템플릿에 기반한 NuSCR 정형 명세의 소프트웨어 고장 수목 생성 방법

(A Synthesis Method of Software Fault Tree from NuSCR Formal Specification using Templates)

김 태 호 † 유 준 범 †† 차 성 덕 †††
(Taeho Kim) (Junbeom Yoo) (Sungdeok Cha)

요 약 본 논문은 NuSCR 정형 명세 언어로 작성된 소프트웨어 요구 명세로부터 소프트웨어 고장 수목을 생성하는 방법에 대하여 제안하였다. 본 연구에서 제안하는 소프트웨어 고장 수목은 소프트웨어의 구조와 동작에 대한 요구 사항을 반영하는 통합된 형태의 고장 수목으로, 안전성에 대한 복합적인 분석이 가능하다. 이러한 소프트웨어 고장 수목을 생성하기 위하여 NuSCR 정형 명세언어의 구성 요소 각각에 대한 템플릿을 정의하고, 이들 템플릿을 사용하여 소프트웨어 고장 수목을 생성하는 방법을 제안하였다. 그리고, 제안된 방법의 유용성을 평가하기 위해 현재 국내 원전계측제어시스템 개발사업단에서 개발 중인 차세대 원자력 시스템 APR1400에 사용될 원자로 보호 시스템의 핵심 트립 논리에 대하여 고장 수목을 생성하고 분석 하였다.

키워드 : 소프트웨어 고장 수목, 고장 수목, 정형 명세, 템플릿, 안전성 분석

Abstract In this paper, we propose a synthesis method of software fault tree from software requirements specification written in NuSCR formal specification language. The software fault tree, proposed in this paper, reflects requirements on both structure and behavior and it is an integrated form. The software fault tree can be used for analyzing safety in the view of structure and behavior. We propose templates for each components in NuSCR specification language and a synthesis method of software fault tree using the templates. The research was applied into the main trip logic of the reactor protection system of ARP1400, the Korean next generation nuclear reactor system, developed by KNICS. And we evaluate feasibility of our approach through this case study.

Key words : Software Fault Tree, Fault Tree, Formal Specification, Template, Safety Analysis

1. 서 론

원자력 제어 시스템이나 항공 제어, 의료 장비 시스템과 같이 안전성이 중요한 시스템의 안전성 확보는 매우 중요하다. 따라서, 규제 기관은 시스템의 운영 이전에 안전성 분석을 요구하고, 개발 회사는 안전성을 확보하기 위해 안전성 분석을 수행한다. 안전성 분석 방법에는 여러가지가 있지만, 역사가 오래되고 산업계에서 가장 많이 사용되는 방법 중에 하나로 고장 수목(fault tree) 분석 방법이 있다. 고장 수목 분석은 원래 1962년에 Bell 연구소에서 미 공군의 Minuteman 시스템의 분석을 위해 개발되었고, 후에 Boeing에서 채택되어 광범위하게 사용되기 시작하였다[1].

고장 수목은 시스템에 발생 될 수 있는 사고(failure)를 정의하고, 그 사고의 원인을 찾아가면서 나무의 뿌리 형태로 표현하는 귀납적인 분석 방법이다. 이 방법은 직관적이고 분석을 위해 특별한 도구가 필요하지 않은 장점을 가진다. 그러나, 몇 가지 한계점이 있는데, 첫째는 고장 수목을 구성하는 체계적인 방법이 정의되어있지 않기 때문에 고장 수목의 품질이 분석자의 지식과 경험에의해 결정된다는 것이고, 둘째는 소프트웨어가 아니라 하드웨어에 주로 적용되었다는 것이다.

본 연구에서는 정형 명세 언어인 NuSCR[2,3]로 기술된 소프트웨어 요구 명세의 소프트웨어 고장 수목 생성

† 정 회 원 : 한국전자통신연구원 연구원
taehokim@etri.re.kr
†† 학생회원 : 한국과학기술원 전자전산학과
jbyoo@salmosa.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전자전산학과 교수
cha@salmosa.kaist.ac.kr
논문접수 : 2005년 1월 26일
심사완료 : 2005년 10월 11일

기법을 제안한다. 기존에도 소프트웨어의 요구 명세 또는 모델에 대해 소프트웨어 고장 수목(이후에는 소프트웨어에 대한 고장 수목을 의미하는 '소프트웨어 고장 수목'은 오해가 생기지 않는 한도에서 '고장 수목'이라는 용어로 대신 사용한다.)을 생성하는 연구들이 있었으나, 기존 연구들은 주로 소프트웨어 또는 시스템의 구조로부터 생성되는 고장 수목에 대한 연구였다[4-14]. 예를 들어, 3개의 시스템 중 2개의 시스템의 출력을 선택하는 고장 감내 시스템(fault tolerance system)을 분석하기 위해 생성된 고장 수목은, 각각의 시스템의 동작에 의한 고장(fault)를 분석하는 것이 아니고, 각 시스템에서의 고장이 전체 시스템의 안전에 어떤 영향을 미치는지 분석하는 것이다. 따라서 고장 수목을 통해 서브시스템의 추상화된 분석은 가능하지만, 고장 수목에 나타나 있는 정보가 시스템의 전체적인 안전성을 분석하기에는 부족하다. 본 연구에서는 구조와 시스템의 동작을 함께 분석할 수 있는 고장 수목을 생성하는 방법을 제안하였다.

제안하는 방법의 유용성을 확인하기 위하여 원전계측제어시스템 개발사업단에서 개발중인 차세대 원전 시스템인 APR1400의 원자로 보호 시스템(RPS Reactor Protection System)의 일부 핵심 소프트웨어[15]에 본 논문에서 제안하는 소프트웨어 고장 수목 분석을 적용하였다. RPS의 요구 명세는 정형 명세 언어인 NuSCR[2,3]로 작성되어 있다. NuSCR은 Parnas의 4변수 모델[16], 주기적인 수행, 자료 흐름도(data-flow diagram)[17], 유한 상태 기계(FSM; finite state machine), 시간 천이 시스템(TTS; timed transition system)[18]에 기반하고 있으며, 원자력 발전소의 제어 소프트웨어 특성을 쉽게 명세할 수 있도록 수정, 보완된 정형 명세 언어이다. 본 연구 결과는 원자력 분야에 적용되었지만, NuSCR이 가지고 있는 제어 소프트웨어의 기술 요소 때문에 원자력 분야 외에 다른 분야의 제어 소프트웨어에도 적용될 수 있다.

본 논문의 구성은 다음과 같다. 2장은 소프트웨어 고장 수목 분석에 관련된 기존의 연구에 대해 소개한다. 3장에서는 NuSCR의 각 구성 요소에 대한 템플릿을 정의하고, 그 템플릿을 이용해서 고장 수목을 생성하는 방법을 소개한다. 4장에서는 제안된 방법을 실제 산업계 소프트웨어인 APR1400에 적용한 예를 보이며, 5장에서는 결론을 기술하였다.

