ELSEVIER

Information and Software Technology 44 (2002) 137–150

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Temporal fault trees

Girish Keshav Palshikar*

Tata Research Development and Design Centre, 54B, Hadapsaur Industrial Estate, Pune 411013, India

Received 25 May 2001; revised 9 November 2001; accepted 11 December 2001

Abstract

Fault tree (FT) is a simple, visual, popular and standardized notation for representing relationships between a fault in a system and the associated events. FTs are widely used for supporting products and systems in diverse industries like process control, avionics, aerospace, nuclear power systems, etc. where they are used to capture specialized and experiential knowledge for diagnosis and maintenance. FTs are also used to represent safety requirements of a system, obtained during the hazard analysis phase of the system development cycle. However, a problem that prevents more analytical use of FT is their lack of rigorous semantics. Users' understanding of an FT depends on the clarity and correctness of the natural language annotations used to label and describe various parts. Moreover, it is not clear how to adapt the FT notation to represent temporal relationships between faults and events in dynamic systems. We propose to augment the FT notation by adding simple temporal gates to capture temporal dependence between events and faults. We propose techniques to perform qualitative analysis of such temporal fault trees (TFT) to detect the causes of the top event fault by matching the TFT with the trace (or log) of the system activities. We present two algorithms for depth-first traversal and cut-set computations for a given TFT that can be used for diagnosis based on TFTs.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Fault tree; Temporal logic; Hazard analysis; Safety analysis; Diagnosis

1. Introduction

A large number of notations have been developed for formal specification of requirements of real-time safety-critical systems, where explicit temporal aspects of the system can be rigorously expressed, e.g. temporal logics [1–9] (see Ref. [1] for an excellent survey), timed automata [10–13], timed Petri nets [14–18], as also various process algebras. In many such systems, an important sub-class of requirements deals with detection, control, prevention and handling of various faults. Hence, hazard analysis is a critical step in the building of real-time safety-critical (and mission-critical) systems and products. During hazard analysis, the possible external hazards and internal faults (e.g. component failures) that such a system may suffer from and their relationships to various events are identified. Hazard analysis leads to identification of safety requirements of the system to be built. These safety requirements are taken into account during the design, implementation, testing, operation and maintenance of the system.

Fault tree (FT) is a simple, visual, standardized [19] and popular notation used to state safety requirements gathered during hazard analysis. A number of FT tools have been reported for safety analysis of systems; see for example, Refs. [20,21].

There are two basic ways in which FTs are used in the life cycle of a safety- or mission-critical system. In the first case, the FTs are taken as specifications of a part of the system requirements and then the system design and implementation is checked against these specifications. Often, only parts of the requirements are specified in the FT notation, the rest of the requirements being specified in other notations. In this case, the users of FTs are system analysts, designers and developers. FTs are increasingly being used to express safety requirements of not only physical systems but also of digital and software systems [22]. For example, in Refs. [23,24], software FTs were used to verify safety properties of software written in Ada.

In the second case, the FTs describe the faults that can happen in a system and relate them to their causes. In this case, the FTs are constructed 'away' from the requirements and design documents and the main 'users' of FTs are operations and maintenance/support engineers. In this way, the FT notation is widely used for supporting products and systems in diverse industries like process control, avionics, aerospace, nuclear power systems, bio-medical instrumentation, etc. where it is used to capture specialized and experiential knowledge for diagnosis and maintenance. Many kinds of analysis (e.g. model-based safety analysis

* Tel.: +91-20-687-1058; fax: +91-20-681-0921.
E-mail address: girishp@pune.tcs.co.in (G.K. Palshikar).

[25], automatic FT generation [26], dependability analysis [27] and much more) and fault diagnosis procedures are developed for safety requirements and fault knowledge expressed using FT; see Ref. [28] for an overview of the basic techniques for FT analysis.

However, one problem that prevents more widespread use of the FT notation is their lack of rigorous semantics. Users' understanding of the requirements represented in these notations depends on the clarity and correctness of the natural language annotations used to label and describe various parts. Such lack of rigour leads to ambiguity and prevents formal analysis of the safety requirements and knowledge expressed using FT. Moreover, it is not clear how to adapt the FT notation to represent temporal relationships between faults and events in dynamic systems. In this paper, we attempt to address these issues.

In one stream of closely related work, the essential idea is to enhance the FT notation and relate it to timed Petri net models. For example, in Ref. [29], the authors have used a system of equalities and inequalities (which captures the temporal context of events) to perform analysis of time dependencies in an FT. They have also shown that the expressive power of the resulting FT notation is the same as that of a timed Petri net. In our work, we have added *temporal gates* (to the FT notation) that directly map to connectives in a temporal logic and the users do not provide any further quantitative information. We also do not relate our TFT notation to any executable mode like timed Petri nets or timed automata. However, we have a prototype implementation of the algorithms for qualitative analysis of a given TFT. A similar work is reported in Ref. [30] where an enhanced FT notation is used to specify the nature of intrusions in an intrusion detection system, which is then automatically translated to a coloured Petri net and subsequently, to an implementation. Apart from special gates for trust and context, this work also includes some special gates in the FT notation for temporal ordering and intervals, as based on the interval temporal logic of Allen and Ferguson [5]. The research reported in Ref. [31] is also directly related to our work. However, they have retained the basic FT notation and chosen to label the leaf nodes with formulae in Duration Calculus [2]. We have instead retained the simple propositional labelling of the leaf events and added temporal gates to the FT notation to explicitly depict temporal dependencies. We have also used a much simpler instance-oriented linear temporal logic PLTLP.

Several attempts have been made to provide a simple visual front-end to temporal logics—none of them adopt the FT notation to the best of our knowledge; see Refs. [3,4], which adopted the timing diagram notation. It is also not the intention of the paper to simply define a diagrammatic visual interface for a temporal logic (as in Refs. [3,4]), although this work can be viewed that way.

In this paper, we propose a simple way of enhancing the FT notation, which results in what we call as temporal fault trees (TFT). The TFT notation allows the users to easily specify the temporal dependence between events and faults. The enhancements in TFT are based on the premises that

(i) the simple, qualitative and visual nature of the FT notation should be preserved;
(ii) the underlying semantics of the FT notation should be as intuitive and simple as possible;
(iii) a user need not be expert in temporal logic to build or understand the TFT. More specifically, although the user needs to understand the basic semantics of the temporal connectives, detailed technical knowledge about say temporal deduction, model-checking, etc. is typically not required.

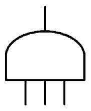
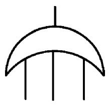
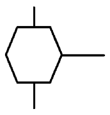
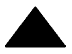
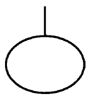
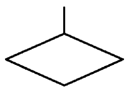
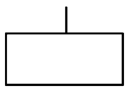
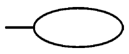
The method we have used to enhance the FT notation is essentially through an introduction of new temporal gates in the FT notation. The events are still labelled by simple atomic propositions without any structure. This is in contrast with other approaches where the events are actually complex temporal conditions. The semantics of the TFT notation is defined in terms of a past-oriented simple propositional linear temporal logic PLTLP [1]. We also extend the standard top-down depth-first traversal procedure for FT to work with a TFT; this procedure is used to identify the set of events that lead to the occurrence of the top event in the TFT.

It is not an intention of the paper to invent any new notation for the specification of time-dependent requirements of real-time safety-critical systems. As mentioned, the purpose of the paper is to enhance the FT notation by adding explicit temporal facilities so that the new FT notation can better and more explicitly reflect the temporal relationships between events/conditions for a specific hazard. We also then intend to perform some FT-specific analysis for the enhanced FT notation. The way in which temporal facilities are added will naturally reflect how the semantics of the enhanced FT notation is to be defined. Thus, for example, the semantics of such an enhanced FT notation can be defined in terms of say, timed Petri nets [29] or timed automata. Following, for example Refs. [30,31], we have chosen temporal logic since we feel that just as propositional logic bears a close resemblance to the (untimed) FT notation so the propositional temporal logic would provide a close foundation for the TFT notation.

