Available online at www.sciencedirect.com
SCIENCE DIRECT
Reliability Engineering and System Safety 87 (2005) 121–131
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

Fault tree construction of hybrid system requirements using qualitative formal method

Jang-Soo Lee[a], Sung-Deok Cha[b,*]

[a]Instrumentation and Control, Human Factors Division, Korea Atomic Energy Research Institute, 150, Duckjin-dong, Yuseong-gu, Daejeon 305-353, South Korea
[b]Computer Science Division, EECS Department and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology, 373-1, Kusong-dong, Yuseong-gu, Daejeon 305-701, South Korea

Accepted 26 April 2004

Abstract

When specifying requirements for software controlling hybrid systems and conducting safety analysis, engineers experience that requirements are often known only in qualitative terms and that existing fault tree analysis techniques provide little guidance on formulating and evaluating potential failure modes. In this paper, we propose Causal Requirements Safety Analysis (CRSA) as a technique to qualitatively evaluate causal relationship between software faults and physical hazards. This technique, extending qualitative formal method process and utilizing information captured in the state trajectory, provides specific guidelines on how to identify failure modes and relationship among them. Using a simplified electrical power system as an example, we describe step-by-step procedures of conducting CRSA. Our experience of applying CRSA to perform fault tree analysis on requirements for the Wolsong nuclear power plant shutdown system indicates that CRSA is an effective technique in assisting safety engineers.

Keywords: Requirements; Software safety; Fault tree; Formal method; Hybrid system

1. Introduction

A hybrid system consists of a discrete program within a continuous plant which is governed by the laws of physics. Ubiquitous physical processes, from car to aircraft, are controlled by such programs. Obviously, safety is vital importance for the hybrid systems. The traditional safety analysis methods [1,15,18], however, allow us to analyze the safety of the control software and the plant in separate. Primary goal of safety analysis is to demonstrate that the system as a whole will not result in exhibiting hazardous behavior. Since safety is the system property, proper software safety analysis must always be conducted within the context of the environment which it is required to control. That is, safety analysis must demonstrate the truth of the proposition (P and C → Sp) where P, C, and Sp refer to the behavioral model of the plant, controller, and requirements for control software, respectively.

Unfortunately, during early phases of identifying and analyzing requirements for a hybrid system, requirements are usually known only in qualitative terms and stated in natural language. For example, the system might be required to open the relay once the current charge level of the solar battery exceeds certain threshold value, and the exact value may not be known until the later phases of development. To perform adequate safety analysis, one must be able to perform causal and backward analysis using qualitative information such as signs, relative magnitudes, comparison against landmark values, and the direction of change in values.

Fault tree analysis is arguably the most widely used safety analysis technique in industry. Advantages include ease of understanding due to graphical representation, ability to flexibly describe various failure scenarios such as hardware failures, software errors, operator mistakes, poorly designed human interfaces, or events in

* Corresponding author. Fax: +82-42-868-8916.
E-mail addresses: jslee@kaeri.re.kr (J.-S. Lee), cha@cs.kaist.ac.kr (S.-D. Cha).

0951-8320/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2004.04.012

the environment external to the system. On the other hand, the technique itself is nothing more than a notation to systematically organize analysis done by safety engineers [1]. Therefore, effectiveness of the technique is highly dependent on the ability of safety engineers, and the fault trees can be different from one safety engineer to another even though they are based on the same information. In addition to informality inherent in fault tree analysis, difficulty of building fault trees during early phases of a hybrid system development is compounded because one must analyze complex interaction patterns involving discrete and continuous events stated in abstract terms.

In Refs. [2,3], we proposed a qualitative formal method (QFM) as a means of introducing rigor when building models corresponding to the behavior of the plant and the controller. In particular, we adapted research results in artificial intelligence, qualitative physics, to formally express qualitative terms. Behavioral models of the control software and the plant are expressed in notations known as the Causal Functional Representation Language (CFRL) [4] and Compositional Modeling Language (CML) [5]. These languages allow one to express inherently qualitative state changes such as the temperature rising and the possible consequences in a manner where formal and causal reasoning can be performed. Outcome of the QFM process is called the state trajectory which is analogous to reachability graph of the Petri Net models. State trajectory can be automatically generated using Device Modeling Environment (DME) tool [6], and it contains useful information to assist safety engineers in building fault trees in a systematic manner.

In this paper, we extend the QFM process and describe how to generate fault trees. We refer to the extended technique as CRSA. Although the process is not fully automated, much of fault tree skeleton can be automatically derived from information on causal relationship captured in the state trajectory and qualitative formal specification. We describe step-by-step procedures of applying CRSA using a simplified Electrical Power System (EPS) for a satellite. EPS system has been used in some safety literature as the case study [6].

The rest of our paper is organized as follows. In Section 2, we briefly review related work. Our focus is primarily on safety analysis, and our review is limited to fault tree analysis technique since it is the most widely used technique in industry. The EPS system is also briefly explained. In Section 3, we briefly review QFM process and discuss in detail step-by-step procedures of applying CRSA and building fault trees. Section 4 concludes the paper.

2. Backgrounds

2.1. Fault tree analysis

Several techniques have been proposed to perform safety analysis. Most of the safety analysis techniques were initially proposed to perform backward and causality analysis at the system level, but some have been adapted to software. As discussed in Ref. [7], an authoritative survey on the subject, there are many safety analysis techniques including checklists, fault tree analysis, event tree analysis, cause-consequence analysis, failure modes and effects analysis, failure modes, effects, and criticality analysis, and hazards and operability analysis.