2. 관련 연구

2.1 소프트웨어 고장 수목

하드웨어를 대상으로 하는 고장 수목은 오랜 기간 연구되고 사용되어서 여러 연구 및 적용 결과가 존재하지만, 소프트웨어를 대상으로 하는 고장 수목에 대한 연구는 많이 이루어지지 않은 실정이다. 소프트웨어를 대상으로하는 것은 프로그램 소스 코드 또는 소프트웨어의 명세를 대상으로 하는 것이다.

프로그램 소스 코드를 대상으로 하는 연구는 Ada83과 Ada95로 작성된 프로그램으로부터 고장 수목을 구성하는 연구가 있었다[4-6]. 이 연구들은 소프트웨어의 고장에 해당하는 프로그램 코드를 지정하고, 코드에 해당되는 템플릿을 사용하여 고장 수목을 전개해 나감으로써 고장 수목을 구성하는 방법이다. 이때 프로그램 코드의 구성 요소각각에 대해서는 사전에 템플릿이 정의되어 있다. 프로그램의 구성 요소에 해당되는 템플릿은 프로그램 슬라이싱(slicing) 기법이나 최약 사전 조건(weakest precondition) 분석과 유사하다.

소프트웨어 명세를 대상으로 하여 고장 수목을 구성하는 연구는 프로그램 코드를 대상으로 하는 경우보다는 많이 진행되었다. 소프트웨어 명세가 아닌 시스템의 명세를 대상으로 하는 경우도 있는데, 소프트웨어의 명세의 고장 수목을 생성하는 것도 관련되어 있다.

시스템 공학 분야의 Apostolakis 등은 Dynamic Flowgraph Methodology(DFM)라는 명세 방법을 제안하여 소프트웨어로 제어되는 내장형 시스템에 대한 통합된 모델링과 분석을 가능하도록 하였다[7]. 이 방법은 Titan II 발사체 디지털 비행 제어 시스템에 적용되었으며[8], 신뢰성(dependability) 분석 방법[9]과 위험 분석(hazard analysis)[10]을 지원하도록 확장되었다. Apostolakis의 연구 방향은 본 연구와 유사하지만, DFM 자체가 소프트웨어의요구 사항을 완전하게 기술하기 위한 것이 아니고 산업 프로세스를 기술하기 위한 명세 방법이기 때문에 일반적인 소프트웨어의 요구 명세를 기술하는 데는 부족함이 있어서 결과적으로 소프트웨어 안전성 분석에 적절하지 않다.

Papadopoulos는 자료 흐름도와 유사한 형식인 아키텍처도(architectural diagram)로부터 고장 수목을 생성하는 HiP-HOPS(Hierarchically Performed Hazard Origin and Propagation Studies)을 제안하였다[11]. 이 방법은 요구 명세로부터 안전성 분석을 수행한다는 점에서는본 연구와 동일하지만, 컴포넌트의 내부 동작은 반영하지 않고 컴포넌트를 블랙 박스처럼 처리하여 고장 수목의 구조에 대한 내용만을 분석할 수 있다는 한계를 지닌다. Papadopoulos는 이 연구를 확장하여 Matlab-Simulink 모델에 대해서도 고장 수목을 생성하는 방법을 제안하였지만[12], 컴포넌트 이하의 수준을 다루지 못하는 한계는 해결되지 않았다.

Sullivan은 아키텍처 기술언어인 RIDL(Reliability Imbedded Design Language)에서 고장 수목을 생성하는 방법을 제안하고, 분석 도구 Galileo를 개발하여 적

용하였다[13,14]. RIDL은 중복된 모듈과 컴포넌트를 표시할 수 있는 방법을 지원하는 명세 언어이다. 하지만, Papadopoulos의 연구들과 마찬가지로 모듈과 컴포넌트 이하의 정보는 반영할 수 없는 한계가 있다. 본 연구에서는 요구 명세로부터 고장 수목을 생성하는 관점에는 다른 연구들과 유사하지만, 모듈 수준 이하에 나타나는 세부적인 정보를 반영하여 고장 수목을 생성한다는 관점에서는 다른방법들과 차이가 있다.

2.2 NuSCR 정형 요구 명세기법

NuSCR[2,3]은 AECL의 SCR 방법론[16,19]을 기반으로 원자력 발전소 제어 소프트웨어의 명세에 유용하도록 정의된 정형 명세 기법이다.

소프트웨어의 상위 관점은 FOD(Function Overview Diagrams)로 기술된다. FOD는 자료 흐름도와 유사한 표기법이다. 전체 시스템은 가장 상위 수준에서는 입력 변수 노드들(input variable nodes)을 통해 외부에서 입력을 읽어 시스템의 내부로 전달하고, 계산된 결과는 출력 변수 노드들(output variable nodes)을 통해 외부로 전달된다. 이러한 노드는 다시 계층적으로 구성될 수 있으며, 노드에는 입력과 출력이 표시된다. 가장 하위 수준의 노드들에는 구조화된 결정표(structured decision table; SDT)로 표현되는 함수 변수 노드(function variable node), 유한 상태 기계(finite state machine FSM)로 표현되는 히스토리 변수 노드(history variable node), 시간 천이 시스템(timed transition system; TTS)으로 표현되는 시간-히스토리 변수 노드(timed-history variable node) 들이 있다.

그림 1은 본 연구에서 사례로 사용한 APR1400에서 RPS의 일부분인 수동 리셋 가변 설정치 상승 트립 논리(manual reset variable set-point rising trip logic)를 NuSCR로 기술한 것이다. 그림 1(a)는 이 트립 논리에 대한 FOD이다. 그림 1(a)의 왼쪽에 나열된 변수들은 이 트립 논리 시스템으로의 입력을 나타내며, 입력 변수들에는 공정 변수를 표시한 f_X, 리셋 신호를 의미하는 th_Reset_Ini, 모듈 오류와 채널 오류를 의미하는 f_M_Err와 f_C_Err이 있다. 그림 1(a)의 오른쪽에 나열된 변수들은 트립 논리 시스템의 출력을 의미한다. 출력 변수에는 정지 신호를 의미하는 th_Trip과 예비 정지 신호인 th_Pretrip이 있다.

그림 1(b)는 그림 1(a)에 표기된 f_X_Valid 함수 변수 노드의 동작을 구조화된 결정표로 기술한 것이다. 즉, k_X_MIN <= f_X AND f_X <= k_X_MAX(열 2)가 만족되면 이 함수 변수 노드는 f_X_Valid로 0을 출력한다. 그외에 k_X_MIN > f_X(열 3)이거나 f_X > k_X_MAX(열 4)인 경우에는 f_X_Valid로 1을 출력한다.