It is well known that temporal logic (particularly the simple temporal logic used in this paper) is inadequate in many ways to capture the entire spectrum of requirements of safety-critical systems (as in the first case). It is to the second use of FTs that the paper is directed. In this case, the FTs are simply used to relate a hazard to a temporal sequence of causal events or conditions. The main contribution of the paper is that the temporal relationships of a top event with such events/conditions are made obvious in the diagram, rather than remaining hidden implicitly in the textual labels on the (ordinary) FT diagrams.

The rest of the paper is organized as follows. Section 2

Table 1
Commonly used symbols used in the FT notation

Gate symbol	Gate type	Informal description
	AND gate	Output event occurs only if all input events occur
	OR gate	Output event occurs if any one of the input events occurs
	INHIBIT gate	Input event produces output event only when the conditional event is present
	NOT gate	Output event is the negation of input event
Event symbol	*Event type*	
	Basic event	A primitive well-understood event with sufficient data
	Undeveloped event	An event that cannot be developed any further, typically as a result of limiting the analysis
	Intermediate event	An event represented by a gate
	Conditional event	An event to control the INHIBIT gate

describes the basic FT notation and a simple formalization for it. Section 3 proposes some temporal enhancements for the FT and defines the semantics of the resulting TFT notation in terms of a simple temporal logic. Section 4 describes some illustrative examples. Section 5 describes algorithms for depth-first traversal and cut-set computations for TFT which can be used for diagnosis based on TFTs. Section 6 contains conclusions, discussion of our work and further work.

2. Fault trees

2.1. The FT notation

An FT allows a top-down logical representation of diagnostic knowledge and failure propagation. An FT is essentially a vertex-labelled, directed, rooted, bipartite tree. There are two classes of vertices in an FT: *logical gates* and *events* of various types (Table 1 depicts the common symbols used in FT; there are several others). Labels are often associated only with leaf (i.e. non-intermediate) event vertices; labels are typically natural language textual descriptions of conditions. Fig. 1 shows an example of an FT.

The root is an event vertex, called the *top-event* and indicates a system failure of some kind. The sub-failures that lead to an event are indicated by the next level of event vertices. These event vertices are connected to the event by means of a logical gate. An AND gate is used if occurrences of all the sub-failures in the next level together cause the event. An OR gate is used if occurrence of one or more sub-failures cause the event. Each of these sub-failures is then expended (or developed) similarly. A *basic event* is an event that is not developed further. Typically, several

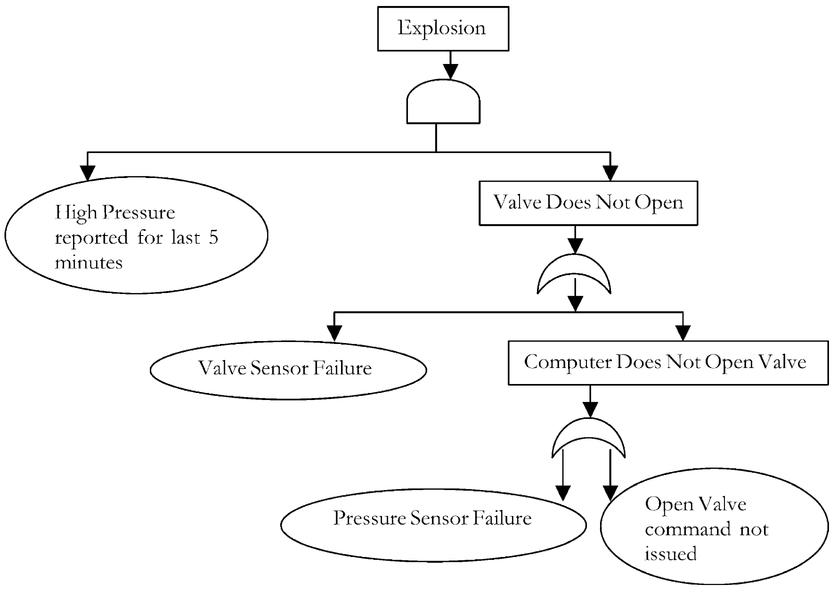

Explosion
High Pressure reported for last 5 minutes
Valve Does Not Open
Valve Sensor Failure
Computer Does Not Open Valve
Pressure Sensor Failure
Open Valve command not issued

Fig. 1. An example of a fault tree.

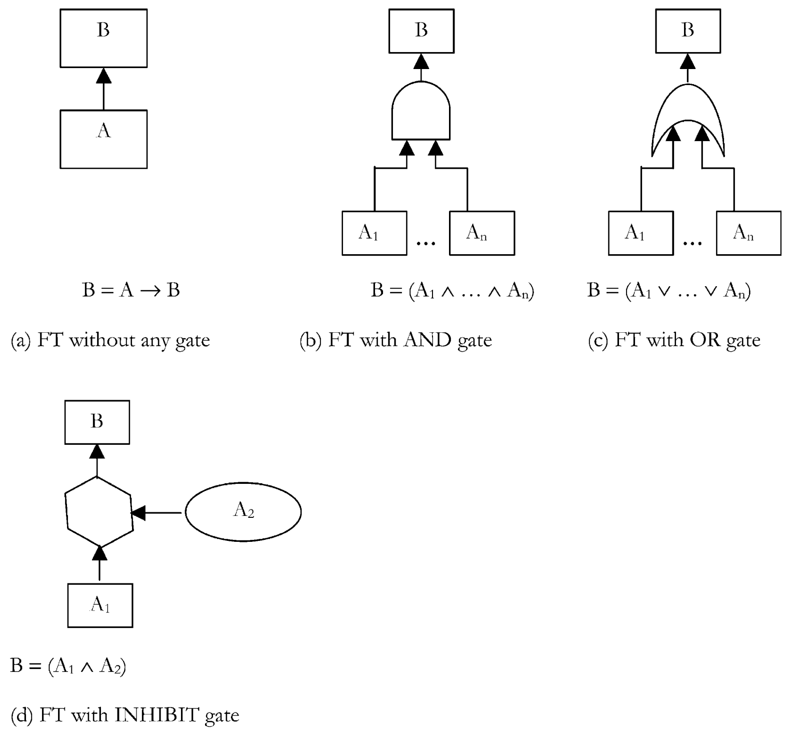


Fig. 2. Semantics of the logical gates in FT.

failures are identified for a system and a set of FT is developed—one for each failure.

There is another way to classify the events in an FT, although we shall not use this classification. An event in an FT is a *primary event* if it represents the failure of a component due to its internal defects (e.g. valve stuck open). An event in an FT is a *secondary event* if it represents the failure of a component due to excessive environmental or operational stress (e.g. motor overloading). An event in an FT is a *command event* if it represents the failure of a component due to incorrect or absent control signals or (e.g. failure of a relay due to the coil not being energized).

In Fig. 1, the top event denotes an occurrence of an explosion. This can happen when the pressure is high for the last 5 min (which is depicted as a primary event) and the valve is not opened. The valve is not opened because the valve sensor may have failed (e.g. erroneously reporting that the valve is already open—depicted as a primary event) or because the control computer failed to issue the valve open command. The computer may fail to open the valve because of failure of the pressure sensor (e.g. erroneously reporting low pressure) or because the valve open command was not generated (e.g. because of a bug in the software—both of which are depicted as primary events). Note that some of the primary events (e.g. high pressure for the last 5 min) are actually explicitly temporal conditions and some others (e.g. valve failure) may contain implicitly temporal information. In this paper, we propose additional facilities in the FT notation using which such temporal conditions and temporal relationships between events can be explicitly represented in the FT.

2.2. Formalization of FT

An important problem in understanding FT is the informal nature of the notation. It is easy to see that there is a need for formalization of the semantics of the FT notation. What exactly is the meaning of the information represented in an FT? What is the meaning of a collection of FTs? Clearly, the logical gates have a precise meaning that can be derived from mathematical logic. However, one problem is the informal (natural language) nature of the event labels associated with leaf nodes in FT. Can some more precise meaning be associated with the event description? A frequent solution for this problem is to impose a rigorous notation for writing an event label. In this approach, events often represent a condition on the state of a system.

