A Fast Search for Temporal Properties of Requirements

Michael E. Houle, Tim Menzies, John Powell

Abstract—**Early testing of requirements can decrease the cost of removing errors in software projects. However, unless done carefully, that testing process can significantly add to the cost of requirements engineering. In this paper, we consider requirements systems models that can be expressed in the form of *topoi*, which are graphs representing statements of gradual influences among variables. We propose an iterative search heuristic for examples of multi-step temporal properties supported by topoi, based on a very efficient exact algorithm for producing a partition of variable-state combinations into two temporally-consistent sets. Such examples may be of interest in their own right, or may serve as counter-examples to conjectures within the model. Topoi have the advantage of being easy to formulate, although they cannot express certain complex concepts such as iteration and sub-routine calls. Despite their unsuitability for traditional model checking domains, we provide evidence of their usefulness for requirements analysis.**

I. Introduction

The case for more formality in requirements engineering is overwhelming. Many errors in software can be traced back to errors in the requirements [2]. Often, the conception of a system is improved as a direct result of the discovery of inadequacies in the current conception. The earlier such inadequacies are found, the better, since the cost of removing errors at the requirements stage can be orders of magnitude cheaper than the cost of removing errors in the final system [3].

The benefit of formally checking a system is that a formal analysis can find more errors than standard testing. A single formal first-order query is equivalent to many white-box or black-box test inputs.

The cost of testing requirements engineering may be impractically high. These costs include the *modelling cost*, the *execution cost*, the *personnel cost*, and the *development brake*. The *modelling cost* is incurred when analysts create the *systems model*, and the *properties model* needed for formal analysis. Both models need to be expressed in some machine-readable form. The properties model contains a formal temporal logic[1] description of the invariants that must be tested in the systems model, and is often much smaller than the systems model.

A rigorous analysis of formal properties has a high *execution cost*, since it implies a full-scale search through the systems model. For example, if a given systems model has n variables, each of which may take on a finite number of unique values m, then the size of the state space associated with that model is m^n. This space can be too large to explore, even on today's fast machines. Despite extensive research into speeding up this search (see the discussion of related work in §IV), analysts often must painstakingly rework the systems and properties models into more abstract and succinct forms that are small enough to permit formal analysis. Analysts skilled in formal methods must be recruited or trained. Since such analysts are generally hard to find and retain, formal methods have a high *personnel cost*. These costs can be so high that the requirements must be frozen for some time while one performs the formal analysis. Hence, one of the costs of formal analysis is *development brake* that slows down the requirements process. Slowing down the requirements process is unacceptable for fast moving software companies.

Ideally, an automated method for reducing the cost of testing requirements would reduce the execution cost as well as the cost and skill involved in building the properties and systems models. If achievable, such a method would also reduce the personnel cost, since it would not require such highly-skilled analysts. Having reduced the personnel, modelling, and execution costs, this hypothetical method would inevitably decrease the development brake.

Some progress has already been made in reducing the cost of properties modelling using *temporal logic patterns*. Dwyer, Avrunin & Corbett [5, 6] have identified patterns within the temporal logic formula seen in many real-world properties models. For each pattern, they have defined an expansion from the intuitive pseudo-English form of the pattern to a formal temporal logic formula. In this way, analysts are shielded from the complexity of formal logics. For example, the simple pseudo-English statement

$$\textit{always } (\textit{brake} = \textit{on}) \textit{ between } (\textit{danger} = \textit{seen}) \textit{ and } (\textit{car} = \textit{stop})$$

can be automatically expanded into the more arcane formal statement:

$$\Box((((\textit{danger} = \textit{seen}) \wedge !(\textit{car} = \textit{stop}) \ \wedge \Diamond \ (\textit{car} = \textit{stop})) \\ \rightarrow ((\textit{brake} = \textit{on}) \ \cup \ (\textit{car} = \textit{stop})))$$

One drawback with temporal logic patterns is that while complex temporal formulae can be automatically generated from intuitive pseudo-English, the execution cost remains. That is, even though we can quickly build the properties model, we may not be able to explore all of that model.

In this article, we argue that we can greatly reduce the execution cost for a class of systems models seen in the requirements stage. We assume that temporal logic patterns have reduced the modelling cost of the properties. Hence, what remains is the modelling cost of the system. The key to this reduction is SP2, a new algorithm for generating *time series* (defined below) from *topoi*, which are statements of gradual influences between variables [4]. Topoi can represented graphically by *topos diagrams*, an example of which is shown in Figure 1. The experience of Dieng, Corby & Lapalut [4] and Menzies [7] is that domain experts can sketch topoi very quickly and so (for requirements that are topos-compatible) our approach also reduces the systems modelling cost.

These cost-reduction benefits can only be realized if we accept certain restrictions. Specifically, the systems model must be expressed as topos diagrams. Topoi are not very expressive, and exclude statements such as first-order assertions, iterations, sub-routine calls, and assignments. Due to these language limitations, our approach is not suited to domains that need the excluded statements, such as complex protocols seen in concurrent processes.

These restrictions are not fatal to the modelling process, at least at the requirements stage. We will show by means of examples that topos diagrams are sufficient to represent a wide range of diagrams seen in certain approaches to requirements engineering and recording design rationales. Hence, when we say that this approach is practical and useful, we really mean *practical and useful for early life cycle requirements discussions only*.

II. About Topoi

Our approach assumes that requirements systems models are expressed in the form of *topoi*, which are statements of gradual influ-

Michael Houle is with IBM Research, Tokyo Research Laboratory, 1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken 242-8502, Japan (on leave from the School of Information Technologies, The University of Sydney, Sydney, NSW 2006, Australia), meh@trl.ibm.co.jp

Tim Menzies is with the Lane Department of Computer Science & Electrical Engineering, West Virginia University, PO Box 6109, Morgantown, WV 26506-6109, tim@menzies.com

John Powell is with the Jet Propulsion Laboratory Quality Assurance Office, Mail Stop 125-233, 4800 Oak Grove Drive, Pasadena, CA 91109-8099, john.powell@ivv.nasa.gov

Much of this work was performed while Menzies & Powell worked at the NASA/WVU Software Research Laboratory, IV&V Facility, Fairmont, West Virginia, partially supported by NASA through cooperative agreement #NCC 2-979.

An earlier and shorter version of this paper appeared at ICSE 2001 as *Fast Formal Analysis of Requirements via "Topoi Diagrams"* [1].

[1] Temporal logic is classical logic augmented with temporal operators such as $\Box X$ (always X is true), $\Diamond X$ (eventually X is true), $\bigcirc X$ (X is true at the next time point), $X \ \mathcal{U} \ Y$: (X is true until Y is true).

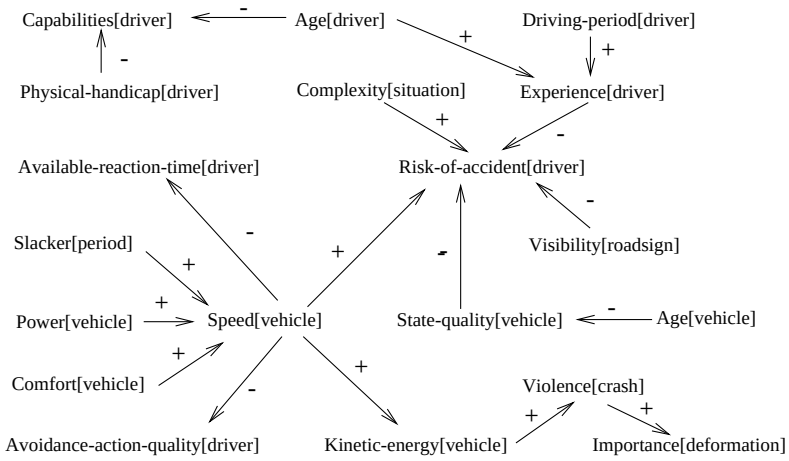

Fig. 1. An example topos from [4]. Informally, we say that $+$ indicates "encourages" while $-$ indicates "discourages".

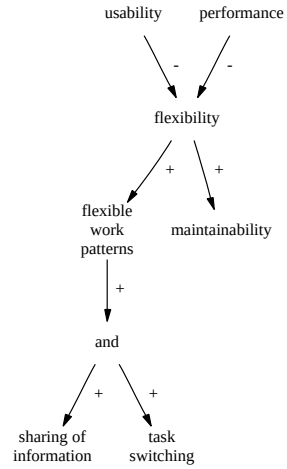

Figure 2.i: A soft-goal graph: the *and* node denotes that both *sharing of information* and *task switching* are enabled by *flexible work patterns*.

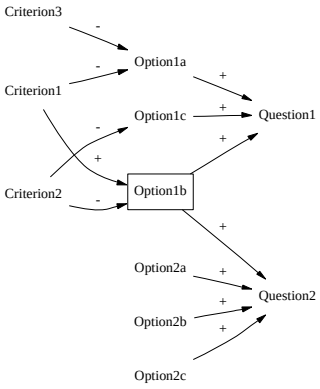

Figure 2.ii: A questions-options-criteria graph from [8].

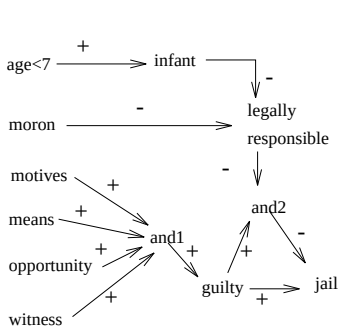

Figure 2.iii: Topos from Figure 3.

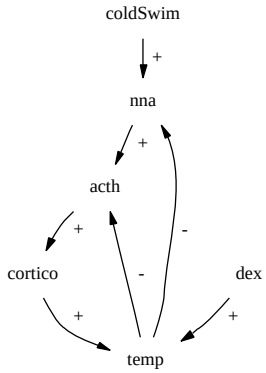

Figure 2.iv: The Smythe '87 theory. From [9]: statements of gradual knowledge relating to laboratory experiments on mammals.

Fig. 2. Sample topos diagrams.

ences such as (i) the more X, the more Y; (ii) the less X the less Y; (iii) the more X, the less Y; or (iv) the less X the less Y. Dieng et al. provide many examples of topoi from their records of interviews with experts [4], in which they offer such statements as: *the more there is water infiltration in the roadway body, the worse the foundation risks to be*; and *if there is a punctual undressing and if the roadway is between five and fifteen years old, then the cause 'too old coating' is all the more certain since the roadway is older*.

Our experience has always been that the systems modelling cost with topoi is very low. Topos graphs can be quickly generated in the requirements stage. Two feuding stakeholders with two marker pens and one whiteboard can generate many, many topoi in just a few hours.

```
if guilty  then jail.
if age < 7 then infant.
if infant or moron
   then not legally_responsible.
if legally_responsible and guilty
   then jail.
if motive and means and opportunity and witnesses
   then guilty.
if guilty and not legally_responsible
   then not jail.
```

Fig. 3. Rule-based requirements from a legal system.

A. Example Topoi

Topos graphs can be found in many domains. Figure 1 shows a topos from an insurance domain using the graphical notation of the 3DKAT tool of Dieng et al. [4]. Figure 2.i shows a *soft-goal* graph of the style used by Mylopoulos, Cheng & Yu [10] to represent gradual knowledge about non-functional requirements. Here, an expert describes how to increase business flexibility. Figure 2.ii shows a *questions-options-criteria* (QOC) graph from the design rationale community [8]. In QOC graphs, questions suggest further options, and deciding on a certain option can raise other questions. Those options that are selected are surrounded by a box. Options are assessed according to criteria based on gradual knowledge that *tend to support* or *tend to reject* them. QOCs can succinctly summarize lengthy debates; for example, 480 sentences uttered in a debate between two analysts on interface options can be displayed in a QOC graph on a single page [11]. Figure 2.iii shows a topos generated from the requirements of a rule-based legal system, shown in Figure 3. This translation assumes that propositions in the rule base are modelled as a belief-strength pair such as (infant, X), for some $0 \leq X \leq 1$.

B. Topoi Are Over-generalized and Over-constrained

Topoi edges constrain the values of the node variables they connect and, often, they are *over-constrained*; that is, they cannot all be satisfied simultaneously. However, it is unrealistic to assume that every influence represented by a topos edge must have an effect. To make use of topoi in explaining an observation or testing the plausibility of a hypothesis, we must therefore be prepared to use only subsets of their edges. For example, consider Figure 2.i and the case of *flexibility* and *usability* increasing while *performance* decreases. Note that we can explain *flexibility* increasing, but only if we can avoid the contradictory conclusion offered by the edge from *usability*.