그림 1(c)는 유한 상태 기계로 기술된 히스토리 변수 노드 h_Spt이다. 이 유한 상태 기계의 초기 상태는 MAX(그림 1(c)의 좌측 하단)이고, 변수 h_Spt의 초기 값은 k_Spt_Ini이다. 이 상태에서 조건 A와 D 즉, f_X < prev(f_X)와 f_X - k_Int < k_Spt_Ini가 만족되면 상태가 Falling(그림 1(c)의 좌측 상단)으로 변경된다. prev(X) 표현은 X의 저장된 이전 값을 의미한다. 이 조건이 만족될 때의 출력은 지정되어 있지 않으므로 계속 h_Spt := k_Spt_Ini를 유지한다. 이 상태에서 A and B and C (즉, f_X < prev(f_X) and th_Reset_Ini=1 and f_X - k_Intl > k_LLimit)를 만족하면 상태는 Falling (그림 1(c)의 좌측 상단)으로 유지되며 이때의 출력값 h_Spt는 f_X - k_Int로 갱신된다. 이 유한 상태 기계의 다른 동작도 이와 같은 방식으로 처리된다.

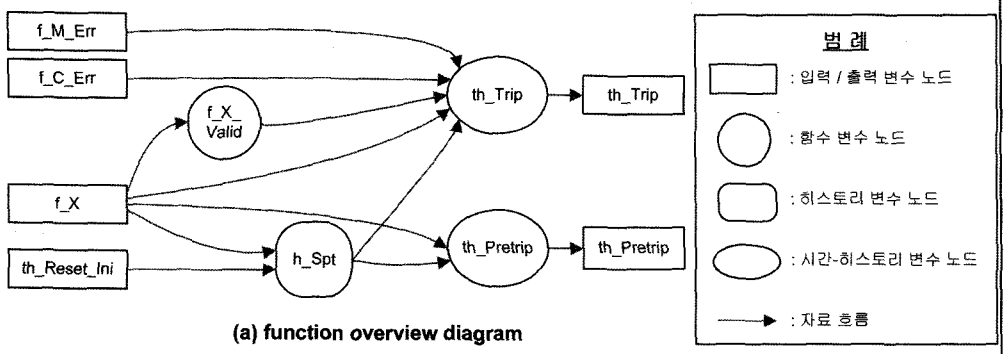
그림 1(d)는 시간 천이 시스템으로 기술된 시간-히스토리 변수 노드이다. 시간 천이 시스템은 시간 제약 조건을 표기할 수 있다는 것을 제외하고는 유한 상태 기계와 동일하다. 따라서, th_Trip의 초기 상태는 Normal (그림 1(d)의 좌측 상단) 이며 출력값 th_Trip은 1이다. 이 상태에서 f_X <= h_Spt - k_Band가 만족되면 상태는 Waiting(그림 1(d)의 우측 상단)으로 변경되고 출력값 th_Trip은 1이다. 여기에서 f_X_Valid = 1 or f_M_Err=1 or f_C_Err=1이면 즉시 Trip(그림 1(d)의 하단) 상태로 변경되고 출력값 th_Trip이 0으로 설정된다. 또한 [k_Trip_Delay, k_Trip_Delay] f_X <= h_Spt - k_Band가 만족될 때, 즉 f_X <= h_Spt - k_Band 조건이 최소 k_Trip_Delay 시간, 최대 k_Trip_Delay 시간 동안 (즉, 이 경우에는 k_Trip_Delay 시간 동안) 유지 되는 경우에도 th_Trip이 0으로 설정된다. 이와 같이 시간 제약 조건을 기술할 수 있는 것이 시간 천이 시스템의 특징이다.

3. 고장 수목 생성

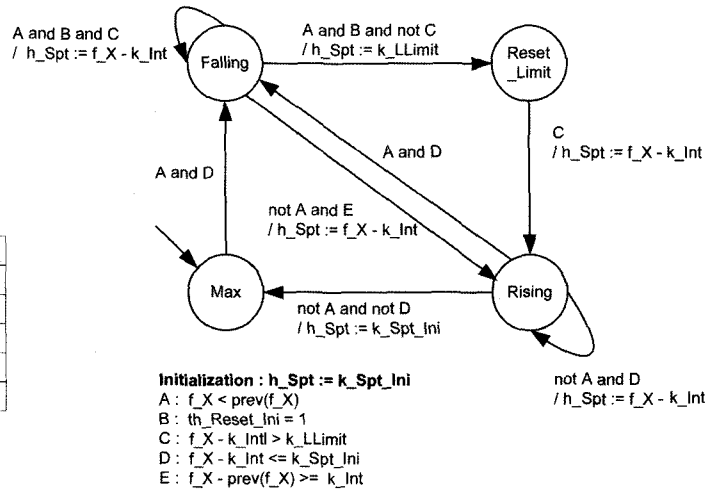
본 장에서는 NuSCR로 작성된 소프트웨어 요구 명세로부터 고장 수목을 생성하는 방법을 소개한다. 3.1절에서는 NuSCR 요구사항 명세로부터 고장 수목을 생성하는 절차를 설명한다. 그리고, 3.2절에서는 NuSCR 요구사항 명세의 각 요소들에 해당하는 고장 수목 템플릿에 대하여 설명한다.

3.1 고장 수목 생성 절차

고장 수목은 발생할 수 있는 사고(failure)를 정의하고 그 원인을 찾아가는 분석 방법이다. 소프트웨어 명세에서 어떤 사고의 원인이란 특정 출력을 생성하게되는 선행 조건이다. 선행 조건은 두 가지로 생각할 수 있는데, 한가지는 특정 출력을 만들기 위한 자료의 계산 흐름에 의한 데이타의 선행 조건이고, 다른 한 가지는 시간의



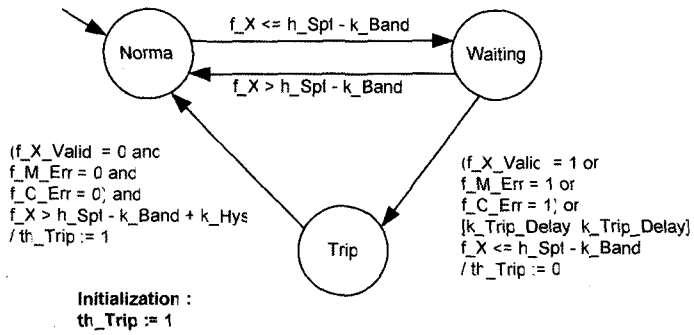
(a) function overview diagram



Conditions			
k_X_MIN <= f_X	T	F	-
f_X <= k_X_MAX	T	-	F
Actions			
f_X_Valid := 0	X		
f_X_Valid := 1		X	X