Let $\mathrm{VAR} = \{V_1 : D_1, \ldots, V_n : D_n\}$ be a set of *n system state variables* V_i, along with a finite non-empty *domain* of possible values D_i associated with the variable V_i. In general, the system state variables include the inputs and controls from the environment, outputs generated by the system as well as internal state variables maintained by the system. At any point in time, the state of the system is a vector of *n* values $s = \langle d_1, d_2, \ldots, d_n \rangle$ where $d \in D_i$ is the current value of the state variable V_i. For a dynamic system, the values of the state variables (i.e. state vector) change over time as the system receives and processes the inputs and generates outputs. A *condition* is a Boolean expression containing the names of variables V_i and the constants from their domains. Thus a condition is *instantaneous* in the sense that it refers only to the current values of the state variables

Table 2
Intuitive meaning for the temporal connectives in PLTLP

$\mathbf{O}^{-}X$	X is true at the previous instant
$\mathbf{O}_n^{-}X$	X is true at the instant t_{k-n} in the past (if $k \geq n$); false if $k < n$
$\Box^{-}X$	X is true now and for all the previous instants in the past
$\Box_n^{-}X$	X is true now and for the last n instants in the past
$\Diamond^{-}X$	X is either true now or at some instant in the past
$\Diamond_n^{-}X$	X is either true now or at some instant within the last n instants
$X\mathbf{U}^{-}Y$	Sometime in the past Y holds and X holds everywhere before that
$X_m\mathbf{C}_nY$	X is true now and at the previous m instants and Y is true for n instants before that
$X_m\mathbf{SC}_nY$	X is true now and at the previous m instants and Y is true for the strictly earlier n instants before that
$X_m\mathbf{CF}_nY$	X is true now and at the previous m instants but Y is not true at all the earlier n instants before that
$X_m\mathbf{FC}_nY$	X is not true either now or at some of the m earlier instants and Y is always true for all the previous n instants before the m instants from now

and does not depend on the past values. Given a state vector (i.e. the system state at a particular time instant), an instantaneous condition evaluates to either *true* or *false*. We assume that each event in an FT is labelled by an instantaneous condition.

As an example, suppose the state variable V_1 represents the water level within a mineshaft and has the domain [0,10] of all real numbers between 0 and 10. Then low $= V_1 < 2$, normal $= V_1 \geq 2 \wedge V_1 \leq 8$ and high $= V_1 > 8$ represent the low, normal and high water level conditions, respectively. Thus low, normal and high can be used as event labels in an FT, with the understanding that they refer to specific conditions on the current system state vector. We shall assume from here onwards that the leaves in an FT are labelled with such atomic proposition symbols.

The semantics of any given FT can then be easily defined by treating the logical gates as the standard connectives in mathematical logic and event labels as atomic proposition symbols. Fig. 2 shows our formalization of such a *propositional FT*. Essentially, each FT corresponds to a formula in propositional logic. In the FT notation, there is an implicit upward flow of increasing abstraction across the levels of the FT from the leaves to the root node. To make this flow explicit we have used directed edges in the FT. Note that this semantics is compositional in the sense that meaning of an FT, which is composed of sub-trees connected by means of a logical gate, is inductively defined in terms of the semantics of the sub-trees and that of the logical gate.

3. Temporal fault tree

It is easy to see that an FT cannot represent conditions that change over time. The labels often include text fragments like 'too late', 'too long', 'eventually', 'before', etc. which need to be understood formally. Stated another way, the entire FT needs to be evaluated at the *same* state vector; it is not clear how to use the FT notation to describe explicit temporal relationships between events. We propose to handle this problem by introducing additional *temporal gates* in the FT notation. We then define the semantics of this TFT in terms of the past-oriented propositional linear temporal logic.

3.1. Propositional linear temporal logic PLTLP

We define a past-oriented linear propositional temporal logic PLTLP [1]. The underlying model of time is a linear sequence of discrete time instants, not necessarily equally separated and beginning at t_0; TIME $= t_0, t_1, \ldots$ where each t_i is a positive integer and $t_i < t_{i+1}$. Let PROP be a finite set of propositional symbols. Each propositional symbol from PROP has a Boolean value at each instant in time and this value may change over time. The past-oriented nature of the temporal logic is due to the top-down declarative temporal reading of a temporal fault, where one begins with a fault in the current instant and traces its causes in the past occurrences.

PLTLP provides the usual non-temporal operators or logical connectives: $\neg$ (not), $\vee$ (or), $\wedge$ (and), $\rightarrow$ (implies), and $\leftrightarrow$ (if and only if). In addition, there are several instance-oriented past temporal operators: $\mathbf{O}^{-}$ (PREV), $\mathbf{O}_n^{-}$ (PREV n), $\Box^{-}$ (ALLPAST), $\Box_n^{-}$ (FORPAST n), $\Diamond_n^{-}$ (WITHIN n), $\Diamond^{-}$ (SOMETIME-PAST), $\mathbf{U}^{-}$ (UNTIL-PAST), here n is a positive integer. There are also several pseudo-interval temporal connectives, called the *chop connectives*; ${}_m\mathbf{C}_n$ (m CHOP n), ${}_m\mathbf{SC}_n$ (m STRONG-CHOP n), ${}_m\mathbf{CF}_n$ (m CHOP-FAIL n) and ${}_m\mathbf{FC}_n$ (m FAIL-CHOP n) where m, n are given non-negative integers. The connectives that include a subscript are called *counting connectives*, e.g. $\mathbf{O}_n^{-}$, $\Box_n^{-}$, $\Diamond_n^{-}$ are counting connectives. The chop connectives appear to be new in linear temporal logic, although they are available in somewhat similar form in interval oriented temporal logics like the Duration Calculus [2] (see also Ref. [32]).

Each of the counting connectives actually stands for a set of connectives, one for each specific value of n. For instance, there are infinite connectives $\Diamond_1^{-}$, $\Diamond_2^{-}$ and so on. These connectives can also be defined inductively. For instance, the connectives $\mathbf{O}_n^{-}$ allow us to state that a formula X was true at precisely the nth instants in the past (from the current instant). These connectives can be inductively defined for a given positive integer n as:

$$\mathbf{O}_1^{-}X \equiv \mathbf{O}^{-}X$$

$$\mathbf{O}_n^{-}X \equiv \mathbf{O}^{-}\mathbf{O}_{n-1}^{-}X \qquad \text{if } n > 1$$

The syntax of a well-formed formula (wff) in PLTLP is

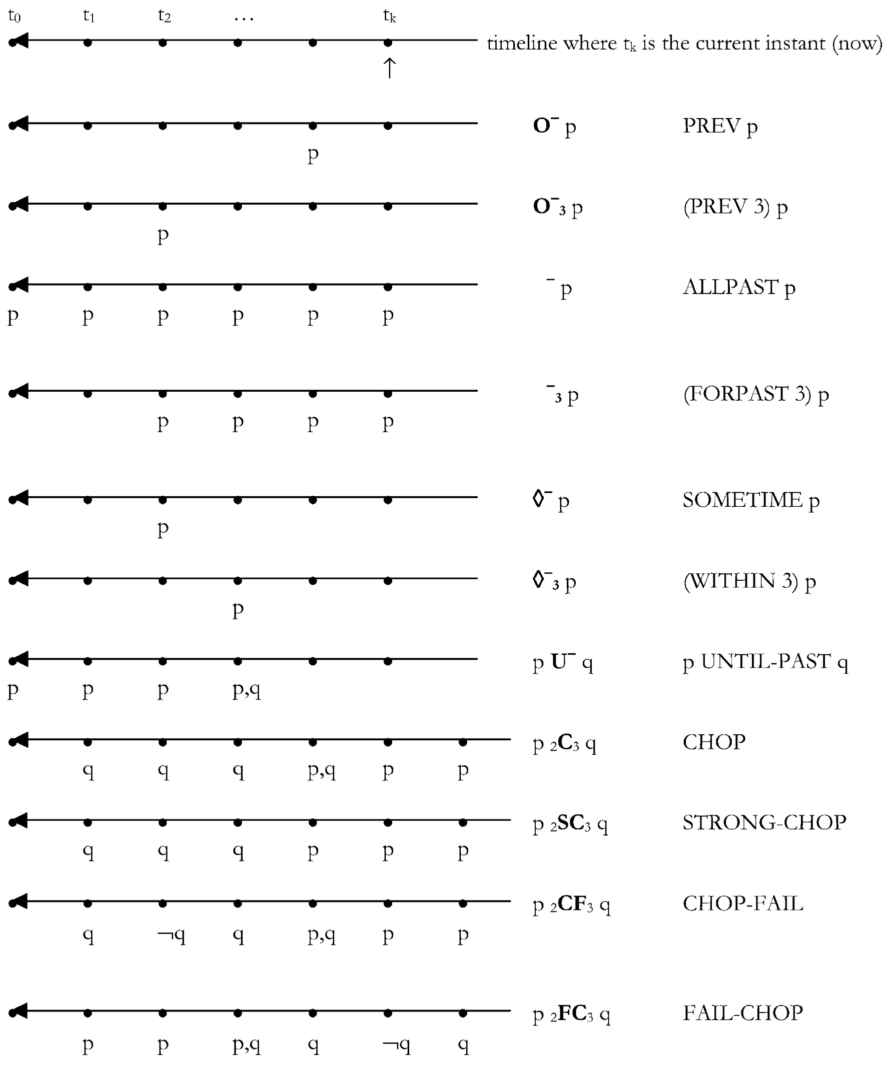


Fig. 3. Example state sequences to illustrate the intuition behind the temporal connectives. The rightmost dot indicates the current instant ‘now’.

