diamond$ is "$e''_{11}, e''_{30}, e''_{28}, e''_{25}, e''_{23}, e''_{24}, e''_{2}, e''_{10}, e''_{15}, e''_{14}, e''_{8}$". The only corresponding candidate for $\bar{e}^\circ$ is the $\bar{e}^\circ$ constructed in Section 7.5.2, so that $\bar{x}^*$ is also exactly as before.

8.3. EFSM-Based Construction of Test Continuations

If the constructed test $\bar{x}^*$ is required to be an $\bar{x}^\circ_{i-1} \cdot \bar{x}^\dagger_i$ for a non-initial i-th round of adaptive testing, with $\bar{x}^\circ_{i-1}$ the already applied part of the test and the I/O sequence produced in response a $\bar{z}^\circ_{i-1}$, the procedure from Section 8.2 must be modified analogously to how we in Section 7.6 modified the procedure from Section 7.5. As the procedure from Section 8.2 is just an adaptation of that from Section 7.5 to a more implicit system model, the procedure below is just an analogous adaptation of that from Section 7.6.

As in Section 8.2, the adopted system model is not an FSM M, but a corresponding EFSM $Q^\bullet$. One, hence, does not construct M^+, M^* and Q°, as in Section 7.6, but their respective WSGs Q^+, Q^* and $Q^\diamond$. Up to and including the first version $Q_1^\diamond$ of $Q^\diamond$, the procedure for constructing the central subgraph of G is exactly as described in Section 8.2, except that, as in Section 7.6, one must take care to include into the GCPS the special-purpose predicates p^+ and p^* and to represent at least one DRT whose label has prefix $\bar{z}^\circ_{i-1}$.

Instead of constructing Q_2° from Q_1°, as in Section 7.6.2, one then uses a slightly modified procedure and constructs $Q_2^\diamond$ from $Q_1^\diamond$. The modifications are as follows:

(i) Instead of $Start(Q^\circ)$, one needs to know $Start(Q^\diamond)$, the set of all vertices in $V_1^\diamond$ which are $sa(v)$ for a vertex v in $Start(Q^\circ)$, i.e., the set of all vertices in $V_1^\diamond$ whose members are all of the form $(\{s_0\}, PI(\varepsilon), \_)$.

(ii) The introduced by-pass edges are not those in $Past_1(Q^\circ)$, but those in $Past_1(Q^\diamond)$, the set to which every walk $\bar{e}$ in $Q_1^\diamond$ with $init(\bar{e})$ in $Start(Q^\diamond)$ and with $SH(\bar{e})$ comprising a walk $\bar{e}'$ with $SH(\bar{e}')$ comprising a walk $\bar{e}''$ with $lab(\bar{e}'') = \bar{z}^\circ_{i-1}$ contributes a by-pass edge $(init(\bar{e}), fin(\bar{e}), l, CP^+(l))$ with l a set to which every such $\bar{e}'$ in $SH(\bar{e})$ contributes a by-pass edge $e_{\bar{e}'} = (init(\bar{e}'), fin(\bar{e}'), l', CP(l'))$ with l' the set of all such walks $\bar{e}''$ in $SH(\bar{e}')$ and with $PT(e_{\bar{e}'}) = WP(l')$.

For illustration, let us return to the test continuation construction problem considered in Section 7.6, but this time assuming that the problem model for the second round of testing is not the FSM from Figure 1 with the edge τ_5 deleted, but the EFSM from Figure 10(ii) with the edge $\{\tau_5\}$ deleted. The constructed $Q_2^\diamond$ is given in Figure 16, in which the $Q_1^\diamond$ subgraph is the part of the