(b) 구조화된 결정표로 표시된 함수 변수 노드 **f_X_Valid**

(c) 유한 상태 기계로 표시된 히스토리 변수 노드 **h_Spt**



(d) 시간 천이 시스템으로 표시된 시간-히스토리 변수 노드 **th_Trip**

그림 1 NuSCR 정형 명세 언어의 주요 구성요소

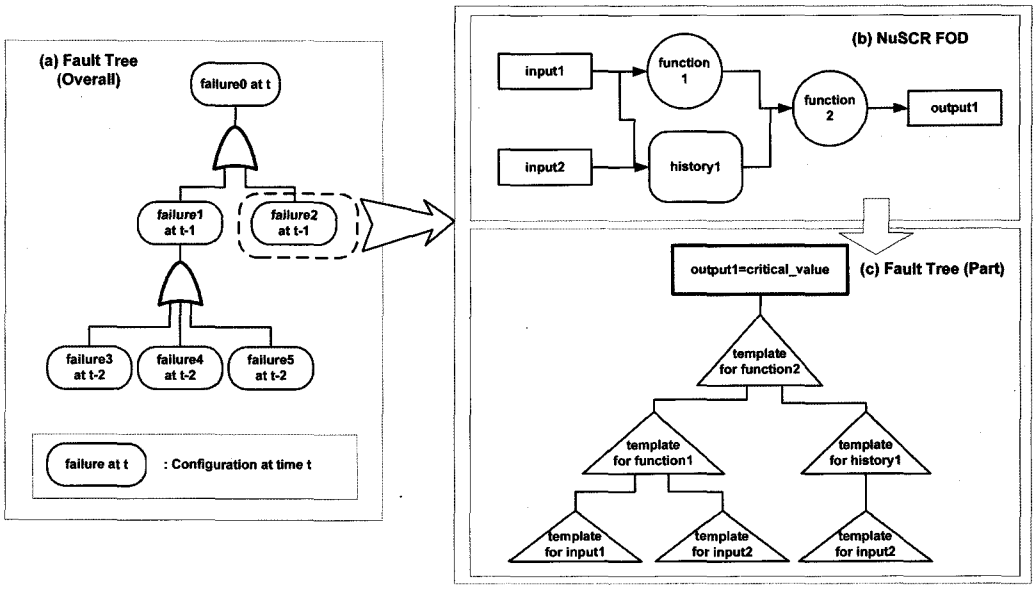


그림 2 고장 수목 개념도

선행 조건이다. 이 두 종류의 선행 조건을 모두 표현할 수 있는 고장 수목이 생성되어야 한다. 두 종류의 선행 조건을 모두 표현하기 위해서, 본 연구에서는 고장 수목을 두개의 계층으로 나누었다. 한가지 계층은 시간의 천이에 의한 시스템의 형상[1](configuration)의 변화를 표현하고, 다른 한가지 계층은 데이타의 흐름에 의한 데이타 값의 결정을 표현한다. 이와 같은 개념은 그림 2에 나타나 있다.

그림 2(a)는 시스템의 형상 간의 천이 관계를 나타낸다. 즉, 사고(failure)가 발생했을 때의 형상과 이전 주기의 형상 간의 천이 관계를 나타낸다. NuSCR 요구 명세로 기술된 시스템은 센서에서 입력 값을 읽어 처리한 후 외부로 출력을 내는 동작을 주기적으로 반복한다. 이러한 동작 패턴은 임베디드 제어 시스템에서 흔히 발견된다. 이해를 돕기 위해 그림 2에서 주기는 1로 가정하였다. 그림 2(a)의 의미를 살펴보면, t 시간에서 형상이 failure0이기 위해서는 t-1 시간에서의 형상은 failure1이거나 failure2라는 것을 보이고 있다. 그리고, t-1 시간에서 형상이 failure1이기 위해서는 t-2 시간에서의 형상은 failure3 또는 failure4 또는 failure5임을 나타내고 있다.

그림 2(c)는 그림 2(a)의 failure2에 해당하는 형상을 얻기위한 과정이다. 예를 들어 중요한 사건이 'output1 =critical_value'라면, 이 표현이 고장 수목의 맨 위 노드가 된다. 고장 수목의 맨 위 노드에 나타난 변수인 *output1*은 함수 변수 노드 *function2*의 출력이다. 따라서, 함수 변수 노드에 해당하는 고장 수목 템플릿(3.2절에서 정의)으로 치환하여 확장한다. 이렇게 확장된 고장 수목의 하위 노드에는 function2 노드의 입력인 function1과 history1이 나타난다(그림 2(b) 참조). 따라서, 함수 변수 노드 function1과 히스토리 변수 노드 history1에 해당되는 템플릿을 사용하여 확장하고, input1, input2에 해당하는 템플릿을 확장하면 최종적으로 그림 2(c)가 얻어진다.

지금까지 설명한 고장 수목 생성 알고리즘을 정리하면 다음과 같다.

고장 수목의 정의 : 고장 수목은 이벤트 노드 v의 집합인 V(v ∈ V)와 게이트 노드 g의 집합인 G(g ∈ G)로 분류(partition)된 노드 N이 있고(N = V ∪ G, V ∩ G = Ø), V 집합과 G 집합의 원소간의 연결선 e의 집합 E(e ∈ E)로 구성된 방향성 이분 그래프(directed bipartite graph) F(즉, F = (N,E))이다. 이때 F의 시작 노드 s와 끝 노드 f는 V의 원소이며(s ∈ V, f ∈ V), s의 개수는 하나이다.

고장 수목 생성 절차 개요 : NuSCR로 기술된 명세에서 고장 수목을 생성하는 절차는 다음과 같다.

1. 사고(failure)를 정의한다. 사고는 시스템에서 발생하지 않기를 바라는 사건이다. 이 사고가 시작 노드 s이다. 전개되지 않은 노드 집합 U에 s를 할당한다(U = {s}).

1) 시스템의 상태와 변수의 상태라는 의미를 구분하기 위해서 시스템의 상태를 형상으로 기술하였다.