inductively defined as follows

$$X ::= p|\neg X|X \vee Y|X \wedge Y|X \to Y|X$$
$$\leftrightarrow Y|\mathbf{O}^- X|\mathbf{O}_n^- X|\square_n X|\square X|\Diamond_n X|\Diamond X|X\mathbf{U}Y|X_m\mathbf{C}_nY|$$
$$X_m\mathbf{SC}_nY|X_m\mathbf{CF}_nY|X_m\mathbf{FC}_nY$$

where p is any propositional symbol from the set P and X, Y are arbitrary wff in PLTLP.

The intuitive meaning of the temporal connectives is given in Table 2. Note that the formulae involving counting connectives are false at the current instant t_k if there are less than n instances between the current instant and the initial instant t_0, i.e. if $k < n$.

Fig. 3 depicts the intuition behind the temporal connectives. Some examples of formulae in PLTLP are: rain $\to$ ($\Diamond_2^-$ (cold $\wedge$ humid)), lift_arrives $\to$ ($\Diamond^-$ (req_from_floor $\vee$ req_from_passenger)), $\square$ (door_open $\to$ (lift_stopped $\wedge$ alarm_on)).

As another example, a cycle of a simple square wave where the line remains high for first 2 s and low for the next 3 s is defined by the formula $S = p_2\mathbf{SC}_3(\neg p)$. Note that we *cannot* say, using the notation given in this section, the fact that the square wave consists of 10 such cycles in the past, as it involves counting modulo 5. Thus the CHOP connectives have to be used cautiously. Often, it is necessary to have the CHOP connectives as ‘outermost’ ones in a formula, e.g. in general, it is difficult to model situations where $\square_n^-$ is used to repeat a formula containing the CHOP connectives.

Note that we can define some of the connectives in terms of the chop connectives, e.g.

$$\mathbf{O}^- X = \text{true}_1\mathbf{C}_0X$$

$$\square_n^- X = \text{true}_0\mathbf{C}_nX$$

$$X_m\mathbf{CF}_nY = X_m\mathbf{C}_0(\neg\,\square_n^- Y)$$

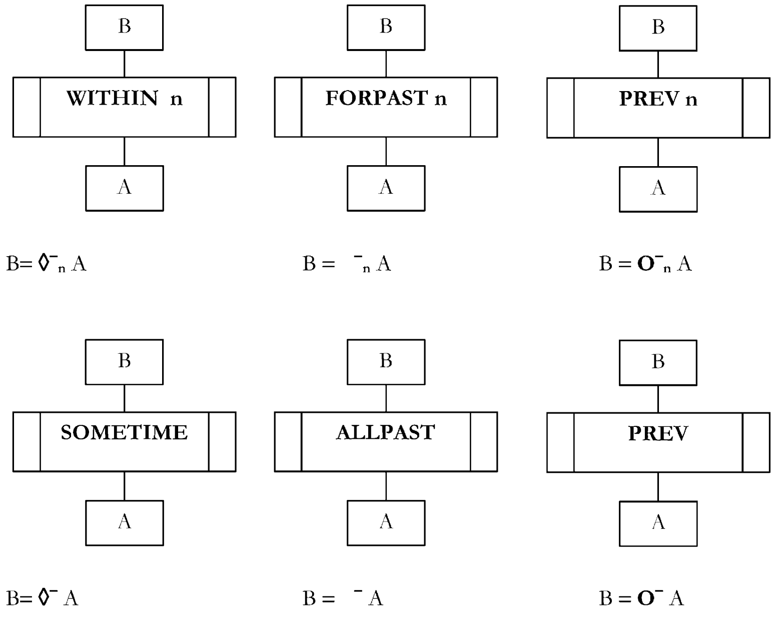


Fig. 4. Some temporal gates and the semantics of the associated TFT in terms of PLTLP.

3.2. Semantics of PLTLP

Let PROP be a finite set of propositional symbols. In this section, without loss of generality, we denote the timeline TIME by the sequence $[0, n] = 0, 1, 2, \ldots, n$. n indicates a (generic) current instance called 'now'. A *temporal interpretation* (or just *interpretation*) I: $[0,n] \rightarrow 2^P$ is a function which associates a subset of propositions from P with every time instant. If I is a temporal interpretation for P and if k is any time instant $k \in$ TIME such that $I(k) = P_1$, where $P_1 \subseteq$ PROP, we assume that all propositions in P_1 are true at the time instant k and all propositions in $P - P_1$ are false at the time instant k. Note that at any instant $k \in$ TIME, the value of the temporal interpretation $I(k)$ constitutes an *instantaneous interpretation* for P in the sense that $I(k)$ associates true or false values with symbols in P which hold only at the instant k. $I(k)$ is also called the *state* at time instant k.

We say that a PLTLP formula is *non-temporal formula* if it does not include any temporal operators; otherwise, the formula is said to be a *temporal formula*. Given a temporal interpretation I and a non-temporal PTL formula X, the instantaneous truth-value of X under I at any instant k in TIME, denoted by $I(k)(X)$, is the truth value of X under the instantaneous interpretation $I(k)$; $I(k)(X)$ can be obtained by any standard method of propositional logic (e.g. truth tables). If I: TIME $\rightarrow 2^P$ is a temporal interpretation and $k \in$ TIME, then I^k denotes the temporal interpretation obtained by a *temporal shift* of k time instants into the past, i.e. $I^k(i) = I(k - i)$ for $i = 0, 1, 2, \ldots$ and $i \leq k$. I^k is called the *k-shifted temporal interpretation*. Note that $I^0 = I$.

Let X be an arbitrary formula in PLTLP. Let I: TIME $\rightarrow 2^P$ be a temporal interpretation. The *truth-value of X in I*, denoted $I(X)$, is inductively defined below:

- if X is a non-temporal formula then $I(X) = I(0)(X)$;
- if X is of the form $\mathbf{O}^- Y$ then $I(X) = I^1(Y)$;
- if X is of the form $\mathbf{O}_m^- Y$ then $I(X) = I^m(Y)$ and false if $m > n$;
- if X is of the form $\Box^- Y$ then $I(X) = I^0(Y) \wedge I^1(Y) \wedge I^2(X) \wedge I^3(X) \wedge \cdots \wedge I^k(X)$. Alternatively, $I(X) =$ false if there is some $k \in$ TIME such that I^k(Y) = false; otherwise, $I(X) =$ true;
- if X is of the form $\Box_m^- Y$ then $I(X) = I^0(Y) \wedge I^1(Y) \wedge I^2(X) \wedge I^3(X) \wedge \cdots \wedge I^m(X)$ and false if $m > n$. Alternatively, $I(X) =$ false if there is some k, $n - m \leq k \leq n$, such that $I^k(Y) =$ false; otherwise, $I(X) =$ true;
- if X is of the form $\Diamond Y$ then $I(X) =$ true if there is some $k \in$ TIME such that $I^k(Y) =$ true; otherwise, $I(X) =$ false;
- if X is of the form $\Diamond_m^- Y$ then $I(X) = I^0(Y) \vee I^1(Y) \vee I^2(X) \vee I^3(X) \vee \cdots \vee I^m(X)$ and false if $m > n$. Alternatively, $I(X) =$ false if there is some k, $n - m \leq k \leq n$, such that $I^k(Y) =$ false; otherwise, $I(X) =$ true;
- if X is of the form $Y\mathbf{U}Z$ then $I(X) =$ true if there is some $k \in$ TIME such that $I^k(Z) =$ true and if $I^k(Y) = I^{k+1}(Y) = \cdots = I^n(Y) =$ true; otherwise, $I(X) =$ false.