Ignoring inconsistencies when it is convenient to do so runs the risk of excessive *over-generalization*; that is, more conclusions may be reachable in the abstract model than in the real system being modelled. A pre-experimental concern is that informal topoi are so over-generalized that we could use them to infer any set of desired properties (researchers in qualitative simulation call this the *chatter* problem [12]). This concern is not always justified. While topoi are over-generalized, they may still be restrictive enough to be useful. For example, recall Figure 2.i and the fragment:

$$usability \xrightarrow{-} flexibility \xleftarrow{-} performance$$

Note that there is no way to explain the output of increased *flexibility* ($\{flexibility\uparrow\}$) from the input of increased *usability* and *performance* ($\{usability\uparrow, performance\uparrow\}$).

For another example, consider Figure 4 due to Smythe, who extracted the list of gradual influences claimed by a set of articles from different authors discussing human internal physiology [13]. Feldman & Compton, then Menzies, showed that despite over-generalization, Figure 4 could be used to detect a large number of previously-unnoticed errors published in international refereed journals [14–16]. Interestingly, these faults were found using the tables of data published to support those theories: their experiments reported observations that could not be reproduced via Figure 4.

For a last example of the utility of topoi, in §III-H we will discuss experiments where a topos-based search for temporal properties outperformed a state-of-the-art model checker (SPIN [17]).

C. Comprehension Problems

Topoi seem quite simple, but often defy manual analysis, possibly due to their over-constrained and over-generalized nature. For example, Menzies' topos tester [7] found behaviors in very small topoi (Figure 2.iv) that had not been discovered even after days of manual analysis. The difficulty in analyzing topoi worsens as the topoi become more complex, as often happens when statements of gradual influence are collected from more than one stakeholder. Our experience has been that humans cannot confidently explore by hand the dense maze of interactions within topoi as complex as the one shown in Figure 4, regardless of the magnification and quality of the layout.

D. Topoi: Formal Definition

Formally, a topos is a directed, possibly cyclic graph $G\langle V, E\rangle$, with the node set V corresponding to variables, and the edge set E indicating influences of one variable upon another.

Each node $v \in V$ is associated with a *state* that describes the type of change that the corresponding variable is experiencing. These states can themselves change over time:

up:	the value of v is increasing (denoted by $v\uparrow$);
down:	the value of v is decreasing (denoted by $v\downarrow$);
steady:	the value of v is set and is neither increasing or decreasing;
unknown:	the value of v has not yet been set.

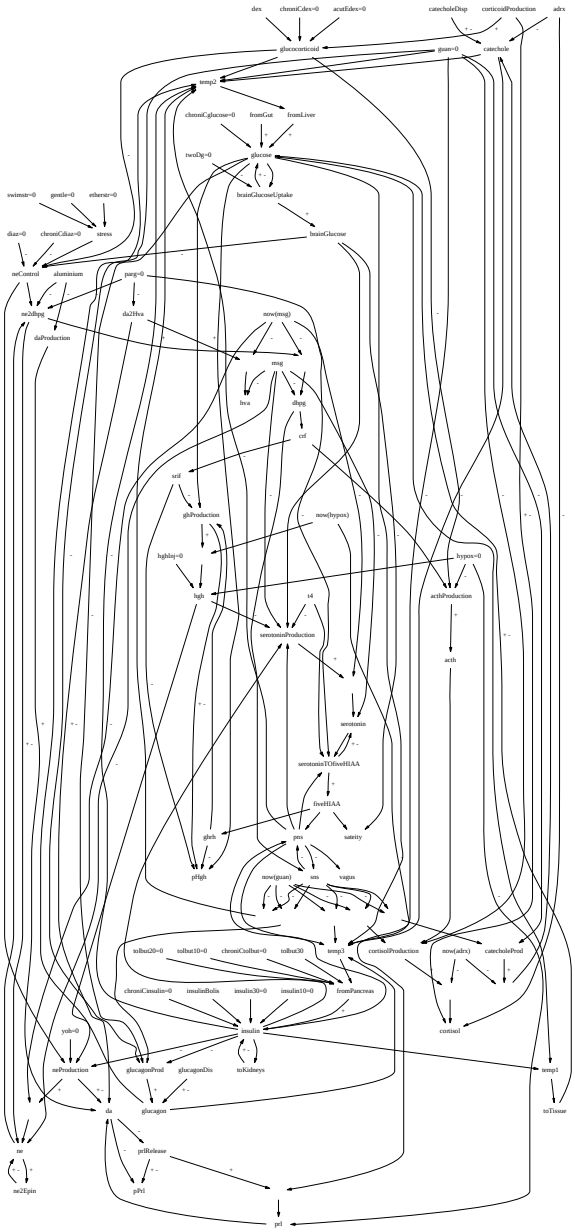

Fig. 4. A large topos with many loops.

An edge $e = (u, v)$ of E must be of one of two types, according to whether the influence of u on v is *positive* or *negative*. A positive edge (indicated by $u \xrightarrow{+} v$) enables v to become *up* if u is *up*, and to become *down* if u is *down*. A negative edge (indicated by $u \xrightarrow{-} v$) allows v to become *down* if u is *up*, and to become *up* if u is *down*.

In a topos, it is possible for pairs of competing influences to cancel out. For example, in a topos with edges $u \xrightarrow{+} w$ and $v \xrightarrow{+} w$, the states $u\uparrow$ and $v\downarrow$ would enable both $w\uparrow$ and $w\downarrow$. In such a situation, either choice of state would be allowed; however, the competing influences can also be recognized simultaneously by placing w into the state *steady*.

Finally, nodes are initially in the state *unknown* until such time as they change to one of the other three states. Once a node has changed state, it can never again become *unknown*.

This formal semantics is sufficient to guide the translation of topoi for a formal model checker such as SPIN. Figure 5 shows the results of

```
#define DOWN    0
#define STEADY  1
#define UP  2
#define UNDEF   3
#define ARRIVED 0
#define LEFT    1

byte chg_cold_swim = UNDEF /* chg_cold_swim = {ARRIVED,LEFT} */
byte chg_dex = UNDEF       /* chg_dex = {ARRIVED,SWIM}       */
byte cold_swim = UNDEF     /* cold_swim = {DOWN,STEADY,UP}   */
byte dex = UNDEF           /* dex = {DOWN,STEADY,UP}         */
byte temp = UNDEF          /* temp = {DOWN,STEADY,UP}        */
byte nna = UNDEF           /* nna = {DOWN,STEADY,UP}         */
byte acth = UNDEF          /* acth = {DOWN,STEADY,UP}        */
byte cortico = UNDEF       /* cortico = {DOWN,STEADY,UP}     */

active proctype smythe() {
      if
        ::dex == UNDEF -> dex = DOWN
        ::dex == UNDEF -> dex = STEADY
        ::dex == UNDEF -> dex = UP
      fi;
      if
        ::cold_swim == UNDEF -> cold_swim = DOWN
        ::cold_swim == UNDEF -> cold_swim = STEADY
        ::cold_swim == UNDEF -> cold_swim = UP
      fi;
      if
        ::chg_dex == UNDEF -> chg_dex = ARRIVED
        ::chg_dex == UNDEF -> chg_dex = LEFT
      fi;
      if
        ::chg_cold_swim == UNDEF -> chg_cold_swim = ARRIVED
        ::chg_cold_swim == UNDEF -> chg_cold_swim = LEFT
      fi;
      if
        ::chg_dex == ARRIVED -> temp = UP
        ::chg_dex == LEFT -> temp = DOWN
      fi;
      if
        ::chg_cold_swim == ARRIVED -> nna = UP
        ::chg_cold_swim == LEFT -> nna = DOWN
      fi;
      do
        ::(chg_cold_swim == ARRIVED && temp == UP) -> nna = STEADY
        ::(chg_cold_swim == LEFT && temp == DOWN) -> nna = STEADY
        ::temp == DOWN -> nna = UP
        ::temp == UP -> nna = DOWN
        ::temp == DOWN -> acth = UP
        ::temp == UP -> acth = DOWN
        ::nna == UP -> acth = UP
        ::nna == DOWN -> acth = DOWN
        ::acth == UP -> cortico = UP
        ::acth == DOWN -> cortico = DOWN
        ::cortico == UP -> temp = UP
        ::(cortico == UP && chg_dex == LEFT) -> temp = STEADY
        ::cortico == DOWN -> temp = DOWN
        ::(cortico == DOWN && chg_dex == ARRIVED) -> temp = STEADY
        ::(temp == UP && nna == UP) -> acth = STEADY
        ::(temp == DOWN && nna == DOWN) -> acth = STEADY
      od;
}
```

Fig. 5. Figure 2.iv expressed in the PROMELA language used by SPIN [17].

such a translation of Figure 2.iv. Inputs and outputs to that model are generated by studying *pairs* of experimental observations. The difference in the experimental treatments is the *input* to the system and the difference in the observations is the *output* to the model. For example, if we increase the injections of *dex* from one experiment to another, then we say that *chg_dex = arrived*.

E. Model Checking and Topoi

We can test topoi using libraries of expected or desired system behavior. Such libraries can be quickly built via interviews with users. We have found it useful to structure these interviews in an OO framework. After generating use cases and particular scenarios, we ask designers to specify the expected inputs and required outputs for each scenario. This leads to the generation of two artifacts: a topos graph describing how designers believe influences should propagate through a system, and a properties model. The latter can be in the language of temporal logic, as used by model checkers such SPIN.

While standard model checkers can check properties of Figure 5 in less than a second, they can fail to terminate for larger systems models. In one study, we offered 40 temporal properties to SPIN along with Figure 4, expressed in the PROMELA language. Given 100MB of maximum RAM, SPIN ran out of memory for most of the trials. We suspect that SPIN was attempting to search too much of what turns out to be a very large search space. Figure 4 contains 80 variables, each of which has can assume one of four states (*up*, *down*, *steady*, *unknown*); in this case, the total state space is of size at least $4^{80} \approx 10^{48}$. In a second study, we reduced the size of the state space to $3^{80} \approx 10^{38}$ by disallowing *steady* values. However, even in this reduced system, SPIN ran out of memory for 29 of the 40 properties [18]. In summary, even though topoi can be processed by standard model checkers, in practice, this may not be feasible.

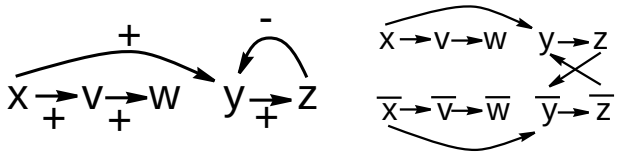

Figure 6.i: analyst topos

Figure 6.ii: twinned topos (edge weights not shown).

Fig. 6. Generation of a twinned topos from an analyst's topos.

III. SP2 AND RAPTURE

While general topoi defeat general-purpose model checkers, specialized search engines can quickly check the temporal properties of related graphs we call *twinned topoi*. A temporal property for the original topos can be demonstrated using its corresponding twinned topos, by checking for set membership of the property negation within a *time series* (defined below) of nodes reachable from the input set $I \subseteq V$. Candidate time series can be generated for the set membership test in nearly-linear time for each, using the SP2 algorithm described later. If a given time series does not provide an example of some desired property, the SP2 input parameters can be slightly modified to produce a different yet similar time series, one that hopefully has a better chance of containing an example. We will refer to this parameter modification strategy as *coaxing*, and the iterative heuristic that makes use of it as RAPTURE. Our proposal is thus to replace the inherently exponential methods of general-purpose model checkers with the repeated application of a nearly-linear-time algorithm.

A. Twinned Topoi

Given an original topos graph $G_O\langle V_O, E_O\rangle$, the twinned topos of G_O is a graph $G\langle V, E\rangle$ such that:

- for every node $v \in V_O$, there exists a pair of nodes $v\uparrow, v\downarrow \in V$ uniquely corresponding to v with states *up* and *down*, respectively;
- every positive edge $u \xrightarrow{+} v \in E_O$ corresponds uniquely with a pair of edges $(u\uparrow, v\uparrow)$ and $(u\downarrow, v\downarrow)$ of E;
- every negative edge $u \xrightarrow{-} v \in E_O$ corresponds uniquely with a pair of edges $(u\uparrow, v\downarrow)$ and $(u\downarrow, v\uparrow)$ of E;
- for each edge pair $e_1, e_2 \in E$ as defined above, the edges e_1 and e_2 are assigned identical non-negative real weights $\lambda(e_1) = \lambda(e_2)$.