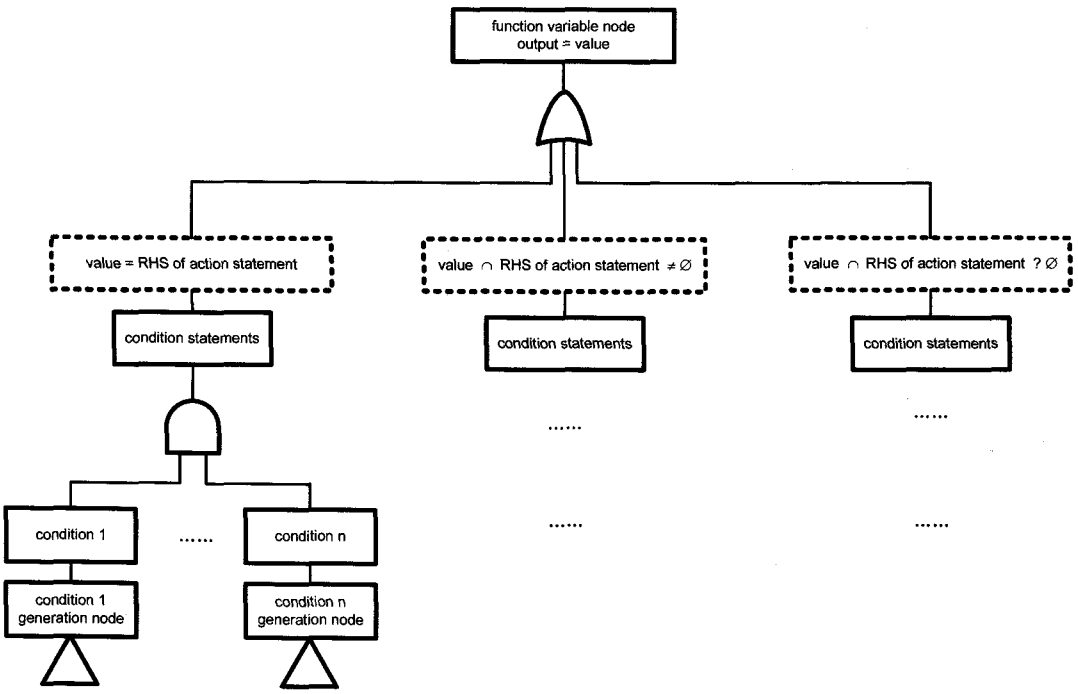


그림 3 함수 변수 노드의 고장 수목 템플릿

2. U의 원소 중 한 개의 노드 u를 선택하여 해당하는 템플릿으로 노드 u에 연결한다. 노드에 해당하는 템플릿은 노드에 나타나는 변수를 출력으로 하는 NuSCR의 변수 노드를 찾아 그 변수 노드의 종류에 해당하는 템플릿을 선택한다. 전개되지 않은 노드 집합 U에서 u를 제거하고, 새로 연결된 노드의 단말 노드 집합 T를 U에 포함한다(U = (U - {u}) ∪ T).
3. 분석의 기준을 만족시킬 때까지 2의 방법으로 고장 수목을 생성한다. 분석의 기준은 시스템의 특성이나 정책 등에 의해 결정된다. 예를 들어, 선형 제어 시스템인 경우 하나 혹은 두개의 주기를 분석하는 것으로 충분하다는 실험적인 기준이 존재하므로 한 주기를 선택할 수 있다.

3.2 NuSCR 요구사항 명세를 위한 고장 수목 템플릿

3.1절의 고장 수목 생성 절차의 2번째 단계에서 고장 수목의 단말 노드의 변수 값을 생성하는 NuSCR의 변수 노드를 찾아 그 변수 노드의 종류에 따른 템플릿을 선택한다고 하였다. 변수 노드의 종류에는 함수 변수 노드, 히스토리 변수 노드, 시간-히스토리 변수 노드, 입력 변수 노드, 출력 변수 노드가 있는데, 이중 출력 변수 노드는 단순하게 외부로 값을 전달하는 역할을 하기 때문에 고장 수목에 나타나지 않는다. 따라서, 함수 변수 노드, 히스토리 변수 노드, 시간-히스토리 변수 노드, 입력 변수 노드에 대한 템플릿을 정의한다.

참고로 템플릿에서 사용되는 기호 중, *"variable = value at t"*는 특정 시각 *t*에서 *variable*의 값이 *value* 임을 기술한다. 만약 *p*가 시스템의 주기(NuSCR로 기술된 시스템은 주기적으로 수행하는 시스템을 가정한다)라고 가정할 때, *"t - p"* 시각은 시각 *t*로부터 한 주기 전의 시각이다. 마찬가지로, *"t - 3p"*는 시각 *t*로부터 3 시스템 주기 이전 시각이다.

(1) 함수 변수 노드

가. 템플릿 정의

함수 변수 노드의 동작은 AND-OR 테이블의 일종인 구조화된 결정 테이블(SDT; Structured Decision Table)로 정의된다. SDT는 조건 부분과 동작 부분으로 구성된다. 함수 변수 노드의 출력을 정의하고 있는 동작 부분에 해당하는 조건들을 나열함으로써 데이타 선행 조건을 찾는 것이 템플릿의 내용이다.

그림 3은 고장 수목의 노드가 *"output = value"*일 때의 템플릿이다. 각각의 가지는 출력을 생성하는 조건을 표시한 것이다. 두번째 수준의 왼쪽에서 첫번째 노드(value = RHS[2] of action statement)는 시작 노드(top node)에 기술된 변수의 값과 동작 문의 우변의 값이 같은 경우이고, 왼쪽에서 두번째 노드(value ∩ RHS of

2) RHS = Right Hand Side

statement = Ø)는 값이 중첩되는 경우이다. 이 두 경우는, 출력이 생성되는 경우를 알 수 있는 경우이다. 마지막으로 왼쪽에서 세번째 노드(value ∩ RHS of statement ? Ø)는 출력을 생성할 수 있는지 없는지 알 수 없는 경우이다. 이것은 사고를 분석하기 위해서는 모든 경우에 대해 분석해야하기 때문에 포함하여야 한다. 노드를 점선으로 표시한 것은 조건이 만족될때만 선택적으로 전개하는 것을 표시한 것이다. 두번째 세번째 노드의 하위 노드인 condition statements는 첫번째 노드의 하위 노드들과 동일한 방법으로 전개되기 때문에 그림을 단순화 하기 위하여 생략하였다.

출력문을 생성하는 조건문은 두번째 수준 이하의 노드에 기술된다. 만약 조건문이 논리 연산자로 연결된 여러 개의 조건들로 구성된다면, 조건들은 AND와 OR 게이트 같은 논리 게이트로 연결된다. 이 과정을 통해 단말 노드의 조건은 변수의 값으로 표시된다. 단말 노드의 변수들이나 외부 입력에 해당하는 템플릿을 사용하여 계속하여 고장 수목을 전개할 수 있다. SDT에서 모든 노드가 같은 시각의 값들로 시간이 한정된다.

나. 예제

그림 1에 기술된 시스템에서 고장 수목의 노드가 *"f_X_Valid = 1"*이라고 할 때, *f_X_Valid*를 생성하는 노드는 함수 변수 노드이다. 그 함수 변수 노드가 그림 1(b)의 SDT로 기술될 때 전개되는 고장 수목은 그림 4와 같다. *"f_X_Valid = 1"*이기 위해서는 그림 1(b)의 SDT에서 3 번째와 4 번째 열에 해당하는 조건이 만족되어야하고, 따라서 이에 해당하는 조건 문인 *"k_X_Min > f_X"*와 *"f_X > k_X_Max "*를 전개한다.