Fault tree analysis has been the subject of most safety research in the past. Leveson and Harvey [18] adapted system-level fault trees to software analysis, and they used sequential program structures and their failure semantics to derive fault tree nodes. The technique has since been applied to all phases of software development lifecycle, including requirements engineering [8,9], design [10–12], and concurrent and object-oriented code [13–17]. In Ref. [8], specifications are written in real-time interval logic, and fault trees are derived from temporal logic formula capturing causal relationship between states. Gorski [19], similar in approach to Ref. [8], attempted to formalize fault trees. Subramanian discussed how to analyze software safety in the requirements engineering and design phases using Statecharts [12]. When evaluated in the context of performing safety analysis for a hybrid system, past research on fault tree analysis is inadequate due to following reasons:

1. Only the discrete aspects of behavioral model can be modeled and analysed.
2. Guidelines or heuristics used in fault tree generation do not work well in early phase of requirements engineering when abstract requirements are given.

2.2. Electrical power system

We use a simplified version of EPS for a satellite as an illustrative example because it has been used in safety literature [6,20]. EPS, whose schematic diagram is shown in Fig. 1, is supposed to supply constant level of electricity to the satellite's other subsystems. The solar array generates electricity when the satellite is facing the sun, supplying power to the load and recharging the battery. The battery is

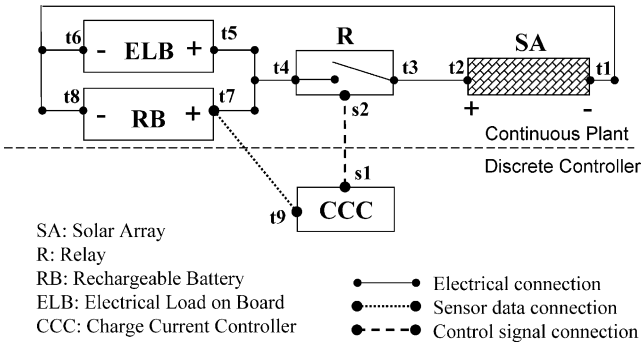

Fig. 1. Electrical power system (EPS).

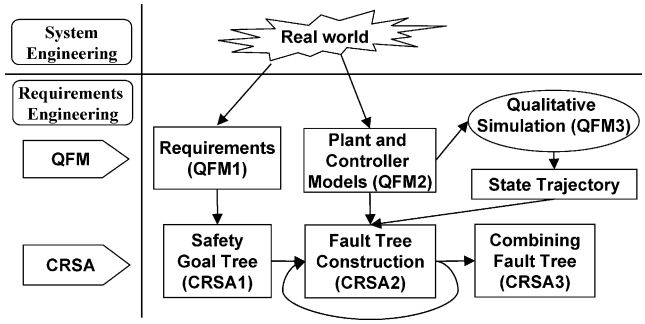

Fig. 2. Safety analysis process with QFM.

considered a constant voltage source when it is charged between 6 and 30 A h. When the charge level is outside this range, the voltage output increases or decreases as the battery charges or discharges.

Since battery can be damaged when charged beyond its capacity, the charge current controller (CCC) must open the relay when voltage exceeds the threshold value. The controller senses the current voltage level using a sensor connected to the positive terminal, t7, of the battery. When the voltage is greater than 33.8 V, the controller must turn on the relay R. When the relay is energized, it opens and breaks the electrical connection to prevent further charging of the battery, thereby switching the current source for the load from the solar array to the battery. When the relay is open or during an eclipse, the battery's charge level starts to decrease. As the battery becomes undercharged, the voltage decreases. When it reaches 31.0 V, the CCC is required to turn relay R off to close it.

3. Causal requirements safety analysis process

As noted earlier, CRSA is an extension of the QFM process proposed earlier [2,3], and the overall process is shown in Fig. 2. The QFM process adapts notations developed in artificial intelligence research, qualitative physics in particular, by Iwasaki et al. [4–6]. Step-by-step procedure of the QFM process is as follows:

3.1. QFM1: system structure and causality specification

CFRL [4] notation captures the system components and the goal properties (e.g. safety requirements) the system must satisfy. For example, requirements for the CCC in Fig. 3 include the system structure and the causal relations of the safety requirements.

Although detailed discussion on the syntax and semantics of CFRL is beyond the scope of this paper, one should note that it describes various system components (e.g. relay, battery, etc.) and how various components are physically connected. For example, the first line in the condition clause, placed inside the box in italic font for illustration, describes that the minus terminal of the solar array (t1 in Fig. 1) is electrically connected to the minus terminal of the rechargeable battery (t8 in Fig. 1). Safety properties the system must satisfy are expressed in a notation called Causal Process Description (CPD) shown in Fig. 4. For example, CPD1 states that the state N1, 'the sun is shining', immediately causes the states N2 and N3 to occur by function of the solar array. State N2 indicates that the current of the plus terminal is positive in value. Likewise,

```
EF: F1          /* function name of the charge current controller */
DF:             /* system entities and their physical connection relations */
  Components:
    (?relay Relay)
    (?battery Rechargeable-battery)
    (?solar-array Solar-array)
    (?ccc Charge-current-controller)
  Conditions:
    (Electrically-connected (Minus-terminal ?solar-array) (Minus-terminal ?battery) )
    (Electrically-connected (Plus-terminal ?solar-array) (Electrical-terminal-one ?relay) )
    (Electrically-connected (Electrical-terminal-two ?relay)
    (Plus-terminal ?battery) )
    (Electrically-connected (Plus-terminal ?battery) (Voltage-sensing-terminal ?ccc))
    (Electrically-connected (Signal-terminal ?ccc) (Signal-terminal ?relay) )
    (Same (Plus-terminal ?eps) (Plus-terminal ?battery))
    (Same (Minus-terminal ?eps) (Minus-terminal ?battery))
CF:             /* context of the system */
  Objects: nil
  Conditions: nil
GF:             /* goal of system behavior */
   (ALWAYS
      (AND
        (OR (AND (Shining-p ?sun) (Closed-p ?relay) CPD1)
              CPD2)   /* subgoal 1: power sources normal operation */
        (IMPLIES (AND (>= (Electromotive-force ?battery) 33.8)
           (Closed-p ?relay)) CPD3) /* subgoal 2: open R when over C */
        (IMPLIES (AND (=< (Electromotive-force ?battery) 31.0)
           (Open-p ?relay)) CPD4))) /* subgoal 3: close R when under C */
```

Fig. 3. System structure and causality specification.