Given a node $w \in V$, we will refer to its unique paired node as the *twin* $\overline{w}$ of w. Similarly, each edge $e = (u, v) \in E$ is paired with its unique twin $\overline{e} = (\overline{u}, \overline{v})$. The twin of a twin is itself; that is, $\overline{\overline{w}} = w$ and $\overline{\overline{e}} = e$. A subset X of V is *consistent* if for all $x \in X$, we have $\overline{x} \notin X$. Given a consistent subset $X \subseteq V$, we denote by $\overline{X}$ the set of twins of the nodes of X. $\overline{X}$ is itself consistent, and again, $\overline{\overline{X}} = X$.

Twinned topoi are related to ordinary topoi in that the variable-state pairs $v\uparrow$ and $v\downarrow$ are explicitly represented as distinct nodes, and the state transitions they enable are represented as distinct edges. However, in twinned topoi, the *unknown* state is not explicitly represented, and the *steady* state is disallowed altogether. A topos and its corresponding twinned topos are shown in Figure 6. Note that we do not require analysts to write twinned topoi. Rather, analysts generate topoi such as Figure 1 and Figure 2, from which twinned topoi are generated by an automatic process in which the edge weights are initially chosen to be identical.

B. Time Series

In order to obtain examples of desired temporal properties, we use the twinned topos to generate sequences of variable states, called *time series*. The variable-state combinations are assigned to discrete time steps, where all state changes at a given step are considered to occur

simultaneously. Before their first appearance in the time series, all variables are assumed to be in an *unknown* state. Thereafter, from one time step to the next, the state of the variables may change to either *up* or *down*. The states at any time step must be *consistent*; that is, a variable cannot be *up* and *down* simultaneously. A time series must also be *realizable*, in that the state changes between time steps must correspond to a set of transitions within the twinned topos. A time series thus provides a history of variable-state combinations from which examples of temporal properties can be extracted.

More formally, given a twinned topos $G\langle V, E\rangle$, a *valid* time series is a sequence $S^{(k)} = (S_0, S_1, \ldots, S_k)$ of subsets of V satisfying the following conditions:
1. For all $0 \leq i \leq k$, S_i is consistent.
2. For all $0 < i \leq k$, all nodes of $S_i \subseteq V$ are reachable from the nodes of S_{i-1} via a collection of paths within G which (except for their starting nodes) pass only through nodes of S_i.
The twinned topos nodes in S_0 are called the *inputs* of S. Many different valid time series can share a common set of inputs S_0.

C. Testing Properties Using Time Series

Model checkers such as SPIN do not prove properties directly. Rather, they disprove properties by finding specific examples where the negation of the property holds. We do the same, by searching for such examples within time series.

As an example, consider the temporal logic statement

$$\Box(a \rightarrow (a \mathcal{W} b)), \tag{1}$$

which can be read as "whenever we have a, then we will always have a at least until such time as we have b". The expression $a \mathcal{W} b$ stands for "*a weak until b*", where b is not necessarily guaranteed to become true.

To disprove the above statement, it suffices to exhibit an example satisfying the negation

$$\begin{aligned} & \neg\Box(a \rightarrow (a \mathcal{W} b)) \\ = \quad & \Diamond\neg(a \rightarrow (a \mathcal{W} b)) \\ = \quad & \Diamond(a \wedge \neg(a \mathcal{W} b)) \end{aligned} \tag{2}$$

Statement (2) can be read as "we will eventually have a, such that from then onward, we will have *not-b* at least until after we have encountered *not-a*". Clearly, any example satisfying (2) would disprove (1).

To find examples satisfying temporal logic statements such as 2, it helps to think of the statement as being represented by a finite-state automaton (FSA) in which the transitions represent the passage of time from one step to the next. Each transition is labeled with conditions on the current state, which if satisfied trigger the transition to the next state. If a valid time series can be verified by the FSA of a temporal logic statement, then the time series constitutes an example satisfying the statement. Methods for converting temporal logic statements into FSA are well-known and discussed elsewhere [19].

Not every temporal logic statement can be represented by such finite state automata. For instance, no FSA can accept an example satisfying statements such as $\Box a$ ("always a"). However, any statement that can be satisfied by an example of finite length can be represented by some FSA.

D. Generating Time Series Using SP2

The key to generating consistent and realizable time series from a twinned topos is an algorithm, SP2, that partitions the set of all reachable nodes into two subsets. Provided that the input node set is itself consistent, the two reachable subsets generated by SP2 are each guaranteed to be consistent. These subsets are then used as the basis of an assignment of nodes of the twinned topos to the steps of a time series.

The full description of SP2, as well as the proof of the consistency conditions, are relatively complex, and for this reason have been relegated to an Appendix. For now, we note only the main features of the algorithm. Let I be a consistent subset of V serving as the inputs to the time series to be generated. The SP2 algorithm produces the following in $O(|V| + |E| \log |V|)$ time:
1. The set of all nodes $R \subseteq V$ reachable from I.
2. A partition of R into consistent subsets R_0 and R_1 such that $I \subseteq R_0$.
3. A directed *shortest path forest* (collection of trees) F spanning the nodes of R in such a way that:
(a) every node $v \in R$ is reachable via F from some node of I,
(b) the length of the unique path in F from the input nodes to v is the minimum possible over all such paths in the topos, where the length of any path is defined as the sum of its edge weights;
(c) every node of R_0 is reachable from some node of I via a path of F containing only nodes of R_0.

Furthermore, if a given node $w \in V$ is reachable from I but its twin $\overline{w} \in V$ is not, then w is guaranteed to appear in R_0 and not in R_1.

Consider the sequence $S^{(2)} = (I, R_0, R_1 \cup (R_0 \setminus \overline{R_1}))$. It is not hard to see that $S^{(2)}$ satisfies all the criteria of a valid time series: since I, R_0 and R_1 are all consistent, and since $R_0 \setminus \overline{R_1}$ contains no twins of elements of R_1, then $S^{(2)}$ must be consistent. From condition 3c stated above, $S^{(2)}$ must also be realizable. However, $S^{(2)}$ spans only two time steps past the initial input step, and thus may be too short to provide examples for the desired temporal properties.

Longer time series can be produced by generating additional consistent and realizable node sets from R_0 and R_1. For all $i > 1$, we define the node set R_i to be the set of nodes of R_{i-2} that are reachable from some node of R_{i-1} in G without passing through any nodes of $R \setminus R_{i-2}$. R_i can be generated iteratively from R_{i-2} and R_{i-1} using methods such as breadth-first search (BFS) in linear time, or Dijkstra's single-source shortest-paths algorithm in $O(|V| + |E| \log |V|)$ time [20]. Clearly

$$R_0 \supseteq R_2 \supseteq R_4 \supseteq \ldots$$

and

$$R_1 \supseteq R_3 \supseteq R_5 \supseteq \ldots,$$

and so R_i is consistent for all $i \geq 0$. These subsets can be used to build a time series $S^{(k)} = (S_0, S_1, S_2, \ldots, S_k)$ where
- $S_0 = I$;
- $S_1 = R_0$;
- $S_i = R_{i-1} \cup (S_{i-1} \setminus \overline{R_{i-1}})$ for all $1 < i \leq k$.

The arguments for the validity of $S^{(2)}$ can also be applied to show the validity of $S^{(k)}$, for all $k \geq 2$.

E. Variation of Time Series via Delay

Even if a twinned topos happens to support a sought-after temporal property, there is of course no guarantee that a single valid time series will contain an example of that property. In practice, testing a variety of time series improves the chances of finding one containing the desired example. Here, we discuss a strategy for increasing the variety of available valid time series, in which a set of nodes is identified that can be safely delayed to a later time step.

As it partitions the reachable nodes into sets R_0 and R_1, the SP2 algorithm always places nodes into R_0 whenever the placement would not lead to an inconsistency — a node is assigned to R_1 only if its twin has already been assigned to R_0. None of the nodes of R_1 could be reassigned to R_0 without generating an inconsistency. On the other hand, there may be elements of R_0 that are guaranteed never to generate consistency or reachability violations if delayed to later time steps.

Consider, as an example, the maximal set of nodes $L_0 \subseteq R_0$ satisfying the following two conditions:
- if $u \in L_0$, then its twin $\overline{u} \notin R$;
- if $u \in L_0$, and v is a descendant of u in F, then neither v nor $\overline{v}$ is in R_1 (note that $v \in R_0$ would also imply that $v \in L_0$).

Clearly, the nodes of L_0 are consistent with any other reachable nodes. Moreover, the two conditions together ensure that the reachability of

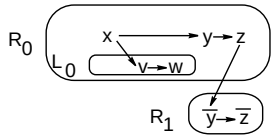

Fig. 7. R_0, R_1 and L_0 from Figure 6, generated using $\{x\}$ as the input set.

any node in $R \setminus L_0$ does not depend on any node of L_0. Figure 7 shows the classification into R_0, R_1 and L_0 of those nodes of the twinned topos of Figure 6 that are reachable from the input set $I = \{x\}$.

The nodes of L_0 can be delayed until an arbitrarily-chosen time step to yield a valid time series $S^{(k)}(j) = (S_0, S_1, S_2, \ldots, S_k)$, with $1 < j \leq k$, defined as follows:

- $S_0 = I$;
- $S_1 = R_0 \setminus L_0$;
- $S_i = R_{i-1} \cup (S_{i-1} \setminus \overline{R_{i-1}})$ for all $2 < i < j$;
- $S_j = L_0$;
- $S_{j+1} = R_{j-1} \cup (S_{j-1} \setminus \overline{R_{j-1}})$, if $j < k$;
- $S_i = R_{i-1} \cup (S_{i-1} \setminus \overline{R_{i-1}})$ for all $j + 1 < i \leq k$.

Using a given spanning forest and initial node partition, many other valid time series could conceivably be generated based on the delay of nodes of L_0 to differing time steps, rather than to a common time step as described above.

F. Variation of Time Series via Coaxing

In the time series generated using the methods described above, the initial node partition into R_0 and R_1 has a great impact on which nodes are placed at a given time step. Changing the partition can thus promote whole new classes of examples, and extinguish others. The SP2 algorithm allows the user to influence this initial partition, by means of the edge weights assigned to the topos graph. As described in more detail in the Appendix, for every reachable twinned node pair $\{v, \overline{v}\}$, SP2 places the node closer to the inputs into R_0, and the node farther from the inputs into R_1. If the edge weights are adjusted so that the order of the two shortest path lengths becomes reversed, then a call to SP2 with the new edge weights would swap the placements of the nodes in the initial partition. We refer to the process of reassigning nodes by means of edge weight adjustments as *coaxing*.

There are two situations in which one might want to use coaxing. First, a failure to generate an example of a desired temporal property may indicate a node swap that could, if successful, produce a time series containing examples that are closer to the target. Second, while the twinning conditions allow SP2 to compute partitions correctly and efficiently, the reachability paths for examples based on these partitions may make use of forbidden topos nodes. By coaxing these forbidden nodes towards the leaves of the shortest path forest F, one can increase the possibility of reaching the desired variable states of an example without encountering the forbidden nodes.

Both these situations arise with the twinned topos associated with Figure 2, when enforcing the special conditions required by the and-nodes of the original topos. Any time series making use of the and-node *and*1 at time step t must have its antecedents *motives*, *means*, *opportunity* and *witness* present at time step $t - 1$, and its consequent *guilty* present at time step t. If one or more or the antecedents do not appear at time $t - 1$, coaxing can be used to encourage them to move from one side of the SP2 node partition to the other, thereby changing the time steps in which they can be found. Similarly, the consequent or even the and-node itself can be influenced to move. Furthermore, the node twin $\overline{and1}$ is essentially meaningless, and must be avoided by the reachability paths of any valid examples.

Whatever the motivation for coaxing, the method is essentially the same. When generating the spanning forest F, it is possible to identify nodes we want to *avoid* and nodes we need to *find* on the pathways to nodes in examples. Increasing the weights of certain edges will push nodes back towards the leaves of F, allowing other nodes to be promoted in their place. The adjustments must be made with care. First, when changing weights, the twinning properties must be maintained. That is, if we increase the weight of edge e by the amount $\delta > 0$, then we must also add δ to the weight of $\overline{e}$. Second, there is little point in coaxing nodes that are descendants in F of other nodes that are being coaxed. Third, there may be many edges along the unique path of F from the input nodes to the node v to be coaxed. Ideally, to limit the impact of the weight change to as few nodes as possible, we should adjust only the weight on the path edge that is incident to to v (as well as the twin of this edge). In §A-E of the Appendix, we present the coaxing heuristics in more detail.