(2) 히스토리 변수 노드

가. 템플릿 정의

히스토리 변수 노드는 유한 상태 기계(FSM; Finite State Machine)를 이용해서 정의되는데, FSM으로는 특정 출력이 상태들 중 어떤 상태에서 나오는지 판단이 어렵다. 이는 출력이 상태에 의해서만 결정되는 것이 아니라, 현재 상태에 도달하는 경로에도 의존하기 때문이다. 분석을 쉽게하기 위해 FSM을 형상{state_name, {variable = value}}으로 기술하도록 확장된 FSM으로 변환하여, 특정 값을 가지는 상태를 판별할 수 있다.

그림 5는 히스토리 변수 노드를 정의한 FSM 이다. 이 FSM은 S1~S4의 4개의 상태, C1~C6의 조건과 k1~k4인 4개의 출력을 가지는 출력 변수 out으로 구성되어 있다. 고장 수목에서 전개할 노드에 기술된 조건이 *"out = k4"*이면, FSM에서는 *"out := k4"*이다.

- 현재 상태는 S4이고 이전 상태는 S2이다. S2에서 S4로의 천이 조건 C4가 만족 되어서 천이가 수행되었다. S2→S4로의 천이에 기술된 수행문 out:=k4가 수행된다.
- 현재 상태는 S4이고 이전 상태도 S4이다. S4에서 이동하는 천이 조건 C5가 만족되지 않아서 천이가 수행되지 않는다. 따라서 S4에서의 값 out=k4가 유지된다.
- 현재 상태는 S3이고 이전 상태는 S4이다. C5가 만족되어서 S4에서 S3로의 천이가 수행되었고, 이때 출력값 out은 변경되지 않으므로 out=k4가 유지된다.
- 현재 상태는 S3이고, 이전 상태는 S3이다. 이전에, S3로 들어올 때 S4로부터 천이된 경우이다. S2→S3를 통해서 온 것이 아니기 때문에 S2→S4의 동작 문에 의하여 out=k4로 변경되고 계속해서 out=k4로 유지되고 있는 상태이다.
- 현재 상태는 S1이고, 이전 상태는 S3이면서 out=k4인 경우이다. 예를 들어, S2→S4→S3→S1인 경우, out은 k4로 남아 있다. 중간에, S4와 S3에서 머무르고 있는 것은 out 값을 변경시키지 않는다.

이와 같이 *"out = k4"*인 경우가 S4, S3, S1 일 수 있다는 정보는 그림 5의 FSM만으로는 판별할 수없다. 따라서 논문에서 제시하는 방법은 FSM에서 고장 수목을 바로 생성하는 것이 아니라, FSM으로부터 형상을 나타내는 확장된 FSM을 생성한 후에 고장 수목을 생성한다. 그림 5의 FSM을 변환한 확장된 FSM은 그림 6

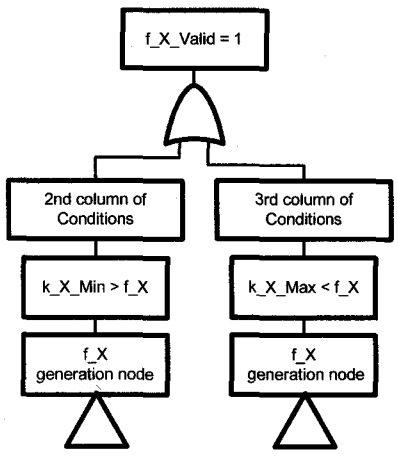


그림 4 f_X_Valid =1의 고장 수목

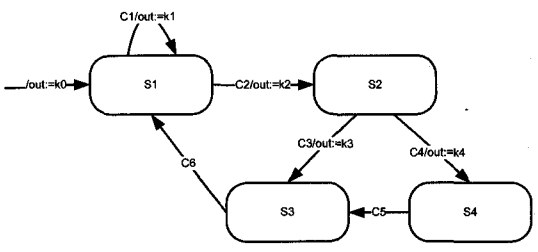


그림 5 히스토리 변수 노드에 대한 FSM

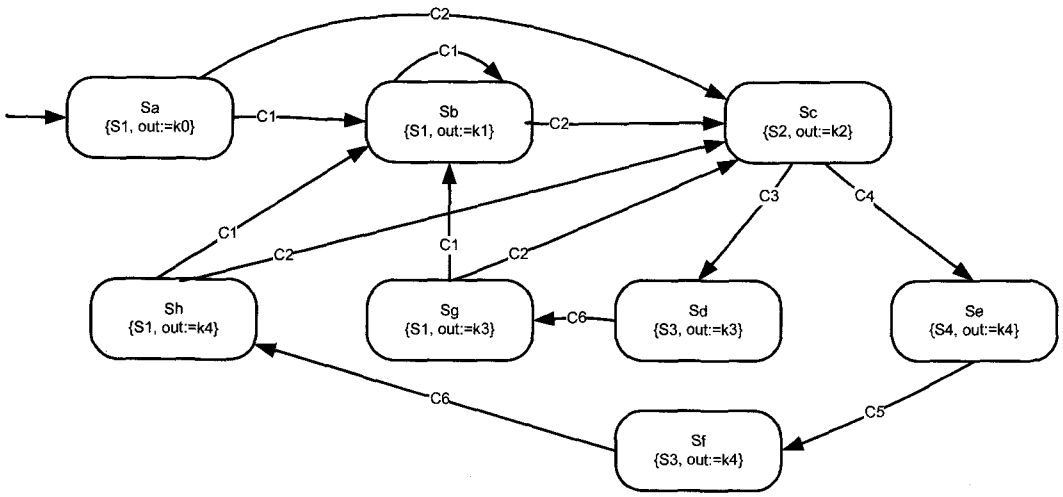


그림 6 그림 5의 FSM으로부터 확장된 FSM

이다.

FSM을 변환하는 방법은 형상 간의 천이 그래프로 변경하는 것이다. 확장된 FSM의 상태에는 출력 값이 상태에 명시적으로 표시되어 있기 때문에, 템플릿을 출력 값의 범위에 의해 정확하게 적용할 수 있다. 예를 들어, 그림 6의 out = k4는 Se, Sf, Sh에서 만족한다는 것을 알 수 있고, 이것은 그림 5에서 각각 S4, S3, S1에 해당한다.

히스토리 변수 노드의 고장 수목 템플릿은 그림 7과 8이다. 그림 7의 2번째 수준의 4개의 노드들 중 왼쪽부터 변수의 값과 노드의 값이 같은 경우, 공통된 영역이 있는 경우, 변수들 사이의 관계를 알 수 없는 경우, 상태가 같은 경우이다. 이와 같이 관련된 상태를 찾은 이후에는 그림 8의 템플릿을 연결한다.