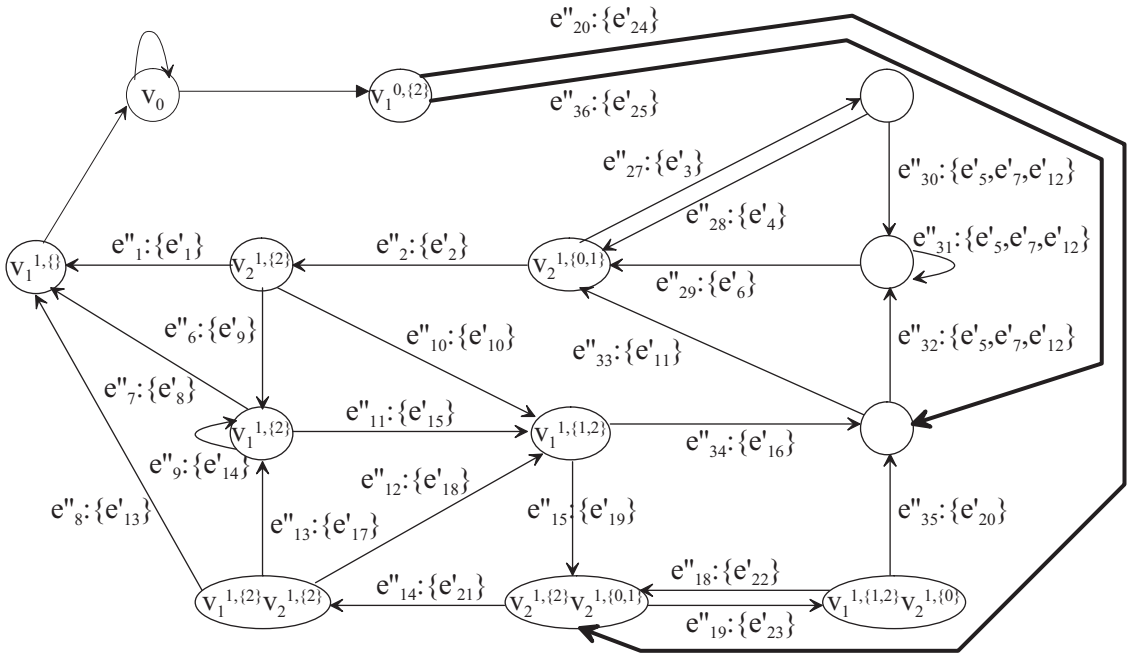

FIGURE 17. The $Q^\diamond$-based G constructed for the example test continuation construction problem.

graph not drawn bold, the members of $Start(Q^\diamond)$ are the vertices comprising just $v_1^{0,\{\}}$, $v_1^{0,\{2\}}$ or $v_1^{0,\{1,2\}}$, the members of $Past_1(Q^\diamond)$ are the edges drawn bold, edge labels comprise edges from Figure 9, and edge names e''_1 to e''_{19} identify the corresponding edges in Figure 12,.

Instead of constructing Q_3° from Q_2°, as in Section 7.6.2, one then uses a slightly modified procedure and constructs $Q_3^\diamond$ from $Q_2^\diamond$. The edges deleted are now those which start in $Start(Q^\diamond)$, but are not in $Past_1(Q^\diamond)$. In Figure 16, such edges are e''_{22} to e''_{26}.

Instead of constructing Q_4° from Q_3°, as in Section 7.6.2, one then uses a slightly modified procedure and constructs $Q_4^\diamond$ from $Q_3^\diamond$. The modification is that, besides introducing by-pass edges, one might also want to perform some edge splitting, vertex splitting, edge deletion and/or vertex deletion, for the reasons and in the way described in Section 8.2, particularly for the mandatory prevention of walks $\bar{e}$ with empty $SH(\bar{e})$. For our example, one would, for the reasons described in Section 8.2, split edge e''_4 and vertex $v_2^{1,\{0\}}$, previously called $v_2^{\{0\}}$.

To construct from $Q_4^\diamond$ the first approximation of G, one follows the procedure from Section 8.2 for constructing G, except that, obeying an analogy of a rule from Section 7.6, one refrains from introducing auxiliary edges starting in $Start(Q^\diamond)$. As for the edge costs, one, as in Section 7.6, follows the usual policy for discouraging the subsequently constructed tour $\bar{e}^*$ from passing from v_0 to the central subgraph multiple times. To finalize G, and thereby the central subgraph $Q^\diamond$, one then, as in Section 7.6, deletes all edges and vertices not lying on a rooted tour. For our example, G is given in Figure 17. We see that it is isomorphic to the G from Figure 9, but exhibits the same additional symmetry in edge labels (see edges e''_{30}, e''_{31}, e''_{32}) as the G from Figure 15 with respect to the G from Figure 6.

To construct $\bar{e}^*$, an optimal rooted tour in G, one proceeds as in Section 8.2, but, obeying an analogy of a rule from Section 7.6.2, accepts the constructed tour only if it enters $Start(Q^\diamond)$ exactly once. If this is the case, a corresponding test $\bar{x}^*$, i.e. $\bar{x}^\circ_{i-1} \cdot \bar{x}^\dagger_i$, is derived as in Section 8.2. For our example, the predicted witnessing power of individual edges in G is that of the corresponding edges in Figure 9. Hence, one of the optimal tours is also the one for which $\bar{e}^\diamond$ is "$e''_{36}, e''_{32}, e''_{29}, e''_{27}, e''_{28}, e''_2, e''_{10}, e''_{15}, e''_{14}, e''_8$". The only corresponding candidate for $\bar{e}^\circ$ is the $\bar{e}^\circ$ constructed in Section 7.6.2, so that $\bar{x}^*$ is exactly as before.

9. CONCLUSIONS

Although methods for systematic test generation have for a long time been strongly model-based, there has so far been little awareness of the common foundations of the employed models. More precisely, there has been little universally useful knowledge on what information to put into such a model and how to encode it to make the model easy to construct and investigate. Without such knowledge, the existing specific methods are difficult to assess, compare and fruitfully combine. Not to forget that by putting them into a broader perspective, one can gain additional insight on possible improvements.

This is why we in [1] proposed a test generation method which is both model-based and generic, but nevertheless reduces the considered test generation problem to a generic optimization problem on a graph, thereby facilitating exploitation of domain-unspecific optimization tools. In the present paper, we made the method even less strategy-specific and provided a generic link between test generation methods for DFSMs and those for NEFSMs.