G. RAPTURE

There are many possible strategies for coaxing unwanted nodes off the paths towards desired nodes. The choices of coaxing rules should ideally be tailored to the kinds of examples sought. However, provided that one can identify the earliest time step at which the violation of the desired example can be determined, and the nodes whose presence or absence is the cause, a promising coaxing operation can be decided upon. The FSA used to verify the example can generally be adapted to report the cause or causes of a violation. If the violation is due to the presence of a node, that node can be coaxed away (that is, towards the leaves of the shortest path forest); if due to the absence of a node, the twin of the node can be coaxed away.

Figure 8 illustrates a simple heuristic, called RAPTURE, that repeatedly uses SP2 and coaxing to generate and evaluate time sequences. The number of iterations M is a parameter decided upon by the user; each iteration can be performed in $O(K|V| + |E|(K + \log|V|))$ time, where K is the maximum number of steps of any time series generated throughout the course of the execution. A contribution of $O(K(|V| + |E|))$ to the complexity comes from the repeated computation of reachable nodes in Step 2d, using standard graph search techniques such as depth-first search or breadth-first search. As mentioned earlier, the call to SP2 requires $O(|V| + |E| \log|V|)$ time. The cost of applying the FSA to the time series (Steps 2k-2m) can be performed in $O(K|V|)$ time assuming that the size of the FSA is considered to be bounded by some constant. Once the target node has been identified, coaxing it away (Step 2o) is a simple matter of adjusting the weights of a constant number of edges, and thus requires only constant time per iteration.

RAPTURE should be viewed as just one possible way to apply our techniques. Clearly, many variants are possible. For example, RAPTURE only coaxes one vertex at a time and makes no use of L_0 delays. Variants on RAPTURE could explore (e.g.) coaxing multiple vertices.

H. Experiments

H.1 Experiments with Coaxing

A major pre-experimental concern was that the nearly linear-time processing of SP2 could be embedded in a coaxing process that terminated only after a very long number of iterations. After much experimentation, we can report that we have never seen this worst-case behavior in practice: either the target temporal property was verified after a small number of iterations, or it could not be verified even after a very large number of iterations. In the large set of experimental studies described in [18], randomly generated topos graphs were probed for randomly selected properties. In those experiments:

- The topos size was fixed to 10,000 nodes;
- The topos fanout was varied from 2 to 6 edges per node;
- The frequency of conjunctions (and-nodes) was varied from 5% to 75% of the total number of nodes;

```
function RAPTURE (inputs: I; topos: G_O; limit: M; FSA: T): time series S
1.    G ← TWIN (G_O)
2.    repeat M times
2a.       R_2 ← ∅
2b.       ⟨F, R_0, R_1⟩ ← SP2 (G,I)
2c.       k ← 2
2d.       while (R_k ≠ R_{k-2} and R_{k-1} ≠ ∅)
2d1.          R_k ← nodes of G reachable from R_{k-1}
2d2.          R_k ← R_k ∩ R_{k-2}
2d3.          k ← k + 1
2d4.          R_k ← ∅
          end while
2e.       k ← k − 1
2f.       S_0 ← I
2g.       S_1 ← R_0
2h.       for i ← 1 to k do
2i.           S_{i+1} ← R_i ∪ R_{i-1} \ ¬R_i
          end for
2j.       S ← (S_0, S_1, ..., S_k)
2k.       if T accepts S then
2k1.          return S
          end if
2l.       P ← set of nodes whose presence caused the rejection of S by T
2m.       Q ← set of nodes whose absence caused the rejection of S by T
2n.       v ← a node of P ∪ ¬Q with minimum path length to I in F
2o.       G ← COAX (v)
      end repeat
3.    return null
end RAPTURE
```

Fig. 8. The RAPTURE heuristic.

	SPIN	SP2	Number of Cases
	??	found	21
	??	not found	8
	found	found	11
	not found	not found	0
	found	not found	0
	not found	found	0
RAM used (max)	100MB	< 1MB	

Fig. 9. Finding properties of Figure 4 using SPIN and SP2. "??" denotes that SPIN did not terminate in 100MB of RAM.

- The results after 10 coaxing iterations were compared to the results after 100 iterations.

In all cases, the coverage of the goals reached a plateau after at most five iterations of SP2 plus coaxing. Also, the plateau reached after 10 coaxes differed little from that reached after up to 100 coaxes. Furthermore, SP2 never used more than 1MB of memory or one minute of runtime. Our conclusion from these experiments is that the use of heuristic coaxing in RAPTURE does not diminish the time or space efficiency of SP2 in either the successful or unsuccessful cases.

H.2 Comparisons with SPIN

Figure 9 shows a comparison of SPIN versus SP2 using properties of the general form $\Diamond X$ ("eventually X") and the systems model of Figure 4. Of the 40 properties which were analyzed by both SPIN and SP2, SPIN was able to return a verification result in only 11 out of 40 cases (27.5%) before running out of memory. In every case where SPIN did return a verification result, SP2's result was in agreement.

Regarding computer resources, SP2 used less than 1% of the RAM required by SPIN. Also, in the case of the properties that were not found, SP2 terminated in less than one second of CPU time while SPIN took much longer.

We mentioned earlier that one pre-experimental concern with informal topoi is that they are so under-defined that we could use them to infer any set of properties at all. Figure 9 shows that this is not always true. In the case of 8 of the 40 properties, SP2 could not find them across the large under-defined topos of Figure 4.

IV. Discussion

A. History

Historically, this work is based on Feldman & Compton's study of the validation of topoi [14] (which they called *qualitative compartmental models*). Menzies conjectured that this inference problem is NP-hard, and attempted a heuristic optimization of the validation process. The resulting procedure, HT4, ran orders of magnitude faster than the validation engine built by Feldman & Compton. However, HT4 still suffered from exponential runtimes [7, 16, 21].

That prior work also made the restrictive assumption that topoi were treated as propositional theories, that is, with each variable receiving a single, precise assignment. This assumption was made after observing that experts only ever used topoi to generate acyclic explanations. Cyclic explanations introduce a temporal aspect to variable assignment; that is, if variables cannot be assigned contradictory values, then the explanation

$$a\uparrow \rightarrow b\uparrow \rightarrow a\downarrow$$

is only sensible when $\{a\uparrow, a\downarrow\}$ occur at different times. Cyclic explanations are required when adding temporal simulation properties to qualitative compartmental models. Assuming a certain restriction on topos edge types, Cohen, Menzies, Waugh & Goss showed that the cost of checking temporal properties of topos-based simulations is a function of the number of time-ticks in the query [22].

B. Lightweight Formal Reasoning for Requirements

Other researchers have explored formal methods for requirements engineering. For example, in the KAOS system [23], analysts generate a properties model by incrementally augmenting object-oriented scenario diagrams with temporal logic statements. Potentially, this research reduces the costs of testing requirements by integrating the generation of the properties model into the rest of the system development. Our reading of the KAOS work is that while the resulting model may be more formal, the level of skill required to write the temporal logic can significantly increase the personnel cost. Further, the extra time required for the augmentation could add to the development brake.

In other work, Schneider et al. [3] explored reducing the manual modelling costs using *lightweight formal methods*. In the lightweight approach, only partial descriptions of the systems and properties models were constructed using the SPIN formal analysis tool [17]. Despite their incomplete nature, Schneider et al. found that such partial models could still detect significant systems errors. While exciting research, this approach still incurs a personnel cost, as scarce expertise is required to drive tools such as SPIN.

C. Representing Temporal Options

In our research, we explore the temporal properties of a topos graph $G\langle V, N\rangle$, whose nodes can be associated with disjunctions (the default) or conjunctions (special and-nodes). A disjunction node $v \in V$ can be reached if any antecedent nodes $u \in V$ can be reached, where u is an antecedent of v if $(u, v) \in E$. On the other hand, a conjunction node can be reached only if all its antecedents can be simultaneously reached.

Two alternate representations for temporal properties are a *flowgraph* G^f and a Clarke-style *state space* G^s. These alternative methods are discussed below.

C.1 Flowgraphs

In a flow graph $G^f\langle V^f, E^f\rangle$, values at a node $v \in V^f$ are the result of a transformation performed during the traversal of an edge upstream from that node. Our original topos graphs reduce to simple flowgraphs in the case when V contains no conjunctions. Methods for exploring the temporal properties of flowgraphs have been extensively explored. For example, Olender & Osterweil's CECIL system [24] uses Tarjan's

path solving algorithms [25] to condense the flowgraph to a regular expression summarizing the whole graph. This reduced space is then explored for violations to system constraints.

For another example of flowgraph-based work, Corbett [26] mapped FSAs to flowgraphs from some start state to some final state where the flow through some arc i is represented by an integer variable x_i. Assuming that in-flows to a state must equal the out-flows from that state, flow equations can be generated. As shown by Corbett, solutions to such a system of flow equations can be sought for using integer programming (IP) techniques. The verification examples in a twinned topos can also be formulated as an IP problem; however, the RAPTURE method can be viewed as an iterative-search optimization heuristic for this IP problem that makes use of an exact solution (SP2) to a closely-related problem.

C.2 State-space Search

In state space $G^s\langle V^s, E^s\rangle$, each $v \in V^s$ contains consistent value assignments to all program variables, and each $e \in E^s$ is a state change permitted by the program. Clarke's classic model checker (MC) performs a linear-time search across a state space to find a set of states that lead to a violation of some desired temporal property [27]. If the MC search terminates, then the search path is a counter-example showing how that program can fail. While the search takes linear time in terms of the size of the state space, this size is exponential on the number of assignments per variable. Elaborate techniques have been developed in an attempt to limit the state space explosion. A sample of these techniques are described below.

- *Abstraction or partial ordering*: use only the part of the space required for a particular counter-example. Implementations exploiting this technique can constrain how the space is traversed [28], or constructed in the first place [3].
- *Clustering*: divide the systems model into sub-systems which can be reasoned about separately [29].
- *Meta-knowledge*: study only succinct meta-knowledge of the space. One example used an eigenvector analysis of the long-term properties of the systems model under study [30].
- *Exploit symmetry*: find properties in some part of the systems model, then reuse those counter-examples if ever those parts are found elsewhere in the systems model [31].
- *Semantic minimization*: replace the space with some smaller, equivalent space. For example, the BANDERA system [32] reduces both the systems modelling cost and the execution cost by automatically extracting (slicing) the minimum portions of a JAVA program's bytecodes which are relevant to particular properties models.

While the above techniques have all been useful in their test domains, they may not be universally applicable. Certain optimizations, such as those of [30], require expensive pre-processing. Also, these methods may rely on certain combinatorial features of the system being studied. Exploiting symmetry is only useful if the system under study is highly symmetric. Clustering generally fails for tightly connected models. Further, for requirements engineering, systems like BANDERA are not suitable. BANDERA only works on already-implemented systems; that is, not until long after the requirements phase has ended.

Hence, in the general case, it seems that only small models can be tested using state-space search techniques. Further, these models must be precisely specified. In contrast, this work describes methods for quickly finding properties in large, loosely-specified models.

Our results to date are promising. However, an open issue with the techniques is their adaptability to other models with more complex temporal properties. Currently, we are working to explore a wider range of models and a wider range of temporal properties.

ACKNOWLEDGEMENTS

This paper benefitted greatly from the insightful comments of the anonymous reviewers.

REFERENCES

[1] T. Menzies, J. Powell, and M. E. Houle, "Fast formal analysis of requirements via 'topoi diagrams'," in *ICSE 2001*, 2001, Available from `http://tim.menzies.com/pdf/00fastre.pdf`.

[2] D.J. Reifer, "Software failure modes and effects analysis," *IEEE Transactions on Reliability*, pp. 247–249, 1979.

[3] F. Schneider, S.M. Easterbrook, J.R. Callahan, G.J. Holzmann, W.K. Reinholtz, A. Ko, and M. Shahabuddin, "Validating requirements for fault tolerant systems using model checking," in *3rd IEEE International Conference On Requirements Engineering*, 1998.