그림 8은 확장된 FSM의 시간 t에서의 상태가 state 기 위한 두가지 경우가 나타낸다. 첫번째 경우는 그림 8의 2번째 수준의 왼쪽 노드(enter the state via state transition)로 상태 state로 들어오는 천이 tr이 수행된 경우이다. 즉, 이전 상태는 천이 tr의 원천 상태 이고 tr의 조건이 만족된 경우이다. 두번째 경우는 상태 state에서 나가는 천이 tr이 수행되지 않고 머무르고 있는 경우(remain at the state because of disabled outgoing transition)이다. 즉, 이전 상태가 현재 상태(tr의 원천 상태)와 같고, tr의 조건이 만족되지 않은 경우이다. 이러한 천이들이 여러 개 존재할 수 있기 때문에 각각의 경우들이 OR 게이트로 연결되어 있고, 이전 주기의 상태를 'Source(tr) at t-p' 로 기술한다.

나. 예제

그림 6은 그림 5의 FSM으로부터 확장된 FSM이다. 이 히스토리 변수 노드에서 생성된 고장 수목은 그림 9,

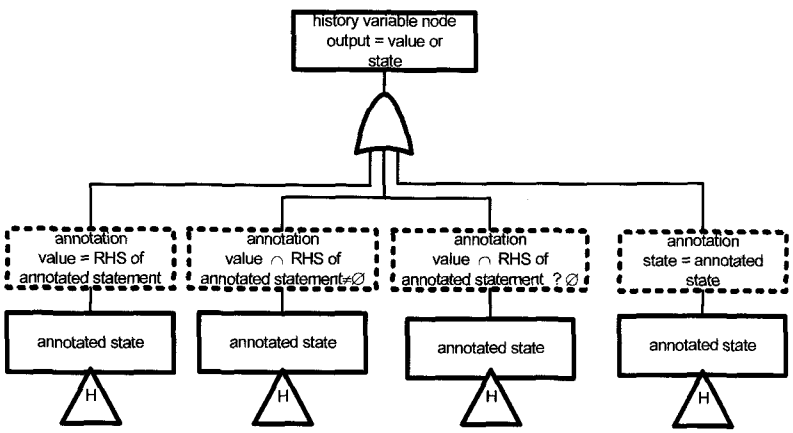


그림 7 히스토리 변수 노드의 고장 수목 템플릿 (1)

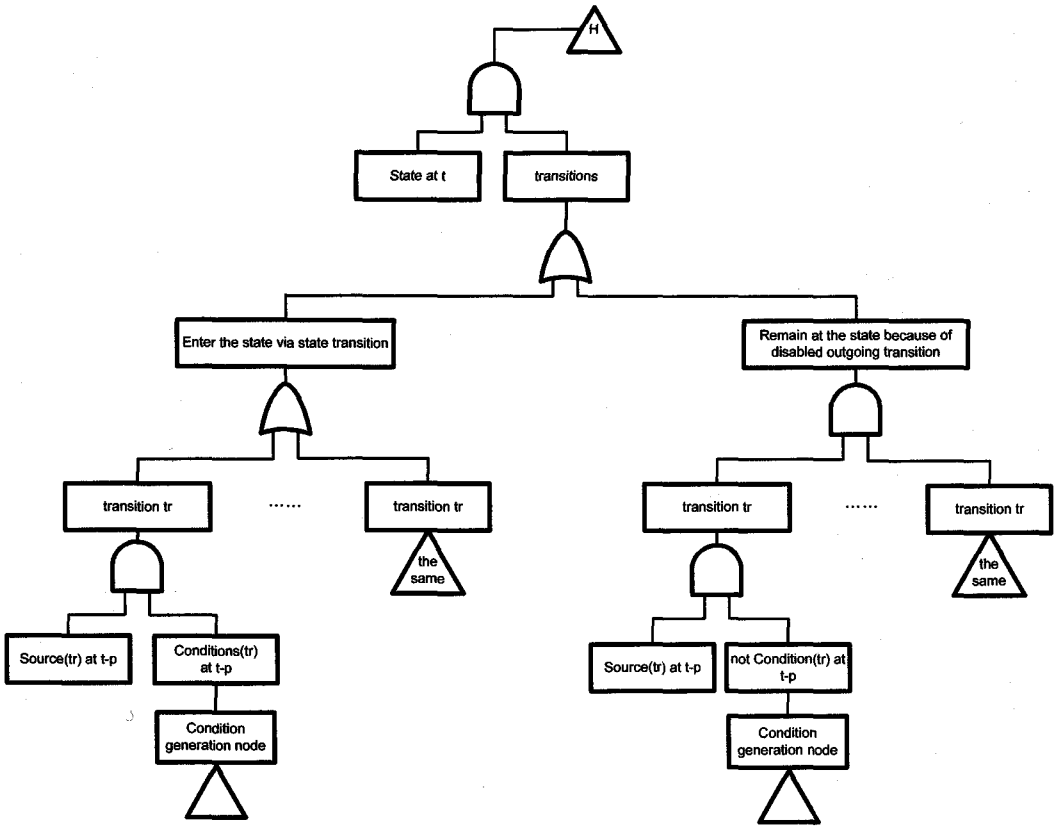


그림 8 히스토리 변수 노드의 고장 수목 템플릿 (2)

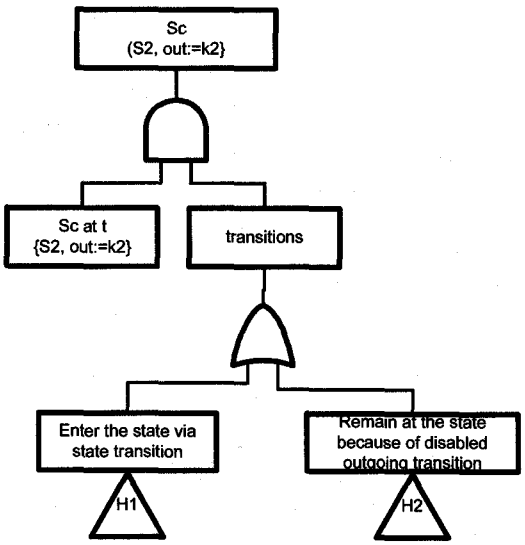


그림 9 히스토리 변수 노드의 고장 수목 예제(1)

10, 11이다. 고장 수목의 최상위 노드가 *"out = k2"*일 때, 이에 해당하는 히드토리 변수 노드의 상태는 Sc이다. 현재 상태가 Sc이기 위해서는 시간 t에서 Sc로 들어왔거나, 시간 t 이전에 Sc로 들어와서 머무르고 있는 두 경우가 있다.