The semantics of the chop connectives can be defined similarly.

A temporal interpretation I is a *temporal model* (or simply, *model*) for a PLTLP formula X if $I(X) =$ true. A PLTLP formula is called *temporally satisfiable* (or simply, *satisfiable*) if a temporal model exists for X; otherwise, X is called *temporally unsatisfiable*(or simply, *unsatisfiable*). A PLTLP formula X is called *temporally valid* (or simply, *valid*) if every possible temporal interpretation is a temporal model for X, i.e. $I(X) =$ true for any temporal interpretation I.

Examples. Let $P = \{p, q\}$, $n = 5$, $I = \{(0\{q\}), (1, (p, q)), (2, \{p, q\}), (3, \{p\}), (4, \{p\}), (5, \{p\})\}$.

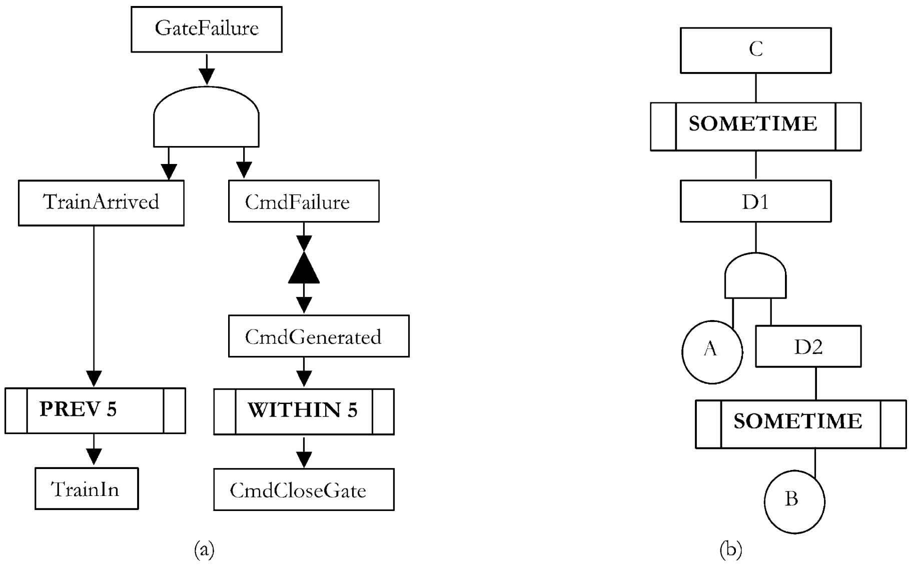


Fig. 5. Some simple TFTs.

Then $I(p \rightarrow q)$ = false and $I^3(p \rightarrow q))$ = true; so that the formula $\Box^-(p \rightarrow q)$ is temporally satisfiable but not temporally valid. Similarly, $I(\Diamond^-\Box^-p)$ = false but it would have been true if I had included $(0,\{p,q\})$; so this formula is temporally satisfiable but not temporally valid. The formula $\Box^-p \wedge \Diamond^- \neg p$ is temporally unsatisfiable. If X is a formula valid in propositional logic, then $\Box^-X$ is temporally valid in PLTLP. For example, $(p \rightarrow p)$ is valid in propositional logic; hence $\Box^-\ (p \rightarrow p)$ is temporally valid in PLTLP. Similarly, $(p \rightarrow q) \leftrightarrow (\neg p \vee q)$ is valid in propositional logic, so $\Box^-((p \rightarrow q) \leftrightarrow (\neg p \vee q))$ is temporally valid in PLTLP.

Let $S = \{F_1, F_2, \ldots, F_n\}$ be a finite set of PLTLP formulae.

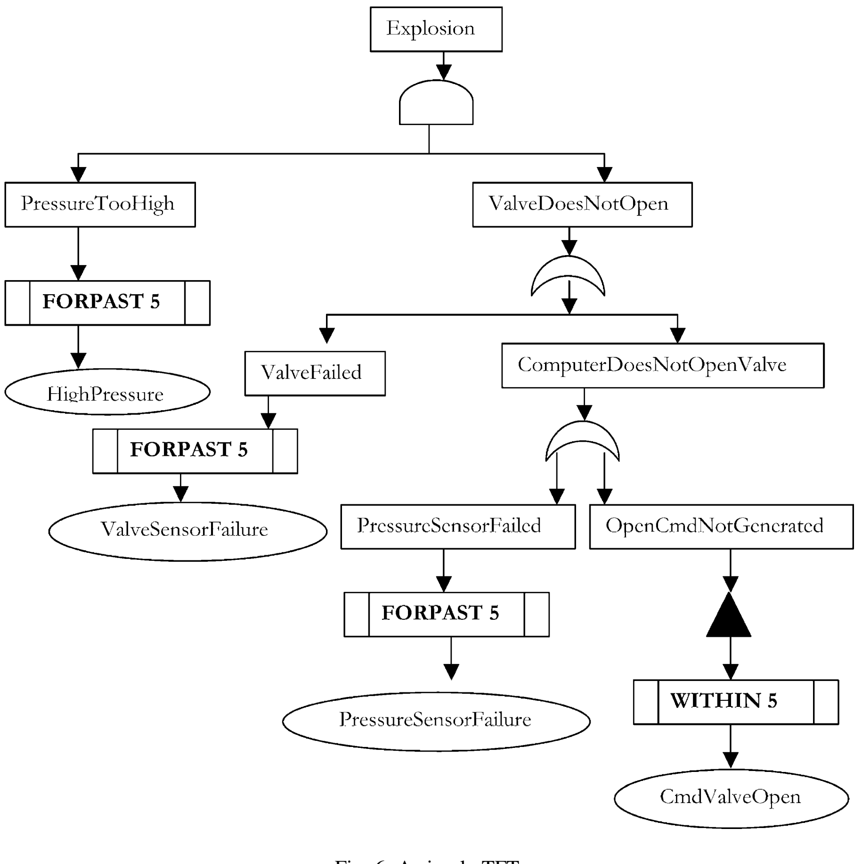
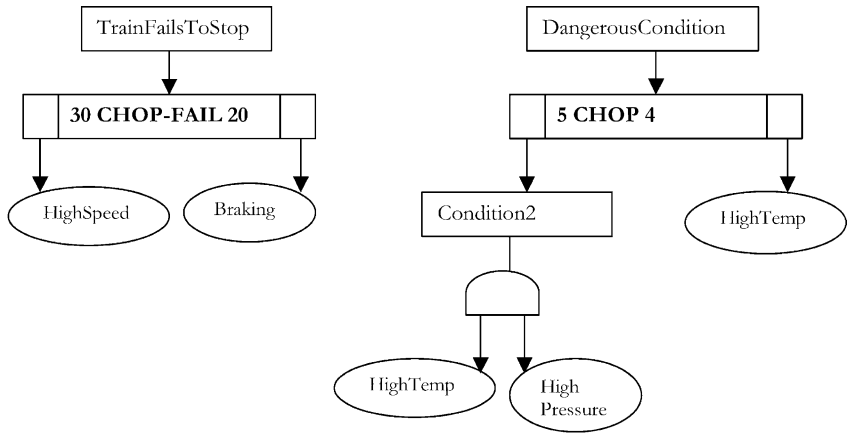


Fig. 6. A simple TFT.



(a) TFT for the failure of a train to stop (b) TFT for a dangerous condition

Fig. 7. Examples of TFT using the CHOP connectives.

Let X be another formula. We say that the PLTLP formula X is a *logical consequence* of S or S logically entails X, written $S \models X$, for any temporal interpretation I such that $I(F_1 \wedge F_2 \wedge \cdots \wedge F_n) = true$, then $I(X) = \text{true}$.