[4] R. Dieng, O. Corby, and S. Lapalut, "Acquisition and exploitation of gradual knowledge," *International Journal of Human-Computer Studies*, vol. 42, pp. 465–499, 1995.

[5] M. B. Dwyer, G. S. Avrunin, and J.C. Corbett, "A system specification of patterns," `http://www.cis.ksu.edu/santos/spec-patterns/`, 1997.

[6] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, "Patterns in property specifications for finite-state verification," in *ICSE98: Proceedings of the 21st International Conference on Software Engineering*, May 1998.

[7] T.J. Menzies, *Principles for Generalised Testing of Knowledge Bases*, Ph.D. thesis, University of New South Wales, 1995, Ph.D. thesis. Available from `http://tim.menzies.com/pdf/95thesis.pdf`.

[8] S. Buckingham Shum and N. Hammond, "Argumentation-based design rationale: What use at what cost?," *International Journal of Human-Computer Studies*, vol. 40, no. 4, pp. 603–652, 1994.

[9] G.A. Smythe, "Hypothalamic noradrenergic activation of stress-induced adrenocorticotropin (ACTH) release: Effects of acute and chronic dexamethasone pre-treatment in the rat.," *Exp. Clin. Endocrinol. (Life Sci. Adv.)*, pp. 141–144, 6 1987.

[10] J. Mylopoulos, L. Cheng, and E. Yu, "From object-oriented to goal-oriented requirements analysis," *Communications of the ACM*, vol. 42, no. 1, pp. 31–37, January 1999.

[11] A. MacLean, R.M. Young, V. Bellotti, and T.P. Moran, "Questions, options and criteria: Elements of design space analysis," in *Design Rationale: Concepts, Techniques, and Use*, T.P. Moran and J.M. Carroll, Eds., pp. 53–106. Lawerence Erlbaum Associates, 1996.

[12] B. Kuipers, "Qualitative simulation," *Artificial Intelligence*, vol. 29, pp. 229–338, 1986.

[13] G.A. Smythe, "Brain-hypothalmus, Pituitary and the Endocrine Pancreas," *The Endocrine Pancreas*, 1989.

[14] B. Feldman, P. Compton, and G. Smythe, "Towards Hypothesis Testing: JUSTIN, Prototype System Using Justification in Context," in *Proceedings of the Joint Australian Conference on Artificial Intelligence, AI '89*, 1989, pp. 319–331.

[15] T.J. Menzies and P. Compton, "Applications of abduction: Hypothesis testing of neuroendocrinological qualitative compartmental models," *Artificial Intelligence in Medicine*, vol. 10, pp. 145–175, 1997, Available from `http://tim.menzies.com/pdf/96aim.pdf`.

[16] T.J. Menzies, "On the practicality of abductive validation," in *ECAI '96*, 1996, Available from `http:/tim.menzies.com/pdf/96ok.pdf`.

[17] G.J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.

[18] J.D. Powell, "The rapture/sp2 approach to model checking: An explanation and viability experimentation," 1999.

[19] R. Gerth, D. Peled, M. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *Proc. PSTV 1995 Conference, Warsaw, Poland*, 1995, Available from `http://cm.bell-labs.com/cm/cs/who/gerard/gz/ltl.pdf`.

[20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[21] T.J. Menzies, "Applications of abduction: Knowledge level modeling," *International Journal of Human Computer Studies*, vol. 45, pp. 305–355, 1996, Available from `http://tim.menzies.com/pdf/96abkl.pdf`.

[22] T.J. Menzies, R.F. Cohen, S. Waugh, and S. Goss, "Applications of abduction: Testing very long qualitative simulations," *IEEE Transactions of Data and Knowledge Engineering (accepted for publication, 2000)*, 2003, Available from `http://tim.menzies.com/pdf/97iedge.pdf`.

[23] A. van Lamsweerde and L. Willemet, "Inferring declarative requirements specifications from operational scenarios," *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, November 1998.

[24] K.M. Olender and L.J. Osterweil, "Interprocedural static analysis of sequencing constraints," *TOSEM*, vol. 1, no. 2, pp. 21–52, 1992.

[25] R.E. Tarjan, "Fast algorithms for solving path problems," *Journal of the Association for Computing Machinery*, vol. 28, no. 3, pp. 594–614, July 1981.

[26] J. Corbett, "An empirical evaluation of three methods for deadlock analysis of ada tasking programs," in *Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, 1994.

[27] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, April 1986.

[28] P. Godefroid, "On the costs and benefits of using partial-order methods for the verificiation of concurrent systems (invited papers)," in *The 1996 DIMACS workshop on Partial Order Methods in Verificaition, July 24-26, 1996*, 1997, pp. 289–303.

[29] E.M. Clark and D. E. Long, "Compositional model checking," in *Fourth Annual Symposium on Logic in Computer Science*, 1989.

[30] Y. Ishida, "Using global properties for qualitative reasoning: A qualitative system theory," in *Proceedings of IJCAI '89*, 1989, pp. 1174–1179.

[31] E.M. Clark and T. Filkorn, "Exploiting symmetry in temporal logic model checking," in *Fifth International Conference on Computer Aided Verification*. 1993, Springer-Verlag.

[32] J. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasarenu, Robby, and H. Zheng, "Bandera: Extracting finite-state models from java source code," in *Proceedings ICSE2000, Limerick, Ireland*, 2000, pp. 439–448.

[33] T.E. Cormen, C. E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990, ISBN: 0262031418.
[34] M. Lowrey, M. Boyd, and D. Kulkarni, "Towards a theory for integration of mathematical verification and empirical testing," in *Proceedings, ASE'98: Automated Software Engineering*, 1998, pp. 322–331.
[35] Y. Iwasaki, "Qualitative physics," in *The Handbook of Artificial Intelligence*, P.R. Cohen A. Barr and E.A. Feigenbaum, Eds., vol. 4, pp. 323–413. Addison Wesley, 1989.
[36] Y. Iwasaki and H.A. Simon, "Causality in device behaviour," *Artificial Intelligence*, vol. 29, pp. 3–31, 1986.
[37] Y. Iwasaki, "Causal ordering in a mixed structure," in *Proceedings of AAAI '88*, 1988, pp. 313–318.
[38] N. Leveson, *Safeware System Safety And Computers*, Addison-Wesley, 1995.

APPENDIX

I. PROOF OF CORRECTNESS FOR ALGORITHM SP2

A. Main Result

Consider a path $\pi = (p_0, p_1, \ldots, p_m)$ in the twinned topos $G\langle V, E\rangle$, originating at $p_0 \in V$ and terminating at $p_m \in V$. The length of π, denoted $\lambda(\pi)$, is the sum of the lengths of its edges, $\sum_i^m \lambda((p_{i-1}, p_i))$. If $m = 0$, the length of the path shall be deemed to be zero. We will restrict our attention to paths which are *simple*; that is, those in which no node appears more than once.

Any path π achieving the minimum length over all paths from p_0 to p_m shall be called a *shortest path* from p_0 to p_m in G; alternatively, the *distance* from p_0 to p_m is the length of any shortest path between them. This terminology can be extended to sets of nodes in the following manner: if X and Y are sets of nodes, then a path π is a shortest path from X to Y if $p_0 \in X$, $p_m \in Y$, and π has length no greater than any other path originating at a node in X and terminating at a node in Y. Shortest paths from sets to a single node, and from a single node to a set, can be defined analogously.

Path π may or may not be consistent. If not, then there exists a value k between 1 and the number of edges of the path such that the deletion of k edges of π yields $k + 1$ consistent subpaths. If k is the minimum such value, we say that the path is *k-inconsistent*. The set of k edges deleted shall be referred to as a *cut set* of the path, each edge of which will be called a *cut edge*.

In this Appendix, we prove the following result:

Theorem 1: Let $G\langle V, E\rangle$ be a twinned topos, and let I be a consistent subset of V. Let R be the set of all nodes of V reachable from some node of I in G. Then there exists a subgraph $F\langle R, E_R\rangle$ of G such that:

1. For every $v \in R$, exactly one shortest path $\sigma(v)$ from I to v in G is also contained in F.
2. F is the union of the paths $\sigma(v)$ taken over all $v \in R$.
3. Every path $\sigma(v)$ is either consistent or 1-inconsistent.
4. There exists a set of cut edges resolving all 1-inconsistent paths $\sigma(v)$, such that no shortest path $\sigma(v)$ contains more than one cut edge.
5. The cut edges define a partition of R into two disjoint node sets R_0 and R_1 such that
   - $I \subseteq R_0$,
   - R_0 and R_1 are both consistent, and
   - edge (u, v) is a cut edge if and only if $u \in R_0$ and $v \in R_1$.

A subgraph F satisfying the conditions of the theorem shall be referred to as the *reachability* graph for I in G. We shall also show that reachability graphs can be computed in $O(|V| + |E| \log |V|)$ time in general, and in $O(|V| + |E|)$ time when all edge lengths are identical.

B. Paths

Each cut edge $e = (p_i, p_{i+1})$ can be associated with the node pairs it separates. If p_a and p_b are nodes of the path such that $a \leq i$ and $b \geq i + 1$ and $p_a = \overline{p_b}$, then the deletion of any edge of π on the path from p_a to p_b (including e) ensures that no subpath of π contains both p_a and p_b. The edges on the path from p_a to p_b form a *cut interval* associated with e. The intersection of all cut intervals associated with edge e will be called the *cut intersection* $\mathcal{I}(e)$ of e.

Since the number of cut intersections is k, and cannot be reduced, no two cut intersections may overlap (otherwise, deletion of an edge in the overlap resolves all cut intervals of both cut intersections, contradicting the minimality of k).

Observation 2: Let e and f be distinct cut edges drawn from the same cut set of path π in G. Then their associated cut intersections $\mathcal{I}(e)$ and $\mathcal{I}(f)$ must share no edges.

Lemma 3: Let x and y be nodes of G such that y is reachable from x. Then
- $\overline{y}$ is reachable from $\overline{x}$, and
- the distance from x to y is equal to the distance from $\overline{x}$ to $\overline{y}$.

Proof Let $\pi = (p_0, p_1, \ldots, p_m)$ be a shortest path from $p_0 = x$ to $p_m = y$. Since the edge (p_i, p_{i+1}) is in G for all $0 \leq i < m$, the edge $(\overline{p_i}, \overline{p_{i+1}})$ must be in G as well. This implies that the path $\overline{\pi} = (\overline{p_0}, \overline{p_1}, \ldots, \overline{p_m})$ exists from $\overline{p_0} = \overline{x}$ to $\overline{p_m} = \overline{y}$.

The paths π and $\overline{\pi}$ are both of length $\lambda(\pi)$. $\overline{\pi}$ must be a shortest path from $\overline{x}$ to $\overline{y}$; otherwise, if a shorter path existed from $\overline{x}$ to $\overline{y}$, a path of the same length would exist from x to y, contradicting the assumption that π was shortest. □

Lemma 4: If t is reachable from s in G, then any shortest path from s to t is either consistent or 1-inconsistent.

Proof By contradiction: let us assume that every path from s to t is neither consistent nor 1-inconsistent. Then consider a shortest path $\pi = (p_0, p_1, \ldots, p_m)$ from $p_0 = s$ to $p_m = t$ in G. Let λ_i be the length of the subpath $\pi_i = (p_0, \ldots, p_i)$ of π, for all $0 \leq i \leq m$.

Path π must be k-inconsistent, for some $k \geq 2$. Let $I_1 = (p_a, p_b)$ and $I_2 = (p_c, p_d)$ be two cut intersections associated with the cut set of π. By Observation 2, I_1 and I_2 must be disjoint, and thus without loss of generality we may assume that $0 \leq a < b \leq c < d \leq m$.

Since the endpoints of cut intersections derive from cut intervals, p_b must be separated from its negation $p_{b'} = \overline{p_b}$ by I_1. This implies that $b' \leq a < b$. Similarly, node p_c must be separated from $p_{c'} = \overline{p_d}$ by I_2, where $c < d \leq c'$.