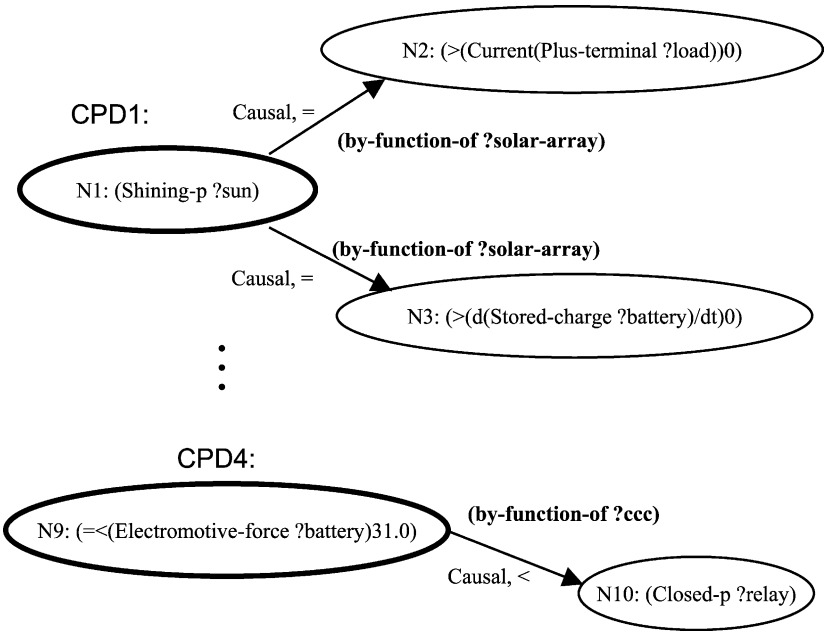

Fig. 4. Causal process description (CPD).

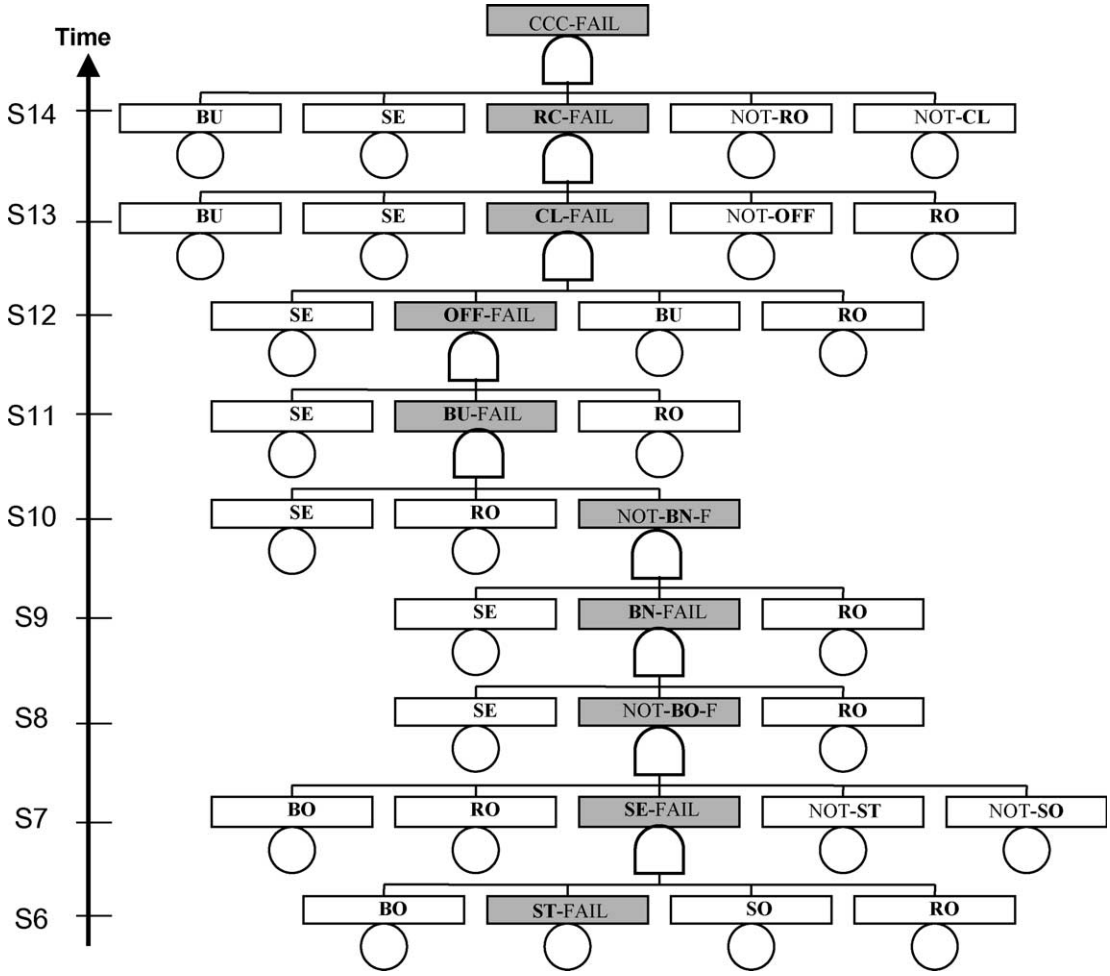

Fig. 5. CRSA2: essential fault tree of the top event 2.