3.3. Temporal fault tree notation

We now propose a method to augment the FT notation and define a TFT notation, which includes facilities for expressing temporal events and temporal dependencies among them. We treat the logical gates as non-temporal and introduce special-purpose temporal gates, one for each of the temporal connectives introduced earlier. The temporal gates for $\Box^-$, $\Box_n^-$, $\Diamond^-$, $\Diamond^-$, $\mathbf{O}^-$ and $\mathbf{O}^-$ are unary and those for other temporal operators are binary (unlike, say, an AND gate which has an arbitrary arity in the FT notation). To avoid a profusion of special symbols in the TFT notation, we use a common symbol to denote all temporal gates—a rectangle with double vertical borders and a text label containing the name of the temporal connective and the associated parameters (integers m, n), if any. Fig. 4 depicts the use of some temporal gates and the semantics of the associated TFT in terms of a formula in PLTLP. Note that the TFT notation has retained the compositional approach of the FT notation of deriving the meaning.

4. Examples

In this section, we present some simple examples where the TFT notation is used to build FT, which depict temporal events and temporal dependencies among them. We also present a top-down declarative temporal method to 'read' and 'understand' the TFT notation.

As a first example, consider a simple controller for a gate on a railway crossing. The gate controller fails if the train arrived exactly 5 s ago but command to close the gate was not generated within the last 5 s. This situation is defined by the following PLTLP formula and depicted in the associated TFT in Fig. 5(a).

$$\text{GateFailure} = (\mathbf{O}_5^- \text{TrainIn}) \wedge (\Diamond_5^- \text{CmdCloseGate})$$

Note that we have used the directional arrows to informally indicate the time flow from the top. The current instant is associated with the top event, i.e. the fault and the causes in the past for that event are depicted along the downward flow in the TFT. A chaining of temporal gates is illustrated in a TFT in Fig. 5(b). The corresponding formula for the top node is given later which says that the top event C is true now if A is true sometime in the past and B is true sometime before that. D1 and D2 are dummy (pseudo) events.

$$\text{C} = \Diamond^-(\text{A} \wedge (\Diamond^- \text{B}))$$

As another example [22], Fig. 6 depicts a TFT for a situation where an explosion occurs because the pressure (in a chamber, say) was too high and the release valve did not operate.

Simple examples for the use of chop connectives are as follows. A failure of a train to stop can be described as HighSpeed ${}_m\mathbf{CF}_n$Braking. In this formula, the condition HighSpeed holds now and for past m seconds but the condition Braking does not hold for all n seconds before that. The condition that a period of high temperature and high pressure is preceded by a period of high temperature is described by the formula (HighTemp $\wedge$ HighPressure) ${}_m\mathbf{C}_n$ HighTemp. Associated TFTs are shown in Fig. 7.

5. Analysis of TFT

Given the observation that a top event in an FT has occurred, the next task is the identification of the possible causes for the top event to occur. The standard approaches to this task are: (i) top-down depth-first traversal of the FT till a set of basic events is reached and matched with the

Table 3
A sample trace (log) for a system

Timestamp	HighPressure	ValveReading	PressureReading	CmdGenerated
18	1	Closed	38.8	Nil
17	1	Closed	37.4	Nil
16	1	Closed	36.0	Nil
15	1	Closed	34.4	Nil
14	1	Closed	33.8	Nil
13	1	Closed	31.0	Nil
12	0	?	29.0	Nil
11	0	?	?	Nil

observations and (ii) computation of the cut-sets and path-sets. We now consider these tasks in turn.

5.1. Depth-first traversal of a TFT

In case of TFT, the TFT traversal is complicated by the fact that there are temporal gates to traverse and the observations are not a single static snapshot of the system, but instead a timestamped trace or log of the system, i.e. a temporal database. We have employed the critical idea that this temporal database forms an interpretation for the PLTLP formula represented by the given TFT. The TFT traversal involves evaluating each sub-tree rooted at a temporal gate according to the method of Section 3.2 for constructing the truth-value of the temporal formula associated with the sub-tree. This recursive top-down depth-first traversal of the TFT begins at the root and traverses the TFT downwards one level at a step. At any level, if the current gate is AND then all the events at that level are assumed to be true and the search proceeds to the next level. If the gate was OR, then each event at the level is recursively searched to see if it is true. If the gate is temporal, then the corresponding temporal formula is checked against the temporal trace or log.

Rather than formally define this procedure, we illustrate it to identify the cause for the top event in the TFT of Fig. 6. Table 3 shows a sample trace (log) for such a system. Timestamps are arbitrary; the top event is assumed to occur at the first record in Table 3, which has the timestamp = 18. HighPressure is a Boolean variable, which is true if the current pressure is above 10.0 and 0 otherwise. ValveReading is a Boolean variable indicating whether the valve is currently open or closed. PressureReading is a numeric value of the current pressure as sent by a sensor. Both ValveReading and PressureReading fields can contain a special value ? to indicate absence of a reading or occurrence of an error while taking a measurement from the associated sensor.

The top event in Fig. 6 has an AND gate connected to two events. The left event PressureTooHigh is the root of a TFT; the associated formula is PressureTooHigh $= \Box_5^-$ HighPressure. This PLTLP formula is easily verified to be true since HighPressure is true at the current time (18) and it is also true for all the past times up to 13. Similarly, the PLTLP formula ValveFailed $= \Box_5^-$ (ValveReading = ?) is easily seen to be false at time 18; note that the errors in ValveReading at time 12 and 11 do *not* affect the truth of this formula. The formula PressureSensorFailed $= \Box_5^-$ (PressureReading = ?) is also false at time = 18. However, the PLTLP formula ValveCmdNotGenerated $= \neg(\Diamond_5^-$ (CmdOpenValve)). Thus the entire PLTLP formula for the top event is true and the causes are events PressureTooHigh and OpenCmdNotGenerated.

We have a prototype Prolog [33] implementation of the depth-first traversal of a TFT. The implementation includes a set of Prolog predicates that directly implements the semantics of the PLTL (i.e. the predicates correspond to temporal gates). The temporal database (e.g. Table 3) is given as a set of application-specific Prolog facts, each of which includes a timestamp. The structure (graph) of the TFT is also provided as a separate set of Prolog facts. The user can ask various questions by asking the system to evaluate a given temporal formula at the given instant; this evaluation essentially corresponds to the depth-first traversal of the TFT. Some examples of such questions (regarding the TFT in Fig. 6) are given later. Note that the last query corresponds to evaluating the PLTL formula for node ComputerDoesNotOpenValve in Fig. 6.

Question: Is it true at instant = 18 that no command was generated at all past instants? *Answer*: yes
Question: Is it true at instant = 18 that there was high pressure for last five instants? *Answer*: yes
Question: Is it true at instant = 18 that there was high pressure for last eight instants? *Answer*: no
Question: Is it true at instant = 18 that either there was an error in pressure reading for the last five instants or no command was generated within the last five instants? *Answer*: yes

The depth-first traversal algorithm for TFT can do better than evaluating given temporal formulae at the given instant. It can also be used to examine the given temporal database and return a list of causes (in the form of a cut-set—see Section 5.2) that have caused the given top event at the given instant. If there are more than one set of causes (cut-sets) that could have caused the top event (as present in the given temporal database) then the algorithm successively reports all of them (not in any particular order). In

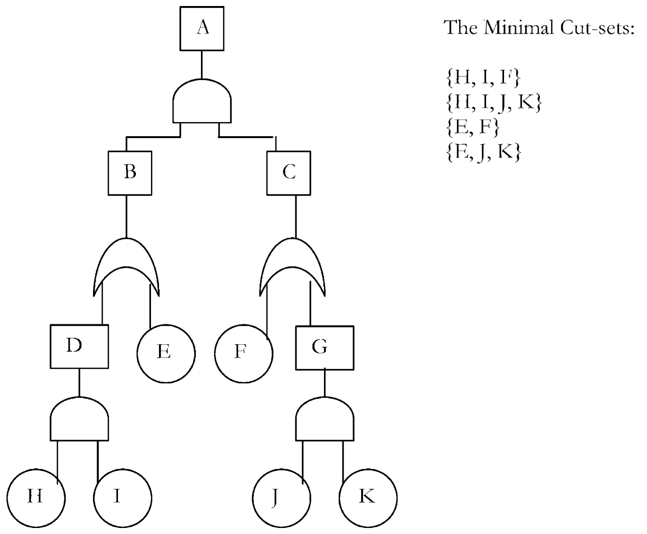


Fig. 8. A non-TFT and all its minimal cut-sets [28].