Lemma 3 implies that the distance from $p_{b'}$ to $p_{c'}$ is the same as the distance from p_b to p_c — that is, $\lambda_c - \lambda_b$. Since the path from p_0 to $p_{b'}$ is of length $\lambda_{b'}$, and the path from $p_{c'}$ to p_m is of length $\lambda_m - \lambda_{c'}$, there exists a path from p_0 to p_m of length

$$\lambda_{b'} + (\lambda_c - \lambda_b) + (\lambda_m - \lambda_{c'}) = \lambda_m - (\lambda_{c'} - \lambda_c) - (\lambda_b - \lambda_{b'}) < \lambda_m.$$

This contradicts the assumption that the distance from s to t was λ_m. □

C. Shortest Path Trees

The proof of the main theorem is by construction. Let I be a consistent subset of V. Consider now an augmentation of the graph $G\langle V, E\rangle$ by two artificial nodes r and $\overline{r}$, and a pair of artificial edges (r, s) and $(\overline{r}, \overline{s})$ for each $s \in I$. The length of each artificial edge is deemed to be $\varepsilon > 0$, an arbitrary positive constant. The resultant graph — call it $G^*\langle V^*, E^*\rangle$ — remains a twinned topos.

Within the graph G^*, a variant of Dijkstra's single-source shortest path algorithm [20] is used to construct a tree of shortest paths from r to all nodes reachable from r; a pseudocode description of the algorithm is shown in Figure 10. Since the only neighbours of r are the nodes of I, the tree would also contain shortest paths to all nodes reachable from I. These paths together constitute a shortest path forest in G.

As with Dijkstra's original algorithm, SP2 grows the shortest path tree F^* from the artificial root in greedy fashion, ensuring that the current tree is always a shortest path tree over the nodes it spans. A priority queue *PQ* is used to keep track of candidate edges for inclusion. Each entry in the queue consists of:
- an edge (u, v), where v is the candidate node and u is a node already in the tree;

```
function SP2 (twinned topos G⟨V, E⟩; inputs I):
        forest F; node sets R0, R1; cut edges C
0.  G* ← G augmented with artificial nodes r, r̄
1.  for each v ∈ V do
        D[v] ← ∞  {all nodes are unvisited}
        C[v] ← ∅  {no node is below a cut edge}
    end for
2.  PQ ← F* ← ∅
3.  R0 ← {r}
4.  R1 ← ∅
5.  for each initial node s ∈ I do
        PQ ← PQ ∪ {(0, (r, s))}
    end for
6.  while PQ is not empty do
6a.     (d, (u, v)) ← deletemin(PQ)
6b.     if D[v] ≠ ∞ then
            go to next iteration of the while loop
        end if
6c.     if d = D[v̄] and v̄ ∈ R1 then
            {let x̄ be the parent of v̄ in F*}
            if u ≠ x then
                go to next iteration of the while loop
            end if
        end if
6d.     D[v] ← d  {v becomes visited}
6e1.    if u ∈ R0 and v̄ ∉ R0 then
            R0 ← R0 ∪ {v}
6e2.    else
            R1 ← R1 ∪ {v}
            if u ∈ R0 then
                C[v] ← (u, v)
            else
                C[v] ← C[u]
            end if
        end if
6f.     F* ← F* ∪ {(u, v)}
6g.     for each w such that (v, w) ∈ E do
            PQ ← PQ ∪ {(D[v] + λ((v, w)), (v, w))}
        end for
    end while
7.  R0 ← R0 \ {r}
8.  F ← F* \ {(r, s)| s ∈ I}
9.  return F, R0, R1, C
end SP2
```

Fig. 10. The SP2 algorithm

- a distance d, the length of the shortest path from the root to v passing through u.

The entries in the queue are prioritized according to their values of d, with small values being given the highest priority. Although not strictly necessary for Theorem 1 to hold, we will later require *PQ* to be *stable*, in that two entries having the same distance d should emerge from *PQ* in the same order in which they were inserted. Once a node v is inserted into the tree, its distance $D[v]$ from r will have been calculated and stored.

As nodes are added to the shortest path tree, they are assigned to one of the two sets R_0 and R_1. Whenever possible, the new node (call it v) is added to set R_0. Only in those situations where an assignment to R_0 would result in an inconsistency (that is, when $\overline{v}$ already belongs to R_0), or in a path with more than one cut edge, is an assignment of v to R_1 considered. If the edge (u, v) is inserted, with $u \in R_0$ and $v \in R_1$, the transition from R_0 to R_1 is noted as the cut edge $C[v]$ above v. All subsequent descendents w below v will also inherit $C[w] = (u, v)$ as the cut edge above it.

In certain conditions, SP2 will choose to defer the insertion of node v altogether. It shall be shown in Lemma 5 that whenever the algorithm defers the insertion (Step 6c of Figure 10), a path of equal length will eventually be found to v passing through another node. Step 6c is the only point of difference in how Dijkstra's algorithm and Algorithm SP2 handle the insertion of nodes into the shortest path tree.

If *PQ* is implemented using a heap, the worst-case time of the algorithm is in $O(|V| + |E| \log |V|)$. As is the case with the many variants of Dijkstra's algorithm, data structures such as two-dimensional lookup tables or Fibonacci heaps can also be used to organize the insertion of nodes into the tree, with varying time complexities (see [33] for details). However, it is worth mentioning that some distributions of edge length (such as when all edges have identical length) allow for the use of an ordinary FIFO (First-In-First-Out) queue, resulting in an overall worst-case time in $O(|E|)$.

Lemma 5: Given a twinned topos $G\langle V, E\rangle$ and a consistent input set $I \subseteq V$:

1. Algorithm SP2 correctly creates a shortest path forest F of all nodes reachable from I in G.
2. The partition sets R_0 and R_1 are both consistent.

Proof The only difference in the construction of the shortest path tree between Algorithm SP2 and Dijkstra's algorithm is in the rejection of certain edges that is carried out in Step 6c. Provided that the nodes are added to the tree in non-decreasing order of their distances from r, the arguments that show the correctness of Dijksta's algorithm apply to SP2 as well. To show that Algorithm SP2 constructs a shortest path tree F^*, one need only show that whenever an edge (u, v) is rejected at Step 6c, node v will be attached by means of another edge without changing its distance from r.

The argument that R_0 and R_1 are consistent is by induction: we assume that after each of the previous insertions of a node w into F^*,

- F^* was a correct shortest path tree for the nodes it spanned,
- all nodes having distances to r strictly less than $D[w]$ have been included in the tree.
- the current sets R_0 and R_1 were both consistent.

Initially, the tree consists of the single artificial root node r, for which the lemma holds.

Assume now that the edge (u, v) is considered for inclusion. If v is already reachable using F^*, the edge will be discarded at Step 6b. Otherwise, the execution continues at Step 6c.

If $\overline{v}$ is not already in the tree, then v is added to the tree in Steps 6d through 6f. Node v is assigned to the same node set (R_0 or R_1) as its parent u.

If $\overline{v}$ is already in the tree, then it must have been assigned to either R_0 or R_1. If $\overline{v} \in R_0$, then v will be correctly placed in R_1 at Step 6e2, regardless of the status of u. If $\overline{v} \in R_1$, then v cannot be placed in R_1 without violating the consistency of R_1. Step 6e appears to allow this to happen in the case where $\overline{v} \in R_1$ and $u \in R_1$, and $D[\overline{v}] < D[v]$. If $D[\overline{v}] = D[v]$ instead, then the edge is rejected at Step 6c unless $u = x$ — control never reaches Step 6e for the edge (u, v) unless $(\overline{u}, \overline{v})$ is in F^*. However, we will now show that the former situation never arises, and that rejecting the edge in the latter situation is always safe, whenever $\overline{v} \in R_1$.

If $u \in R_1$ as well as $\overline{v} \in R_1$, then there must be a cut edge $(\overline{p_{j-1}}, \overline{p_j})$ along the path $\sigma(\overline{v}) = (\overline{p_0}, \overline{p_1}, \ldots, \overline{p_m})$, where $\overline{p_0} = r$ and $\overline{p_m} = \overline{v}$. For $(\overline{p_{j-1}}, \overline{p_j})$ to be a cut edge, p_j must have been added to the shortest path tree before $\overline{p_j}$. The induction hypothesis therefore implies that $D[p_j] \leq D[\overline{p_j}]$. By the properties of twinned topoi, there must be a path from p_j to v of the same length as that from $\overline{p_j}$ to $\overline{v}$. This would imply the existence of a path from r to v of length no more than $D[\overline{v}]$.

If $D[\overline{v}] < D[v]$, we have a contradiction, as we have found a path from r to v of length strictly less than $D[v]$. It remains to be shown that if $D[\overline{v}] = D[v]$, any rejection of (u, v) that occurs in Step 6c is safe. We shall do this by showing that at least one opportunity exists for connecting v without violating the consistency of R_0 and R_1. In particular, we shall show that v can be attached to $p_{m-1} = x$.

Consider now the path $\sigma'(v)$ formed by $\sigma(p_{m-1})$ followed by the

edge (p_{m-1}, v).

$$D[\overline{p_{m-1}}] + \lambda((\overline{p_{m-1}}, \overline{v})) = D[\overline{v}] = D[v] \\ \leq \lambda(\sigma'(v)) = D[p_{m-1}] + \lambda((p_{m-1}, v))$$

Lemma 3 implies that $\lambda((\overline{p_{m-1}}, \overline{v})) = \lambda((p_{m-1}, v))$, and therefore $D[\overline{p_{m-1}}] \leq D[p_{m-1}]$. However, we have already shown that $D[\overline{p_j}] \geq D[p_j]$, from which it follows that

$$D[\overline{p_j}] + \lambda((\overline{p_j}, \overline{p_{j+1}}, \ldots, \overline{p_{m-1}})) \\ \geq D[p_j] + \lambda((p_j, p_{j+1}, \ldots, p_{m-1}))$$

and $D[\overline{p_{m-1}}] \geq D[p_{m-1}]$. We must therefore have $D[\overline{p_{m-1}}] = D[p_{m-1}]$.

Since $D[v] = D[\overline{v}] > D[\overline{p_{m-1}}] = D[p_{m-1}]$, the length of the path from r to v through p_{m-1} is equal to $D[v]$, the shortest path length from r to v. Also, since $D[p_{m-1}] < D[v]$, p_{m-1} must already have been added to the tree by the time v is considered. In this regard at least, $p_{m-1} = x$ is a suitable candidate for attaching v to.

The node $\overline{p_{m-1}}$ must be in R_1. Otherwise, if $\overline{p_{m-1}}$ were in R_0, $\overline{p_m} = \overline{v}$ would have been added to R_0 at Step 6e1 (as v had not yet been encountered when $\overline{v}$ was added to the tree). The induction hypothesis then implies that $p_{m-1} \in R_0$. When attempting to attach v to the shortest path tree via node p_{m-1}, v would be assigned to set R_0 — consistent with the previous assignment of $\overline{v}$ to R_1. We conclude that $p_{m-1} = x$ is a suitable candidate at which to attach v.

Once the shortest path tree F^* of the augmented graph G^* has been computed, we can obtain a subgraph $F\langle R, E_R\rangle$ of the original twinned topos G simply by deleting r together with the edges leading from it. □

We are now in a position to complete the proof of the main theorem, by showing that F satisfies its conditions.

Proof of Theorem 1

1. *For every $v \in R$, exactly one shortest path $\sigma(v)$ from I to v in G is also contained in F.*

This follows from the properties of the shortest path tree F^*, and Lemma 5. Note that every path in F^* from r to another node must pass through a node of I.

2. *F is the union of the paths $\sigma(v)$ taken over all $v \in R$.*

This also follows from the properties of the shortest path forest.

3. *Every path $\sigma(v)$ is either consistent or 1-inconsistent.*

Follows from Lemma 4.

4. *There exists a set of cut edges resolving all 1-inconsistent paths $\sigma(v)$, such that no shortest path $\sigma(v)$ contains more than one cut edge.*

By the construction of F^*, no node v can be assigned to R_0 if its predecessor u in F^* has been assigned to R_1. Since the root r is assigned to R_0, there can therefore be only at most one alternation from R_0 to R_1 along any path. As (u, v) can be a cut edge only if $u \in R_0$ and $v \in R_1$, this condition is satisfied by F.

5. *The cut edges define a partition of R into two disjoint node sets R_0 and R_1 such that*

- *$I \subseteq R_0$,*
- *R_0 and R_1 are both consistent, and*
- *edge (u, v) is a cut edge if and only if $u \in R_0$ and $v \in R_1$.*

The consistency of R_0 and R_1 follows from Lemma 5. The set of cut edges satisfies the condition by construction.

To see that $I \subseteq R_0$, consider the relationship of the nodes of I with r in G^*. Root r is connected to all nodes of I, and only the nodes of I. Each node of I is strictly closer to r than any other node of $V^* \setminus (I \cup \{r\})$. This implies that the nodes of I will be considered for insertion in F^* before any of the other nodes.