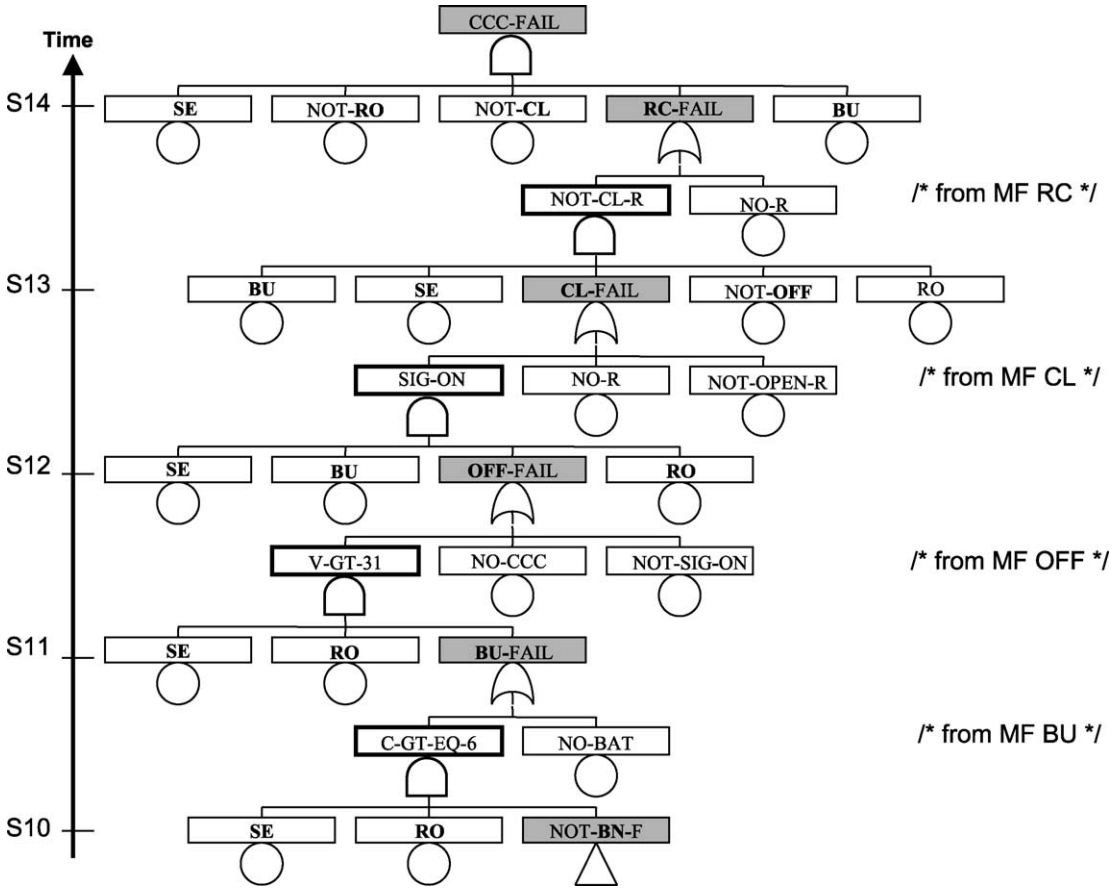

Fig. 6. CRSA2: fault tree breakdown.

N3 captures the phenomenon that the stored charge of the battery starts to increase. It also specifies properties the system must satisfy in the GF section. In this example, it states that CPD3 and CPD4 must always be satisfied and that either CPD1 or CPD2 must hold, too.

3.2. QFM2: specification of physical phenomenon

Behavioral models of the plant, controller, and their interface are written in CML [5]. CML is a declarative model-building language for logically specifying the symbolic and mathematical properties of the structure and behavior of physical systems. CML specifies abstract behavior of the plant using qualitative differential equations and causality relation. The physical situation is modeled as a general-purpose domain theory as a collection of model fragments, each of which represents a physical object or a conceptually distinct physical phenomenon. A model fragment[1] specifies a set of conditions under which a phenomenon would occur and the consequences. The conditions are indicated as Pm and specify a set of instances of object classes that must exist, called Participants, and a set of relations, called Conditions, that must hold among those objects and their attributes. Their consequences are indicated as Cm for model fragments representing continuous phenomena and as Dm for model fragments representing discrete events. For example, battery is said to be in the normal and operating range if the current charge level is between 6.0 and 30.0 A h. That is, the voltage (Electro Motive Force) of the battery maintains a constant value of 33.0 V. When the charge falls below 6.0 A h, the voltage produced decreases as it is discharged.

3.3. QFM3: qualitative simulation

DME [6] performs qualitative simulation of the system behavior in the context of the plant model and produces state trajectory[2] shown below. To perform simulation, causal ordering among variables in the qualitative equations must be decided. Note that state trajectory describes

[1] Complete CML specifications are available in Ref. [6]. In this paper, we include only the model fragments that are needed to understand CRSA example shown later.

[2] The set of all active model fragments in each state is actually much larger, but since most of them represent components, terminals, junctions, etc. and are active throughout the simulation, we show only the ones that change their activation status at some point. In addition, the state variables of I2, I3, I7 mean the current value at the terminals t2, t3, and t7, respectively. V7 is the voltage at terminal t7. *c* denotes the capacity of the battery.

the values of state variables in terms of their magnitudes or direction of changes (e.g. rise or fall). In a state where their derivative is undefined, the sign is shown as x. The leftmost column of the table shows the set of active model fragments in each state in bold. Similarly, deactivated model fragments are shown in underline.

In the initial state s0, since the sun is up and the relay is closed, the charge-level starts to increase. When it eventually reaches 30 A h, thereby entering the state s1, the battery enters the over-charged state, and the voltage level starts to rise. When it reaches 33.8 A h in s3, the CCC changes the signal to ON (s4), and R opens (s5). At this point, the solar array stops generating current, and the battery starts to discharge. It should be noted that state trajectory contain sufficient information on the causes and consequences, and CRSA technique utilizes such information to investigate failure modes. Detailed step-by-step procedure of CRSA technique is as follows.

3.4. CRSA1: Top-level event identification

Fault tree analysis begins by identifying the top-level events whose consequences are considered hazardous and therefore must be avoided. Safety goal tree, specified in the GF section of the CFRL specification in the QFM1 process and shown in Fig. 3, is used to identify the candidates. Although notations, especially the use of gate symbols, are similar to fault trees, it does not state potential failure modes, causes, and the relationship among them. Rather, it describes conditions that must hold for the system to satisfy the goal stated in the root node. Some are hardware related goals, and they are excluded because our focus is primarily on software. This step results in identifying software related goals, and the negation of such goals, therefore, becomes the candidates of the fault tree root node. The process of developing safety goal tree is not automated and requires domain-specific knowledge to determine which goals are to be satisfied by which component.