Fig. 6, the top event *Explosion* can be caused by several combinations of events (i.e. cut-sets—see Section 5.2). However, the given trace in Table 3 shows the occurrence of a specific cut-set, which is detected and reported by the depth-first algorithm for TFT traversal. Hence, in the following case, the algorithm reports that there is only one cut-set that is observed to have occurred in the given trace. If there were several possible combinations of events (i.e. cut-sets) that could have led to the occurrence of the top event in the TFT, the algorithm can backtrack and generate all of them.

Question: What caused the top event Explosion in the TFT of Fig. 6 at instant 18?
Answer: $\Box_5$ HighPressure $\wedge \neg (\Diamond_5$ CmdValveOpen), i.e. the pressure was high for the last five instants and no command was issued within the last five instants.

5.2. Fault diagnosis using cut-sets

A *cut-set* of a (non-temporal) FT is the set of those basic (leaf) events in the FT such that the occurrence of all events in the set will cause the top event to occur. A cut-set is *minimal* if no proper subset of it is itself a cut-set. Cut-sets are useful for fault diagnosis using FT. Here, given a single FT and given that fact that its top event has occurred, the algorithm finds all minimal cut-sets. Each minimal cut-set is a possible *diagnosis* for the top event. After finding all cut-sets, they are ranked according to importance or probability. Elements in each cut-set are then further ordered according to factors like importance, probability or ease of testing. A cut-set whose all elements (events) are checked to have occurred is then a *confirmed diagnosis*. There are well-known algorithms [28] that generate the set of all cut-sets for a given (non-temporal) FT (Fig. 8). We now consider the problem of generating all cut-sets for a TFT.

Essentially, there are two equivalent approaches to this task. In both approaches the aim is to make use of the existing algorithm for generating all minimal cut-sets of a

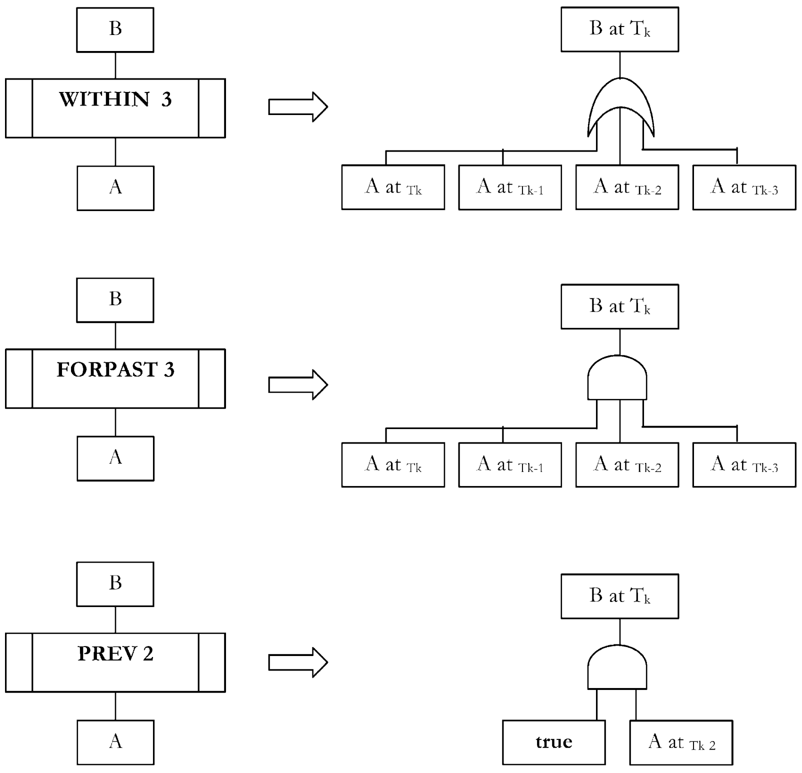


Fig. 9. Some temporal gates and their replacements.

non-TFT [28]. In the first approach, the idea is to replace every temporal gate and its associated events by a basic event labelled with a temporal formula. This process results in a non-TFT to which the usual algorithms can be applied to generate the minimal cut-sets. Note that each element in the cut-set is actually a temporal event and consequently checking whether it has occurred or not needs to be handled carefully. The programs in our implementation handle this aspect. Another complication is that the events associated with a temporal gate may not be basic events and they themselves be connected to further temporal gates (see Fig. 5(b)). In such a case the algorithm should recursively replace the temporal gates starting at the 'lowest-level' temporal gates first. This will result in a non-TFT in which some of the events are labelled with complex temporal formulae. For example, applying this algorithm to the TFT in Fig. 6 will result in essentially the same non-TFT as in Fig. 1, except that the basic events are now labelled with the temporal formulae as follows:

e_1 = High_pressure_reported_for_last_5_minutes = $\Box_5$ HighPressure
e_2 = ValveSensorFailure = $\Box_5$ ValveSensorFailure
e_3 = PressureSensorFailure = $\Box_5$ PressureSensorFailure
e_4 = OpenValveCommandNotIssued = $\neg(\Diamond_5$ Cmd ValveOpen)

Clearly, the minimal cut-sets for this non-TFT are: $\{e_1, e_2\}$, $\{e_1, e_3\}$, $\{e_1, e_4\}$. Note that each e_i is a temporal formula (as given before).

There is actually another (basically similar) approach to convert a TFT into a non-TFT. Here, we replace each temporal gate by a logical gate and the events associated with the temporal gate by one or more events. In this way, the resulting FT is non-temporal, although much larger than the associated TFT. Moreover, this approach can handle temporal events that are defined in turn in terms of temporal gates. The transformations associated with a selected subset of temporal gates (for a specific value of bounds in the temporal gates) is shown in Fig. 9. The dummy event true is always assumed to occur at every instant. The TFT in Fig. 6 can easily be transformed using this algorithm.

6. Conclusions and further work

The popular and simple FT notation is widely used for hazard analysis and to express diagnostic/maintenance knowledge. However, a problem that prevents more analytical use of FT is their lack of rigorous semantics. Users' understanding of an FT depends on the clarity and correctness of the natural language annotations used to label and describe various parts. Moreover, it is not clear how to adapt the FT notation to represent temporal relationships between faults and events in dynamic systems. In this paper, we have proposed an enhanced TFT notation by adding simple temporal gates to capture temporal dependence between events and faults. We also described a technique to perform qualitative analysis of TFT to detect the causes of the top event fault by matching the TFT with the trace (or log) of the system activities as also transformations to convert a TFT into a non-TFT so that minimal cut-sets can be generated.

We feel that the TFT notation is simple, rigorous and in the same spirit as the original FT notation. It adds considerable expressive power to allow the end-users to express temporal dependencies between events, hopefully without any training in formal temporal logic. Many different kinds of automatic analysis techniques can be devised to work with the TFT notation.

There is a large variety of temporal logics defined (see Ref. [1] for a comprehensive survey), depending on whether the time is linear/branching, dense/discrete, point-oriented/interval-oriented, etc. Limitations of propositional linear temporal logic, as used in this paper, in so far as expressive power for specifying requirements of complex system is concerned, are well known; see Ref. [1] for detailed theoretical comparisons. We have chosen this logic because it is the simplest temporal logic and we expect it to be easier for safety engineers (and not necessarily system developers) to understand and use. We feel that branching time and many other varieties of temporal logic are not always the easiest and most intuitive ones, although they may be more expressive. Moreover, actually, if we remove the PREV gates from our TFT notation, the semantics of the resulting notation can be defined in terms of any linear temporal logic, since most of them provide past-oriented operators like UNTIL-PAST, SOMETIME, ALWAYS-PAST, etc. Of course, we could also have provided more expressive facilities, borrowing from some temporal logics; for example, the 'freeze context' operator from Ref. [9]. We could also have chosen an interval oriented temporal logic [2,5] but the resulting FT notation would need associating quantitative information with the FT nodes. We wanted to retain the essentially qualitative flavour of the FT notation and not add too many complex quantitative facilities to it (see Refs. [29,30]) although many of them have obvious advantages. However, we have chosen what we hope to be a sufficiently useful subset of temporal operators.