Let s be a node of I. Since $\overline{s} \notin I$, s must be considered for insertion before $\overline{s}$. Since s is considered by means of the edge (r, s), and since $r \in R_0$, therefore s is assigned to R_0 at Step 6e1. This shows that $I \subset R_0$ as required.

□

Provided that every pair of edges e and $\overline{e}$ satisfies $\lambda(e) = \lambda(\overline{e})$, Algorithm SP2 is guaranteed to produce a consistent partition of V into R_0 and R_1. However, the actual sets R_0 and R_1 produced depend greatly on the initial choice of edge weights. In the next section, we will see how the freedom of choice of edge weights can be exploited when attempting to generate partitions satisfying desired criteria.

D. Preserving Paths While Coaxing

Although the goal of weight adjustment is to produce a change in an existing shortest path forest, it is important to be able to do so without changing certain desirable paths or subtrees. We begin our investigation by giving conditions in which a shortest path remains unchanged after weight adjustment and a subsequent application of SP2.

When referring to shortest paths within the graph G^*, we will use the notation $G_1^*\langle V^*, E_1^*\rangle$ to refer to the graph before the weight adjustment, and $G_2^*\langle V^*, E_2^*\rangle$ after the adjustment. The partitions of V^* produced for G_1^* and G_2^* shall be denoted $\{R_0^1, R_1^1\}$ and $\{R_0^2, R_1^2\}$, respectively. Let F_1^* be the shortest path tree produced by an application of SP2 on G_1^*, and F_2^* be the shortest path tree produced by an application of SP2 on G_2^*. For any edge $e \in E^*$, its weight before and after the adjustment shall be denoted $\lambda_1(e)$ and $\lambda_2(e)$, respectively. We will consider only adjustments for which $\lambda_2(e) = \lambda_2(\overline{e})$ for all $e, \overline{e} \in E^*$.

Lemma 6: Let A be a collection of edge pairs whose weights are to be increased between two applications of SP2 (the weights of all other edges are left unchanged). Let $\Pi_1 = (\pi_1^1, \pi_2^1, \ldots, \pi_k^1)$ be the sequence of shortest paths to v in G_1^*, in the order in which they emerge from *PQ* at Step 6a of the first application of SP2. Similarly, let $\Pi_2 = (\pi_1^2, \pi_2^2, \ldots, \pi_k^2)$ be the sequence of shortest paths to v in G_2^*, in the order in which they emerge from *PQ*. Let $\pi \in \Pi_1$ be the shortest path to v in F_1^*. If π contains no edge of A, then Π_2 is a subsequence of Π_1. Furthermore, path π is also in Π_2.

Proof Let ρ be any path from r to v in G_1^*. Since no edge of E^* has had its weight reduced, $\lambda_2(\rho) \geq \lambda_1(\rho)$. As π contains no edges of A, $\lambda_1(\rho) \geq \lambda_1(\pi) = \lambda_2(\pi)$, and therefore π must also be a shortest path in G_2^*.

Let us now assume that ρ' is also a shortest path of G_2^*. Path ρ' must not contain any edge of A; otherwise, we would have $\lambda_1(\rho) < \lambda_2(\rho) = \lambda_2(\pi) = \lambda_1(\pi)$, contradicting the assumption that π was a shortest path in G_1^*. ρ' must therefore be a shortest path of G_1^* as well. This implies that $\Pi_2 \subseteq \Pi_1$, and therefore that no path of Π_2 contains an edge of A.

Two assumptions have been made on the use and operation of the priority queue in Algorithm SP2: that entries are inserted into *PQ* in some canonical order (at Step 6g), and that *PQ* is stable. Considering only those nodes appearing on paths in Π_2, one can show by an inductive argument that the relative order in which the nodes are inserted into *PQ* is the same in both applications of SP2, as no weight adjustment has been performed on any edges of these paths. The stability of *PQ* in turn implies that the order in which the nodes emerge from *PQ* (at Step 6a) is the also the same in both applications, and the result follows. □

The previous lemma shows that the choice of a stable data structure for the priority queue can help to maintain continuity in some parts of the shortest path tree after a weight adjustment. The extent to which portions of the tree are left unchanged is made clear by the following lemma.

Lemma 7: Let A be a collection of edge pairs whose weights are to be increased between two applications of SP2 (the weights of all other edges are left unchanged). Let π be the shortest path to a node v in the tree F_1^*.

1. If $\overline{v}$ is not reachable from r and π contains no edge of A, then π also belongs to the tree F_2^*. Moreover, all nodes of π belonging to R_0^2 and R_1^2 are exactly those belonging to R_0^1 and R_1^1, respectively.
2. If $\overline{v}$ is reachable from r, let π' be the shortest path to $\overline{v}$ in F_1^*. If both π and π' contain no edge of A, then π and π' belong to F_2^* as well. Moreover, all nodes of π and π' belonging to R_0^2 and R_1^2 are exactly those belonging to R_0^1 and R_1^1, respectively.

Proof Note that it suffices to show that π belongs to F_2^* when the conditions of the lemma are satisfied, due to the symmetry of the conditions on π and π' in the case when both exist. Note that by Lemma 6, π must be considered for inclusion in F_2^* by the second application of SP2.

The proof is by induction on the sequence of insertions of nodes into F_1^*. If $v \in I$, the lemma clearly holds; we therefore may assume that $v \notin I$, that v and $\overline{v}$ satisfy one of the two sets of conditions of the lemma, and that the lemma holds for all nodes already in F_1^* that satisfy the conditions.

When the path π is considered by the first application of SP2 at Step 6c (via the edge (u, v)), we have three cases:

- $d > D[\overline{v}]$.

In this case, SP2 attaches v to F_1^* using the first entry to emerge from *PQ*. Since π is considered first among all paths to v in the first application of SP2, Lemma 6 implies that π is the first path to v considered in the second application as well. The assumption that no edges of A lie on the shortest path to $\overline{v}$ imply that $d > D[\overline{v}]$ when π is considered in the second application, and therefore π is inserted into F_2^*.
Since $d > D[\overline{v}]$, $\overline{v}$ must already have been added to F_1^*, and is therefore reachable. Step 6d must have been executed, since $d \neq D[\overline{v}]$. The inductive hypothesis implies that the paths to u and $\overline{v}$ in F_2^* are identical to those in F_1^*, and that the assignment of nodes u and $\overline{v}$ in the second application of SP2 matches that of the first application. Node v is therefore assigned to R_0^2 if $v \in R_0^1$, and to R_1^2 if $v \in R_1^1$. The lemma follows for this case.

- $d < D[\overline{v}]$.

For reasons similar to those of the previous case, π is inserted into F_2^* in this case as well.
Since $d < D[\overline{v}]$, $\overline{v}$ must not yet have been added to F_1^*; it may or may not be reachable. In any event, it cannot yet have been assigned to R_0^1. Step 6d must have been executed, since $d \neq D[\overline{v}]$. The inductive hypothesis implies that the path to u in F_2^* is identical to that in F_1^*, and that the assignment of node u in the second application of SP2 matches that of the first application. Since $\overline{v} \notin R_0^1$, v is therefore assigned to the same set as u. Since v is assigned to the same set as u in the second application of SP2 as well, v is assigned to R_0^2 if $v \in R_0^1$, and to R_1^2 if $v \in R_1^1$.

- $d = D[\overline{v}]$.

In this case, if $\overline{v} \notin R_1^1$ when the first path to v emerges from *PQ*, then this path is used to attach v to F_1^* — that is, π. As in the previous cases, Lemma 6 implies that π is the first path to v considered in the second application as well. The assumption that no edges of A lie on the shortest path to $\overline{v}$ imply that at the moment when π is considered in the second application, $\overline{v}$ is in R_0^2 if it is in R_0^1, and unassigned otherwise. In either case, π is inserted into F_2^*. Since u satisfies the conditions of the lemma, the inductive hypothesis implies that $u \in R_0^2$ if $u \in R_0^1$, and $u \in R_1^2$ if $u \in R_1^1$. Steps 6d to 6f then ensure that v is assigned to the same sets as u in both applications of SP2.
If $\overline{v} \in R_1^1$ when the first path to v emerges from *PQ*, then the correctness of SP2 implies that when v is attached via π, that (u, v) and $(\overline{u}, \overline{v})$ both appear in F_1^*. Steps 6d to 6f ensure that u and v are assigned to R_0^1, and $\overline{u}$ and $\overline{v}$ are assigned to R_1^1. Nodes u, $\overline{u}$, and $\overline{v}$ all satisfy the conditions of the lemma, and therefore by the inductive hypothesis we have $u \in R_0^1$, $\overline{u} \in R_1^2$, and $\overline{v} \in R_1^2$. As a result, the only way v can be attached to F_2^* is via u, assigned to R_0^1. This completes the proof of this case, and the lemma. □

E. Heuristics for Coaxing

Lemma 7 provides us an important guarantee regarding the limitations of the effect of edge weight adjustments upon the structure of the shortest path tree. It tells us that the fewer the edges whose weights are adjusted, the greater the portion of the tree whose structure is undisturbed. This is a particularly important consideration when attempting to improve upon a shortest path tree which already has many desirable characteristics.

Consider the effect of adjusting the weights $\lambda(e)$ and $\lambda(\overline{e})$ of edges $e = (u, v)$ and $\overline{e} = (\overline{u}, \overline{v})$ by adding to them the amount $\Delta > 0$. If Δ is chosen to be larger than the sum of the original edge weights of G, on the next application of SP2, e will not be inserted into the shortest path tree unless no other path of unadjusted edges exists to v. The following three heuristics all adjust the weights of only a single pair of edges, by a constant amount Δ.

1. *Given $u \in R_1^1$ and $v \in R_0^1$, we want $u \in R_0^2$.*

This situation can arise when v corresponds to an and-node in G, as discussed earlier in §III-F. Interpreting R_0^1 and R_1^1 as temporal divisions in which R_0^1 events occur before R_1^1 events, the and-node v from the earlier division would appear to depend on another event occurring at the later division. Moving u to R_0^2 in a second application of SP2 would resolve this anomaly.
If the shortest path length to $\overline{u}$ is no more than that of u, and if the parent $w_{\overline{u}}$ of $\overline{u}$ in F_1^* is not paired with that of u, $\overline{u}$ can be coaxed away from R_0^1 and into R_1^2 by increasing the distance from $\overline{u}$ to $w_{\overline{u}}$. This adjustment of the weights of $(w_{\overline{u}}, \overline{u})$ and its paired edge $(\overline{w_{\overline{u}}}, u)$ does not affect the distance from r to u, since $\overline{w_{\overline{u}}}$ is not the parent of u in F_1^*.
On the other hand, if $\overline{u}$ is unreachable from r, if the shortest path length to $\overline{u}$ is already greater than that of u, or if the parent of $\overline{u}$ in F_1^* is paired with that of u, the weight adjustment described above will be ineffective. In the first two cases, u is attached to F_1^* before $\overline{u}$, yet nevertheless was assigned to R_1^1. Increasing the distance to $\overline{u}$ will not alter this assignment of u. In the third case, increasing the distance to $\overline{u}$ in this way increases the distance to u by the same amount. Instead, one way in which u can be encouraged to join R_0^2 is to increase the weight of the unique cut edge (a, b) in the shortest path leading to u in F_1^*. By adding Δ to the weight of (a, b), the second application of SP2 is guaranteed to find a different, unadjusted path to u if such a path exists. This path, if shorter than that to $\overline{u}$, may allow u to be assigned to R_0^2 instead of $\overline{u}$. Note that the weight of $(\overline{a}, \overline{b})$ must be adjusted by the same amount.
Heuristic 1: Let (u, v) be an edge of E^* such that $u \in R_1^1$ and $v \in R_0^1$. Let (a, b) be the unique cut edge on the path to u in F_1^*, and let w_u and $w_{\overline{u}}$ be the parents of u and $\overline{u}$ in F_1^*, respectively (let $w_{\overline{u}} = \emptyset$ if $\overline{u}$ is not reachable from r). To encourage u to be assigned to R_0^2 in the next application of SP2, then:

- If $D[\overline{u}] \leq D[u]$ and $\overline{w_u} \neq w_{\overline{u}}$ after the execution of SP2, adjust weights $\lambda_2((w_{\overline{u}}, \overline{u})) \leftarrow \lambda_2((\overline{w_{\overline{u}}}, u)) \leftarrow \lambda_1((w_{\overline{u}}, \overline{u})) + \Delta$.
- If $D[\overline{u}] > D[u]$ or $\overline{w_u} = w_{\overline{u}}$ after the execution of SP2, adjust weights $\lambda_2((\overline{a}, \overline{b})) \leftarrow \lambda_2((a, b)) \leftarrow \lambda_1((a, b)) + \Delta$.