Upon completion of the first step of the CRSA process, in the case of the EPS, two top-level events of the fault tree are identified:

- CCC-WORK-OP failure where CCC fails to generate the open signal when battery over charged.
- CCC-WORK-CL failure where CCC fails to generate the close signal when battery under charged.

3.5. CRSA2: fault tree construction

Once top-level nodes of the fault tree are identified, causal ordering information included in the state trajectory as well as the causality information specified in the model fragments are used to investigate credible causes and the relationship among them. State trajectory is especially useful for this purpose because it contains causal—though qualitative—ordering information explaining the hybrid system behavior. The detail steps to construct the fault tree are as follows.

Identify the state in the state trajectory which corresponds to the top-level fault tree node. This decision is made by a domain expert using information relevant to the top-level failure scenario. Structural information such as components, terminals, and junctions are ignored. In the case of the CCC-WORK-CL failure, state S14 is identified as the initial state, and the all the states from the initial state leading up to S14 are included in what we call the essential fault tree. The event names in the fault trees of Figs. 5 and 6 and their explanations are in Table 1.

Find out which software components cause the system to enter the ‘failure state’. Examination of the state s14 indicates that RO and CL model fragments were deactivated and that RC model fragment was activated. Based on the engineering judgment, BU and SE model fragments can be assumed to be irrelevant. All the components in S13 (BU, SE, CL-FAIL, NOT-OFF, and RO) have involved in changing the system states from S13 to S14 at the same time (AND condition). Of course, each component in S13

Table 1
Event description

No.	Event name	Event explanation	Source
1	BN-FAIL	Battery-normal-operating-range (BN) failed	
2	BO	Battery-over-charged (BO)	
3	BU	Battery-under-charged (BU)	
4	BU_FAIL	Battery-under-charged (BU) failed	
5	C-GT-EQ-6	Charge-level was greater than or equal to 6 A h	BU
6	CCC-FAIL	Charge current controller failed	
7	CL-FAIL	Relay-closing (CL) failed	
8	NO-BAT	Rechargeable battery did not exist	BU
9	NO-CCC	Charge current controller did not exist	OFF
10	NO-R	Relay did not exist	RC
11	NOT-BN-F	Battery-normal-operating-range (BN) deactivation failed	
12	NOT-CL	Relay-closing (CL) deactivated	
13	NOT-CL-R	Relay not closed	RC
14	NOT-OFF	Turn-R-off (OFF) deactivated	
15	NOT-OPEN-R	Relay not opened	CL
16	NOT-RO	Relay-open (RO) deactivated	
17	NOT-SIG-ON	Signal of CCC was not on	OFF
18	NOT-SO	Solar-array-in-open-circuit (SO) deactivated	
19	NOT-ST	Sun-set (ST) deactivated	
20	OFF-FAIL	Turn-R-off (OFF) failed	
21	RC-FAIL	Relay-closed (RC) failed	
22	RO	Relay-open (RO)	
23	SE	Solar-array-in-eclipse (SE)	
24	SE-FAIL	Solar-array-in-eclipse (SE) failed	
25	SIG-ON	Signal of relay was on	CL
26	SO	Solar-array-in-open-circuit (SO)	
27	ST-FAIL	Sun-set (ST) failed	
28	V-GT-31	Voltage of CCC was greater than 31 V	OFF

has a different priority to be analyzed according to the degree of how directly to change the states. The event related to the fail of the activation, CL-FAIL is the immediate event with highest priority to cause the fail of upper event, RC-FAIL. The event to fail the required deactivation, NOT-OFF, will be next priority event. The other events, BU, SE, and RO are just the situational events with low priority even though they involved in the transition of the states from S13 to S14. Therefore, the event RC-FAIL has been selected because it is the immediate cause with the highest priority to the top event, and is highlighted in the essential fault tree of Fig. 5. Likewise, from the state trajectory of Table 2, we can identify that CL model fragment in the predecessor state, S13, has been activated, and OFF model fragment has been deactivated. Therefore, the components in S13 (BU, SE, CL-FAIL, RO, and NOT-OFF) should be connected to key component of S14 (RC-FAIL) in Fig. 5.

Further elaboration of the immediate causes. In the case of state S14, according to the model fragment on relay-closed, denoted RC in Fig. 7, (Relay ?r) ^ (Closed-p ?r) condition must hold for the relay to be closed. Therefore, if we were to assume that failure occurred involving relay-closed model fragment, either (Relay ?r) or (Closed-p ?r) must have failed, and we capture such knowledge using the OR gate. This elaboration process is graphically shown in Fig. 6.

It should be noted that causes unrelated to software are ignored. Such decision is justified because we are primarily interested in investigating if and how failures of the software requirements may contribute to the occurrence of hazardous system states. In this example, conditions, No-Relay (No-R), Not-Opened-Relay (Not-Open-R), and No-Battery (No-BAT) are assumed to be unrelated to software and therefore ignored. This step is repeated until the nodes are drawn from the conditions of the model fragments for control software. The fault trees in Figs. 5 and 6 are incomplete ones. They need a refinement process to find the causes by the multiple events. The deactivated components should also be studied at the next iteration of the steps in CRSA2. CRSA cannot generate the complete fault trees automatically, but just guides the analyst to construct the fault trees by providing the basic fault trees of Figs. 5 and 6. Usually, during the refinement process, analyst can detect