As a final point, we wish reiterate that the intended use of the enhance FT notation is really 'post facto', i.e. after the system is developed. Hence, we have assumed that the system is observed by means of 'periodic sampling'. This is the view for most operational and maintenance/support engineers and consequently the users of our TFT notation. In such a case, whether or not the underlying model of the time is dense, the 'observed' model of the time is certainly linear and discrete. That is one more reason why we have adopted a linear discrete-time temporal logic. In such a case, one of the problems that need to be solved is the identification of the causes of the observed hazard (top event) by examining the trace/log of the system (which is necessarily a periodically sampled timestamped database, as in a telemetry data, for example). We have treated these problems in Section 5.

We repeat that we have no intention to re-invent any notation for specification of real-time safety-critical system. In particular, there is no intention to re-invent timed Petri nets or timed automata. Both of these are 'executable' or operational notations whereas temporal logic presents a more abstract view—although of course there is a relation between them (e.g. temporal logic semantics can be defined in terms of say Buchi automata). Temporal logic also does not provide any explicit facilities (as in Petri nets) for asynchronous, distributed, concurrent events. That is why we do not consider the problem of translating the TFT into either timed Petri nets or timed automata. We have only focused on the problems of (i) what kind of temporal facilities are needed by the users of the FT notation (ii) how the semantics of this enhanced TFT notation can be defined, keeping as close to the logical framework of the untimed FT notation as possible and (iii) what kind of analysis can be performed on the enhanced FT notation.

For further work, one can investigate the suitability of the TFT notation to work with interval oriented temporal logics like the Duration Calculus [2] or Interval Temporal Logic [5]. Synthesis of TFTs is also an interesting problem. We are investigating the methods to adapt the standard fault diagnosis algorithms based on FT to work with the TFT notation. We are also investigating the definitions of fuzzy TFT where each of the propositional symbols associated with the leaves of the TFT are actually fuzzy propositions whose truth-values are real numbers from the interval [0,1]. These propositions are useful to model non-crisp observations like 'the temperature is too high' or the 'pressure is increasing rapidly'. The semantics of the temporal gates also then need to be modified accordingly, to work with such a fuzzy temporal logic [32]. One further work concerns investigating how probabilistic extensions can be added in the TFT notation.

Acknowledgements

I sincerely appreciate the support of Prof. Mathai Joseph. I thank the referees for their useful comments. Thanks also to Dr Manasee Palshikar for her dedication, persistence and hopes.

References

[1] E.A. Emerson, Modal and temporal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, Elsevier, Amsterdam, 1990, pp. 995–1072.

[2] Z. Liu, Specification and verification in duration calculus, in: M. Joseph (Ed.), Real-time Systems: Specification, Verification and Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1996, pp. 182–228.

[3] L. Lamport, TLA in pictures, IEEE Trans. Software Engg 21 (9) (1995) 768–775.

[4] Y.S. Ramakrishna, L.K. Dillon, L.E. Moser, P.M. Melliar-Smith, G. Kutty, A real-time temporal logic and its decision procedure, Proceedings of FSTTCS 1993, LNCS 761, Springer, Berlin, pp. 173–190.

[5] J.F. Allen, G. Ferguson, Actions and events in interval temporal logic, J. Logic Comput. 4 (5) (1994) 531–579.

[6] A. Pneuli, Temporal Logic of programs, Proceedings of 18th FOCS, 1977.

[7] R. Mattolini, P. Nesi, An interval logic for real-time system specification, IEEE Trans. Software Engng 27 (3) (2001) 208–227.

[8] R. Koymans, Specifying real-time properties with metric temporal logic, Real Time Syst. J. 2 (1990) 255–299.

[9] R. Alur, T. Henzinger, A really temporal logic, J. ACM 41 (1) (1994) 181–204.

[10] R. Alur, D. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (1994) 183–235.

[11] X. Nicollin, J. Sifakis, S. Yovine, Compiling real-time specifications into extended automata, IEEE Trans. Software Engng 18 (9) (1992) 794–804.

[12] D. Harel, E. Gery, Executable object modeling with statecharts, IEEE Comput. 30 (7) (1997) 31–42.

[13] F. Jahanian, A.K. Mok, Modechart: a specification language for real-time systems, IEEE Trans. Software Engng 20 (12) (1994) 933–947.

[14] M. Felder, D. Mandrioli, A. Morzenti, Proving properties of real-time systems through logical specifications and Petri net models, IEEE Trans. Software Engng 20 (2) (1994) 127–141.

[15] B. Barthomieu, M. Diaz, Modeling and verification of time-dependent systems using timed Petri nets, IEEE Trans. Software Engng 17 (3) (1991) 259–273.

[16] C. Gezzi, D. Mandrioli, S. Morasca, M. Pezze, A unified high-level Petri net model for time-critical systems, IEEE Trans. Software Engng 17 (2) (1991) 160–172.

[17] I. Suzuki, Formal analysis of alternating bit protocol by temporal Petri nets, IEEE Trans. Software Engng 16 (11) (1990) 1273–1281.

[18] E.Y.T. Juan, J.J.P. Tsai, T. Murata, Y. Zhou, Reduction methods for real-time systems using time delay Petri nets, IEEE Trans. Software Engng 17 (5) (2001) 422–448.

[19] US Nuclear Regulatory Commission, Fault Tree Handbook, NUREG-0492, Washington, DC, 1981. See also http://www.nrc.gov/NRC/NUREGS/ABSTRACTS/sr0492.html.

[20] K. Sullivan, J.B. Dugan, D. Coppit, The Galileo fault tree analysis tool, Proceedings of FTCS 1999, pp. 232–235.

[21] Web-sites of some commercial vendors that offer tools related to fault trees: http://isographdirect.com/indexft.html, http://www.iitr.org/cgi-ac/ProdDescription?FTA, http://www.relexsoftware.com/products/faulttree.html,http://www.sydvest.com/Products/Cara/c-ft.html.

[22] N.G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, Reading, MA, 1995.

[23] N.G. Leveson, S.S. Cha, T.J. Shimeall, Safety verification of Ada programs using software fault trees, IEEE Software 8 (4) (1991) 48–59.

[24] S.-Y. Min, Y.-K. Jan, S.D. Cha, Y.R. Kwon, D.-H. Bae, Safety verification of Ada95 programs using software fault trees, Proceedings of SAFECOMP 1999, pp. 226–238.

[25] S. Liu, J.A. McDermid, A model-oriented approach to safety analysis using fault trees and a support system, J. Syst. Software 2 (1996) 151–164.

[26] P. Liggesmeyer, M. Rothfelder, Improving system reliability with automatic fault tree generation, Proceedings of FTCS 1998, pp. 90–99.

[27] A. Bobbio, L. Portinale, M. Minichino, E. Ciancamerla, Comparing fault trees and Bayesian networks for dependability analysis, Proceedings of SAFECOMP 1999, pp. 310–322.

[28] N. Viswanadham, V.V.S. Sarma, M.G. Singh, Reliability of Computer and Control Systems, North-Holland, Amsterdam, 1987.

[29] J. Magott, P. Skrobanek, A method of analysis of fault trees with time dependencies, Proceedings of SAFECOMP 2000, pp. 176–186.

[30] G. Helmer, J. Wong, M. Slagell, V. Honavar, L. Miller, R. Lutz,

Software fault tree and colored Petri nets-based specification, design and implementation of agent-based intrusion-detection systems, Proceedings Symposium on Requirements Engineering for Information Security, Indianapolis, IN, USA, March, 2001.

[31] K.M. Hansen, A.P. Ravn, V. Stavridou, From safety analysis to software requirements, IEEE Trans. Software Engng 24 (7) (1998) 573–584.

[32] G.K. Palshikar, A fuzzy temporal notation and its application to specify fault patterns for diagnosis, Pattern Recognition Lett. 22 (2001) 381–394.

[33] W.F. Clocksin, C.S. Mellish, Programming in Prolog″, 3/e, Springer, Berlin, 1989.