2. *Given $u \in R_1^1$ and $v \in R_0^1$, we want $v \in R_1^2$.*

Another way of resolving the anomaly mentioned above is to coax v into entering R_1^2.
If the shortest path length to $\overline{v}$ is no less than that of v (including the case where $\overline{v}$ is unreachable from r), and if the parent w_v of v in F_1^* is not paired with that of $\overline{v}$, v can be coaxed away from R_0^1 and into R_1^2 by increasing the distance from w_v to v. This adjustment of the weights of (w_v, v) and its paired edge $(\overline{w_v}, \overline{v})$ does not affect the distance from r to $\overline{v}$, since $\overline{w_v}$ is not the parent of $\overline{v}$ in F_1^*.
On the other hand, if the shortest path length to $\overline{v}$ is already less than that of v, or if the parent of $\overline{v}$ in F_1^* is paired with that of v, the weight adjustment described above will be ineffective. In the former case, $\overline{v}$ is attached to F_1^* before v, yet nevertheless was assigned to R_1^1. Increas-

ing the distance to v will not alter this assignment of v. In the latter case, increasing the distance to v in this way increases the distance to $\overline{v}$ by the same amount. Instead, one way in which v can be encouraged to join R_1^2 is to increase the weight of the unique cut edge $(\overline{a}, \overline{b})$ in the shortest path leading to $\overline{v}$ in F_1^*. By adding Δ to the weight of $(\overline{a}, \overline{b})$, the second application of SP2 is guaranteed to find a different, unadjusted path to $\overline{v}$ if such a path exists. This path, if shorter than that to v, may allow $\overline{v}$ to be assigned to R_0^2 instead of v. Note that the weight of (a, b) must be adjusted by the same amount.

Heuristic 2: Let (u, v) be an edge of E^* such that $u \in R_1^1$ and $v \in R_0^1$. Let $(\overline{a}, \overline{b})$ be the unique cut edge on the path to $\overline{v}$ in F_1^*, and let w_v and $w_{\overline{v}}$ be the parents of v and $\overline{v}$ in F_1^*, respectively (let $w_{\overline{v}} = \emptyset$ if $\overline{v}$ is not reachable from r). To encourage v to be assigned to R_1^2 in the next application of SP2, then:

- If $D[\overline{v}] \geq D[v]$ and $\overline{w_v} \neq w_{\overline{v}}$ after the execution of SP2, adjust weights $\lambda_2((w_v, v)) \leftarrow \lambda_2((\overline{w_v}, \overline{v})) \leftarrow \lambda_1((w_v, v)) + \Delta$.
- If $D[\overline{v}] < D[v]$ or $\overline{w_v} = w_{\overline{v}}$ after the execution of SP2, adjust weights $\lambda_2((\overline{a}, \overline{b})) \leftarrow \lambda_2((a, b)) \leftarrow \lambda_1((a, b)) + \Delta$.

3. *Given $(u, v) \in F_1^*$, we want $(u, v) \notin F_2^*$.*

This situation can arise when $\overline{u}$ corresponds to an and-node in G. Although $\overline{u}$ may have a valid interpretation within the graph, the paired node u generally will not. If u were a leaf of the shortest path tree F_2^*, no other node of F_2^* would depend from u, relieving the need to interpret u. The action to be taken here is straightforward: by increasing the weight of (u, v) and its paired edge $(\overline{u}, \overline{v})$, v will be connected to F_2^* through some unadjusted path whenever such a path exists. Increasing the weights of all children of u in F_1^* would strongly encourage u to become a leaf of F_2^*.

Heuristic 3: Let (u, v) be an edge of E^*. To discourage v from being the child of u in R_1^2 after the next application of SP2, then adjust weights $\lambda_2((\overline{u}, \overline{v})) \leftarrow \lambda_2((u, v)) \leftarrow \lambda_1((u, v)) + \Delta$.

Each of the heuristics outlined above adjusts the edge weights of a single pair (call it (x, y) and $(\overline{x}, \overline{y})$). Lemma 7 implies that this reweighting can affect only the portion of F_1^* below y and $\overline{y}$. In particular, some of the nodes in these portions of F_1^* may find themselves migrating from R_1^1 to R_0^2 or from R_0^1 to R_1^2. Although this migration may resolve anomalies of the kind discussed here, one should be aware that it has the potential to create them as well. For this reason, it is best to minimize the number of edge weight adjustments.

CHANGES SINCE THE ICSE 2001 VERSION

Here, we describe how this paper changed in light of the comments of the ICSE 2001 reviewers.

The major changes are:

- This paper is twice the length of the ICSE 2001 version and contains a formal proof of correctness for our algorithm
- The ICSE 2001 version of this work had a "two time tick" restriction on the temporal queries. This paper removes that restriction.
- The ICSE 2001's claim to generality was based on an analysis of coverage of the Dwyer corpus of temporal logic patterns. The removal of the "two time tick" restriction makes that analysis redundant. Hence, it was removed.

This paper incorporates many of the changes proposed by the ICSE 2001 reviewers: see below.

1. *Reviewer 1:* made many positive comments and made some editing suggestions- which we applied.
2. *Reviewer 2:* "How much of a system's requirements can be expressed in topoi diagrams (clearly some of the requirements can be)?". This is addressed in the introduction when we say:

Specifically, the systems model must be expressed as topos diagrams. Topoi are not very expressive, and exclude statements such as first-order assertions, iterations, sub-routine calls, and assignments. Due to these language limitations, our approach is not suited to domains that need the excluded statements, such as complex protocols seen in concurrent processes.

These restrictions are not fatal to the modelling process, at least at the requirements stage. We will show via an example that topos diagrams are sufficient to represent a wide range of diagrams seen in certain approaches to requirements engineering and recording design rationales. Hence, when we say that this approach is practical and useful, we really mean *practical and useful for early life cycle requirements discussions only*.

3. *Reviewer 2:* "What is the connection of this work to flow graph analysis?". This connection was unknown to us in the earlier version and is discussed in §IV-C.1.
4. *Reviewer 3:* This reviewer was concerned that the presentation of our techniques was being over-sold, that we were claiming more than we should regarding the power of our tools. We agree- the ICSE 2001 paper confused logical 'proofs' with 'counter examples'. This draft removes that confusion.
5. *Reviewer 3:* The ICSE 2001 version argued for the generality of this technique via a review of the Dwyer corpus of temporal logic patterns. Reviewer 3 doubted this analysis but this issue is now redundant. The ICSE 2001's claim to generality was based on an analysis of coverage of the Dwyer corpus of temporal logic patterns. The removal of the 'two time tick' restriction makes that analysis redundant. Hence, it was removed.
6. *Reviewer 3:* "Why would subroutine calls be of concern at the requirements stage?" They should not- we only mention these to distinguish our topos approach from systems like SPIN which can handle sub-routines.
7. *Reviewer 3:* "How is 'rigorous requirements engineering' related to 'formally checking a system by formal first-order query'?". By rigorous RE, we mean that formal testing is applied early in the lifecycle. As to 'formal first-order queries':
   - Various researchers have explored lightweight formal methods using temporal logic model checkers.
   - Cost-benefit studies of these approaches include Lowrey, Boyd & Kulkarni [34], who argue that a single test is like a ground propositional query while a formal query is like a first-order sentence with non-ground variables that can range over a larger set of values.
   - Hence, a single formal query can substitute for many tests and can increase our rigor in RE
8. *Reviewer 3:* "Why is the freezing of requirements (applying the development brake) a result of costs? " A full formal analysis is expensive and, once completed, may be too costly to repeat. That is, any changes in the requirements found *after* the formal analysis may be ignored (since to do otherwise can be too expensive).
9. *Reviewer 3:* "these terms seem interchangeable: testing requirements, formal requirements analysis, rigorous requirements engineering" Yes. we have removed the repeated terms.
10. *Reviewer 3:* "A reference to topoi diagrams would be useful.". The reference [4] has been added to page one.
11. *Reviewer 3:* "It sounds as if the topoi diagrams are used for both the properties and the systems model; and then later the two models seem to be mixed up, used interchangeably." Yes- the text now distinguishes between compiling a system description into a topos and compiling a temporal logic properties model into set memberships of parts of a time series generated from that topos, see §III-C.
12. *Reviewer 3:* "is a query a property in the properties model". No- a query is a test for the entire properties model.
13. *Reviewer 3:* "The claims about topoi graphs being quickly generated and systems modelling costs being low is only justified based on the authors' experience. It would at least be help to state their experience doing what, especially since this claim underlies the entire methods' effectiveness." Relevant references ([4] and Menzies [7]) are offered in the introduction.
14. *Reviewer 3:* "Where in the topoi Figures are the strengths which are continuous numbers.". In all our topos examples, the strengths default to 1 for each edge. The repeated assignment of 1 to each edge is not shown since this would confuse the reader.
15. *Reviewer 3:* "Figure 4 is totally unreadable". Yes- and that is our point. Real world topoi such as Figure 4 soon become too complicated to read manually. Automatic tools are required to implement this process.
16. *Reviewer 3:* "The authors had a pre-experimental concern that informal topoi are so under-defined that anything could be inferred; they state that this turns out not to always be the case therefore implying that sometimes it is the case. How good is this?" The experience of [7] is that, often, it is good enough. A similar belief that they are good enough (that is, under-defined models can still be used to make interesting conclusions) is the base premise of the qualitative reasoning community [12, 35–37]. A similar endorsement of under-defined theories is offered by researchers discussing the merits of lightweight notations for testing software (e.g. [38, chp 18] [3]).
17. *Reviewer 3:* "Are vertices equivalent to variables since they can be assigned values?" No- if a variable can be assigned N distinct values, then that would imply N vertices
18. *Reviewer 3:* "Who does the coaxing by applying edge weights to a topoi? This seems extremely burdensome." The appendix and the source code of RAPTURE (in Figure 8) makes it clear that it is an automatic process.
19. *Reviewer 3:* "It's not clear what the plateau reached after 10 coaxes barely changed in up to 100 coaxes means; isn't a plateau usually set and processing stops when it is reached?" That is the strategy that should be applied at runtime. However, to test if coaxing is missing anything, what we did here is to compare the goals found after a small number of coaxes to the the goals found after a very very large number of coaxes.
20. *Reviewer 3:* "Is it possible to tell in advance whether requirements can be mapped into symmetric topoi?" Yes- if the model can be abstracted to a propositional theory (that is, two assignments per variable).
21. *Reviewer 4:* "Most of the technical meat of the paper is glossed over. The SP2 algorithm is summarized, but relatively easy to follow. The details are apparently presented in a technical report and another seemingly unpublished report, rather than presented in this paper." The SP2 algorithm is presented in full in this paper.

22. *Reviewer 4:* "One has to question whether a topoi whose analysis needs to be coaxed is cost effective. Automated analysis works best on models that are difficult to reason about manually (e.g., model checking a concurrent models, where it is difficult for a human to reason about the global state of the system). Topos diagrams are a global model of simple relations. While I can see the benefit of having an automated analyzer push a flow through the graph, this benefit deteriorates if the human analyzer has to play with assigning weights to edges in order to force the analyzer to consider feasible paths. Wouldn't it be easier, more cost effective, and no less error-prone to have the human analyzer inspect the Topoi?" Two comments here:

- Coaxing is an automatic process- this was NOT clear in the ICSE2001 paper but is here
- As to manually inspecting the topos, we repeat our remarks at the start of §II-C:

Topoi seem quite simple, but often defy manual analysis, possibly due to their over-constrained and over-generalized nature. For example, Menzies' topos tester [7] found behaviors in very small topoi (Figure 2.iv) that had not been discovered even after days of manual analysis. The difficulty in analyzing topoi worsens as the topoi become more complex, as often happens when statements of gradual influence are collected from more than one stakeholder.

23. *Reviewer 4:* "The paper should reference the work of Jim Corbett, who models requirements as flow graphs (not unlike Topoi graphs), but transforms the graph and the property into a set of equations and inequalities and looks for an integer solution (if there is none, then the system never violates the property)." Done- see §IV-C.1.