Table 2
State trajectory of EPS

Active Models	state	Sun	Relay	Signal	I2	I5	I7	V7	c	Time
BN, SG, RC	**S0**	shine	close	Off	- std	+ std	+ std	33.0 std	6-30 inc	0-60 inc
BN, SG, RC	**S1**								30 inc	
BO, SG, RC	**S2**							33.0-33.8 inc	30-inf inc	
BO, SG, RC, **ON**	**S3**							33.8 x		
BO, SG, RC, ON, **OP**	**S4**			on						
BO, SG, RC, **SO, RO**	**S5**		open		0 std		- std	33.0-33.8 dec	30-inf dec	
BO, **ST**, SO, RO	**S6**									60 inc
BO, ST, **SE**, SO, RO	**S7**	~shine								60-100 inc
BO, SE, RO	**S8**							33.0 dec	30 dec	
BN, SE, RO	**S9**							33.0 std	6-30 dec	
BN, SE, RO	**S10**								6 dec	
BU, SE, RO	**S11**							31.0-33.0 dec	0-6 dec	
BU, SE, RO, **OFF**	**S12**							31.0 dec		
BU, SE, RO, OFF, **CL**	**S13**			off				0-31.0 dec		
BU, SE, RO, CL, **RC**	**S14**		close							
BU, SE, **RST**, RC	**S15**									100 x
BU, SE, RST, **RISE**, RC	**S16**									0 x
BU, SE, RISE, **SG**, RC	**S17**	shine			- std		+ std	0-31.0 inc	0-6 inc	0 inc
BU, SG, RC	**S18**									0-60 inc
BU, SG, RC	**S19**							31.0 inc		

Behaviors of battery
Battery-normal-operating-range (BN)
Pm: (Rechargeable-battery ?b) 6.0 amp-hours < (Charge-level ?b) < 30.0 amp-hours
Cm: (EMF ?b) = 33.0 volts /* the voltage of the battery maintains a constant value of 33.0 volts */
...
Battery-under-charged (BU)
Pm: (Rechargeable-battery ?b) (Charge-level ?b) < 6.0 amp-hours
Cm: (EMF ?b) = M- ((Charge-level ?b)) /* the voltage produced decreases monotonically as it is discharged */

Behaviors of the solar array
Solar-array-generating (SG)
Pm: (Sun ?s) (Shining-p ?s) (Solar-array ?a) (In-closed-circuit ?a)
Cm: (Current-thru-terminal (Plus-terminal ?a)) = -
…

Behaviors of the relay
Relay-closed (RC)
Pm: (Relay ?r) (Closed-p ?r)
Cm: (voltage-at-terminal (electrical-terminal-one ?r)) = (voltage-at-terminal (electrical-terminal-two ?r)) (current-thru-terminal(electrical-terminal-one ?r)) = - (current-thru-terminal (electrical-terminal-two ?r))
Relay-closing (CL)
Pm: (Relay ?r) (Open-p ?r) (Signal-on (Signal-terminal ?r))
Dm: (Closed-p ?r)
...
Turn-R-off (OFF)
Pm: (Charge-current-controller ?ccc) (Signal (Signal-terminal ?ccc)) = on (Voltage-at-terminal (Voltage-sensing-terminal ?ccc)) < 31.0 volts
Dm: (Signal (Signal-terminal ?ccc)) = off

Fig. 7. Specifications of plant and controller models of EPS.

the relationship among the physical hazard of the plant (P), the logical faults of the controller (C), and the requirements (Sp).

3.6. CRSA3: fault tree composition

The last step is the composition of fault trees using negation of goal trees and individual fault trees generated in the step CRSA2. For example, in the safety goal tree shown in Fig. 8, satellite would be powered as needed if power sources work properly and if the controller operates correctly as required. Therefore, the top level fault tree, shown in Fig. 9, indicates that satellite may not be properly powered if either power sources fail or controller fails. Similarly, although unrelated to software requirements, power sources will work properly if either solar array is in operation or battery operates in a normal range. (See the node P-SRC-WORK in Fig. 8.) Therefore, in the fault tree, AND gate is used in

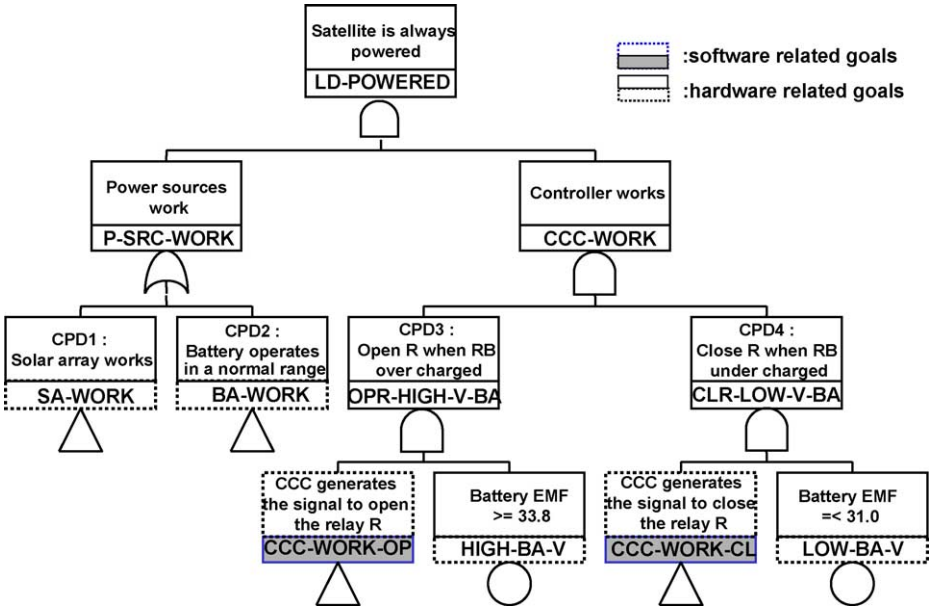

Fig. 8. CRSA1: Safety goal tree.