We assumed that the given (N)EFSM is just a concise representation of an (N)FSM, but the method is actually applicable to any NEFSM on which the specified procedures run to a termination, i.e., produce a finite problem model. If this is not the case, one can still resort to an approximate problem model, by conducting the procedures in the recommended approximate manners. This is necessary also if the state space is very large or if the nondeterminism present in the system is not sufficiently bounded.

If the system model is precise and every step of the method is conducted precisely, the generated test is among the optimal ones for the adopted testing strategy, because the generalized method is semantically just a clever application of the original one, which has been proved to produce optimal tests [1]. Otherwise, the quality of the test depends primarily on the selection of the candidate tests represented in the problem model. When deciding which candidates to represent, one would typically rely on the suggestions of some simpler test generation methods, possibly based on different testing strategies, thereby conveniently combining the optimization power of each of the methods and the inherent optimization power of the generic method.

We plan to generalize the method also to timed systems, more specifically, to such which can be adequately modelled as untimed NEFSMs, for example, as Set/Expire automata [21]. With the support for multi-criteria optimization, test duration could easily be incorporated as one of the optimization criteria. Another interesting generalization would be to distributed testing, where the main issue would be formalization of strategies overcoming the involved controllability and observability problems [22] without much communication between the distributed testers.

REFERENCES

[1] Kapus-Kolar, M. (2007) Testing as collecting of evidence: An integrated approach to test generation for finite state machines. *Computer Journal*, **50**, 315–331.

[2] Robinson-Mallett, C., Liggesmeyer, P., Mücke, T. and Goltz, U. (2006) Extended state identification and verification using a model checker. *Information and Software Technology*, **48**, 981–992.

[3] Luo, G., von Bochmann, G. and Petrenko, A. (1994) Test selection based on communicating nondeterministic finite state machines using a generalized Wp-method. *IEEE Transactions on Software Engineering*, **20**, 149–162.

[4] Hierons, R. M., Kim, T.-H. and Ural, H. (2004) On the testability of SDL specifications. *Computer Networks*, **44**, 681–700.

[5] Petrenko, A., Boroday, S. and Groz, R. (2004) Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering*, **30**, 29–42.

[6] Duale, A. Y. and Uyar, M. Ü. (2004) A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers*, **53**, 614–627.

[7] Hierons, R. M. and Harman, M. (2004) Testing conformance of a deterministic implementation against a non-deterministic stream X-machine. *Theoretical Computer Science*, **323**, 191–233.

[8] Lenstra, J. K. and Kan, A. H. G. R. (1976) On general routing problems. *Networks*, **6**, 273–280.

[9] Hierons, R. M. (2006) Separating sequence overlap for automated test sequence generation. *Automated Software Engineering*, **13**, 283–302.

[10] Anido, R., and Cavalli, A. (1995) Guaranteeing full fault coverage for UIO-based testing methods. *Proc. IWPTS'95*, Evry, France, 4–6 September, pp. 221–236. Chapman & Hall, London.

[11] Aho, A. V., Dahbura, A. T., Lee, D. and Uyar, M. Ü. (1991) An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. *IEEE Transactions on Communications*, **39**, 1604–1615.

[12] Chen, W.-H. (1998) Test sequence generation from the protocol data portion based on the Selecting Chinese Postman algorithm. *Information Processing Letters*, **65**, 261–268.

[13] Chen, W.-H. and Lu, C.-C. (2002) Executable test sequence for the protocol control and data flow property with overlapping. *Proc. ISCC'02*, Taormina, Italy, 1–4 July, pp. 251–257. IEEE Computer Society Press.

[14] Chen, W.-H. (2003) An optimization technique for protocol conformance testing based on the Wp method. *International Journal of Applied Science and Engineering*, **1**, 45–54.

[15] Chen W.-H. (2005) Executable test sequence for the protocol data portion based on two criteria. *Journal of Information Science and Engineering*, **21**, 529–545.

[16] Petrenko, A., Yevtushenko, N. and von Bochmann, G. (1996) Testing deterministic implementations from nondeterministic FSM specifications. *Proc. IWTCS'96*, Darmstadt, Germany, 9–11 September, pp. 125–141. Chapmann & Hall, London.

[17] Hierons, R. M. (1998) Adaptive testing of a deterministic implementation against a nondeterministic finite state machine. *Computer Journal*, **41**, 349–355.

[18] Hierons, R. M. (2003) Generating candidates when testing a deterministic implementation against a nondeterministic finite-state machine. *Computer Journal*, **46**, 307–318.

[19] Hierons, R. M. (2004) Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Computers*, **53**, 1330–1342.

[20] Hierons, R. M. (2006) Applying adaptive test cases to nondeterministic implementations. *Information Processing Letters*, **98**, 56–60.

[21] Khoumsi, A. (2005) Complete test graph synthesis for symbolic real-time systems. *Electronic Notes in Theoretical Computer Science*, **130**, 79–100.

[22] Hierons, R. M. and Ural, H. (2008) The effect of the distributed test architecture on the power of testing. *Computer Journal*, **51**, 497–510.