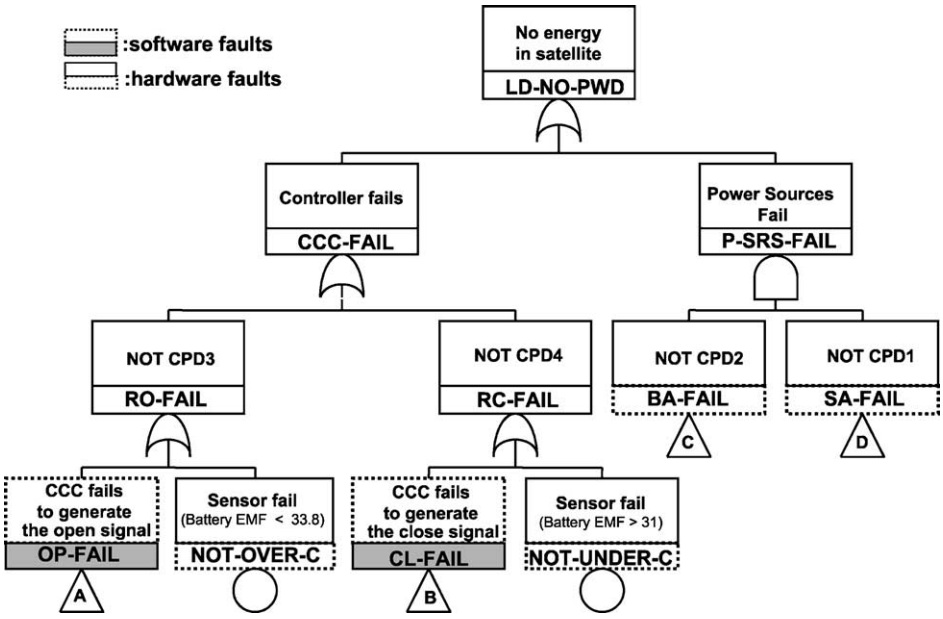

Fig. 9. CRSA3: fault trees composition.

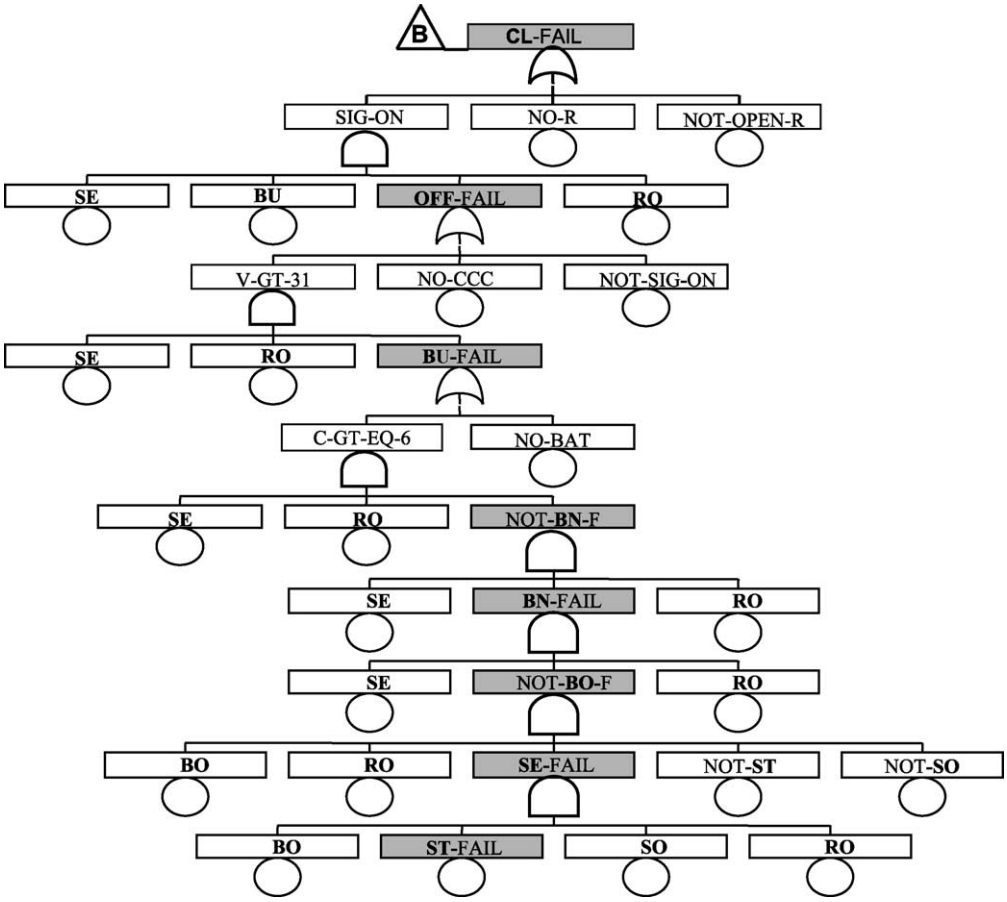

Fig. 10. CRSA3: fault trees composition (cont.).

the P-SRC-FAIL node. Negation of safety goal tree provides templates into which fault trees generated from the step CRSA2 can be integrated. For example, as shown in Fig. 9, node labeled CL-FAIL is determined to be software-related, and the fault tree generated in the previous step is attached using standard connective symbols [1]. The resulting fault trees are shown in Figs. 9 and 10.

4. Conclusions

In this paper, we have presented a technique to systematically generate fault trees from qualitative specification available during early phases of requirements engineering. Our technique is well-suited to the development of hybrid real-time systems in that the behavioral models of the controller (software) as well as the plant (environment) behavior are included in the analysis [21]. Such comprehensive modeling and analysis is essential to safety engineers since safety property must be defined and verified within the overall system's context. Isolated analysis of software requirements alone would be inadequate.

Our approach extends the QFM framework we had previously proposed. While previous research demonstrated that qualitative physics formalism, developed in artificial intelligence research, was useful in modeling discrete and continuous behavior of hybrid systems, technique described in this paper further extended the QFM process to include specific guidelines and procedures to conducting safety analysis, systematic generation of fault trees in particular.

Conventional fault trees are also constructed based on systematic analysis on possible system states. However, it is difficult to extract the possible system states at the requirements analysis phase of the software development lifecycle. It is also difficult to find the causal conditions of the state changes. Even though the state trajectory does not represent the complete set of procedures on all the possible system states, CRSA is a process to support the construction of fault trees to be more systematic by understanding the system states and the conditions of the state changes as early as possible.

We have applied the proposed technique to partially specify and validate safety requirements of Wolsong SDS2 which is an emergency shutdown system for a nuclear power plant. Because CRSA was a fault tree method based on the dynamic QFM models of a system, we could reduce the dependency on the subjective knowledge when constructing the fault trees by utilizing the causality information from the models.

I would like to say the usefulness of our approach by comparing it to the existing approach for NPP. AECL, the Canadian supplier of Wolsong 2/3/4 nuclear power plants, developed a procedure of software fault tree analysis for the program code of the Shutdown System 2 (SDS2) software. When creating these fault trees using the Wolsong FTA approach, there were following drawbacks:

No.	Drawbacks of Wolsong approach	Usefulness of CRSA
1	It is a static FTA based only on the information of the program structure. In a software FTA, it is difficult to find the causes of software hazards from the structural information of the code, such as calling sequence, program structure, and module structure, because the I/O and the state changes of software determine whether the system could be hazardous or not.	Because CRSA has not only the logical but also the dynamic relations with the time ordering between the parent and the children nodes of the fault tree, it make possible to find the causal relations between the physical hazard of the system and the hazardous behavior of the software. That is, because the QFM model has both the structural and the behavioral information of the continuous and discrete time, CRSA can analyze the hazard caused by dynamic causes, such as time ordering errors in the software requirements for HRTS.
2	It is difficult to relate the logical fault of software to the physical hazard of system because of the long conceptual distance between the logic of code and the physics of the system. It is not cost-effective because it is too late to fix the hazardous software in the coding phase of the lifecycle.	CRSA provides a mean to check the safety of software in early phase of the lifecycle. CRSA helps the testing and the detail safety analysis in later phases of the lifecycle. The results of CRSA can be used to guide software testing in implementation phase. The interfaces of the software parts of the fault trees can be used to determine the test case.
3	It depends only on the subjective knowledge from engineer's experiences when searching the cause of the upper event. It is difficult to find the possible system states and the causal conditions of the state changes.	CRSA is the model based FTA method. Because it is based on the dynamic QFM model for a hybrid system, we can reduce the dependency on the subjective knowledge by utilizing the causality information from the model.

Acknowledgements

The work described here was supported in part by the Ministry of Science and Technology under the KNICS project, and in part by AITrc of KAIST.

References

[1] Veseley WE. Fault tree handbook, Division of the System Safety Office of Nuclear Reactor Regulation.: US Nuclear Regulatory Commission; 1981.

[2] Lee JS, Cha SD. Qualitative formal method for requirements specification and validation of hybrid real-time safety systems. IEE Proc Software J 2000;14:1–11.
[3] Lee JS, Cha SD. Behavior verification of hybrid real-time requirements by qualitative formalism. In: Proceedings of the fourth RTCSA, Taipei. Taiwan: IEEE Computer Society Press; 27–29 October 1997. p. 127–34.
[4] Iwasaki Y, Vescovi M, Fikes R, Chandrasekaran BA. Causal functional representation language with behavior-based semantics. Appl Artificial Intell J 1995;9:5–31.
[5] Falkenhainer B, Farquhar A, Bobrow D, Fikes R, Forbus K, Gruber T, Iwasaki Y, Kuipers B. CML: a compositional modeling language. KSL Report No. KSL-94-16, Stanford University; 1994.
[6] Iwasaki Y, Low CM. Model generation and simulation of device behavior with continuous and discrete change. KSL Report No. KSL-91-69, Stanford University; November 1991.
[7] Leveson NG. SAFEWARE: system safety and computers. Reading, MA: Addison-Wesley; 1995.
[8] Hansen KM, Ravn AP, Stavridou V. From safety analysis to formal specification. ProCos II Technical Report, ESPRIT, Department of Computer Science, Technical University of Denmark; 1994.
[9] Liu S, McDermid JA. Model-oriented approach to safety analysis using fault trees and a support system. J Syst Software 1996;35: 151–64.
[10] Cha SS. A safety-critical software design and verification technique. PhD Dissertation, Information and Computer Science Department, UC Irvine; 1991.
[11] Fenelon P, McDermid JA. An integrated tool set for software safety analysis. J Syst Software 1993;21:279–90.
[12] Subramanian S, Vishnuvajjala RV, Mojdebakhsh R, Tsai WT, Elliott LA. Framework for designing safe software systems. COMPSAC'95, Dallas, TX, USA; August 1995. p. 409–14.
[13] Friedman MA, Voas JM. Software assessment: reliability, safety, testability. Wiley: New York; 1995.
[14] Clarke SJ, McDermid JA. Software fault trees and weakest preconditions: a comparison and analysis. IEE Proc Software J 1993;225–36.
[15] Leveson NG, Stolzy JL. Safety analysis of Ada programs using fault trees. IEEE Trans Reliab 1983;5:479–84.
[16] Leveson NG, Stolzy JL. Safety analysis using Petri nets. IEEE Trans Software Engng 1987;13:386–97.
[17] Cha SS, Leveson NG, Shimeall TJ. Safety verification in murphy using fault tree analysis. In: Proceeding of the ICSE-10. Singapore: IEEE Computer Society Press; 1988. p. 377–86.
[18] Leveson NG, Harvey PR. Analyzing software safety. IEEE Trans Software Engng 1983;5:569–79.
[19] Gorski J, Wardzinski A. Formalizing fault trees. In: Proceedings of the safety-critical systems symposium, UK; 7–9 February 1995. p. 311–27.
[20] SSM Systems Procedures for electrical power subsystem (SE-23, vol. III), Lockheed Missiles and Space Company document No. D889543A; 1985.
[21] Ostroff JS. Formal methods for the specification and design of real-time safety critical systems. J Syst Software 1992;33–60.