그림 10의 H1에 기술된 것은 Sc로 들어온 경우를 나타낸다. 각각의 가지는 왼쪽에서부터 Sa, Sb, Sg, Sh에서 Sc로 움직이는 천이를 의미한다. 만약 시간 t-p에서 원천 상태가 Sa, Sb, Sg, Sh 중의 하나이고, C2를 만족한다면 t에서 Sc에 머무르게 된다. 그림 11의 H2는 t-p에서 원천 상태가 Sc이고 조건이 C3과 C4가 t-p에서 만족하지 않아서, t에서 계속 Sc에 머무르는 것을 보여준다.

(3) 시간-히스토리 변수

시간-히스토리 변수 노드는 히스토리 변수 노드와 거의 유사하지만 시간 천이(timed transition)가 추가되어 있는 점이 다르다. 즉, 천이에 시간 제약 조건 C[l,u]가 사용된다는 점이 다르다. *"C[l, u]"* 표시에서 조건 C가 지속적으로 만족된 후 l에서 u 시간 사이에 천이가 수행되어야 하는 것을 의미한다. 이 노드는 TTS(Timed Transition System)로 기술된다. 히스토리 변수 노드처럼 시간-히스토리 변수 노드의 TTS도 확장된 TTS로

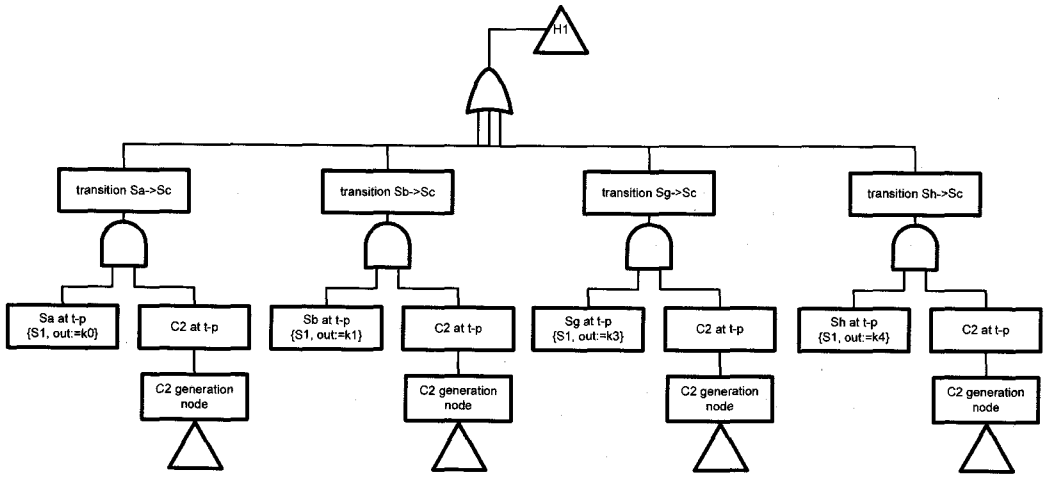


그림 10 히스토리 변수 노드의 고장 수목 예제 (2)

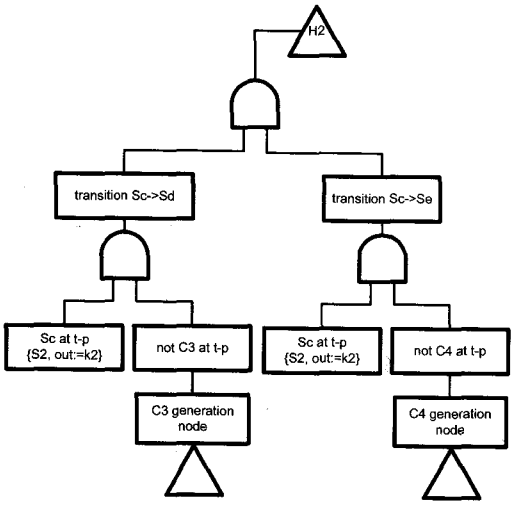


그림 11 히스토리 변수 노드의 고장 수목 예제 (3)

변경한 후, 템플릿을 사용하여 고장 수목을 전개한다.

시간-히스토리 변수 노드의 템플릿은 히스토리 변수 노드와 유사하지만, 시간 제약 조건의 비결정성(조건 C가 만족되면 l과 u 시간의 임의의 순간에 천이가 발생할 수 있다는 성격) 때문에 고장 수목의 템플릿이 복잡하게 정의된다. 그리고, 생성된 고장 수목도 히스토리 노드에 비해 복잡해진다. 자세한 변수 노드와 템플릿, 그리고 관련 예제는 [20]의 4.2.2절에 있다.

(4) 입력 변수 노드

가. 템플릿 정의

입력 변수 노드는 외부에서 FOD로의 입력을 의미한다. 따라서, 한 주기의 고장 수목을 구성할 때, 최하단에 나타나게 된다. 이 단계에서 고려되는 오류는 논리 오류나 계산 오류라기 보다는 외부에서 오는 값들이 제대로 전달되지 않는 경우나 입력 값이 없는 경우를 고려한다. 또한 입력이 하드웨어로부터 전달되는 경우와 소프트웨어로부터 전달되는 경우로 나누어서 생각할 수 있다. 이와 같은 종류의 오류는 [21-25]에서 광범위하게 연구되었다. 그림 12는 이러한 연구결과를 반영하여 작성된 입력 변수 노드의 고장 수목 템플릿이다.

4. 사례 연구

지금까지 설명한 내용을 실제 시스템에 적용하여 본 연구의 유용성을 확인하였다. 고장 수목 생성 방법을 적용한 시스템은 원전계측제어시스템 개발사업단(KNICS)에서 현재 개발중인 APR1400 모델의 원자로 보호 시스템(RPS)의 수동리셋 가변설정치 상승 트립 논리이다.

4.1 예제 시스템 소개

그림 1(a)는 수동리셋 가변설정치 상승 트립 논리에 대한 FOD이다. 이 FOD의 출력값은 예비 트립신호인 th_Pretrip과 트립 신호인 th_Trip이다. *"th_Trip = 0"* 이 되면 원자로는 정지(shutdown) 되는 중요한 신호이므로, 이 논리에 대한 안전성 분석은 필수적으로 수행되어야 한다.

이와 같은 FOD로부터 고장 수목을 구성하기 위하여는 우선 히스토리 노드와 시간-히스토리 노드의 FSM과 TTS를 확장된 FSM과 확장된 TTS로 변환하여야 한다. 그림 1(c)의 h_Spt와 그림 1(d)의 th_Trip에 대해 확장된 그래프는 각각 그림 13과 그림 14이다.

4.2 고장 수목 분석 예

본 논문에서 제안하는 템플릿을 이용하여 수동리셋 가변설정치 상승 트립 논리를 분석하였을때, 그림 15와 같은 고장 수목이 생성되었다. 사고 노드의 결정은

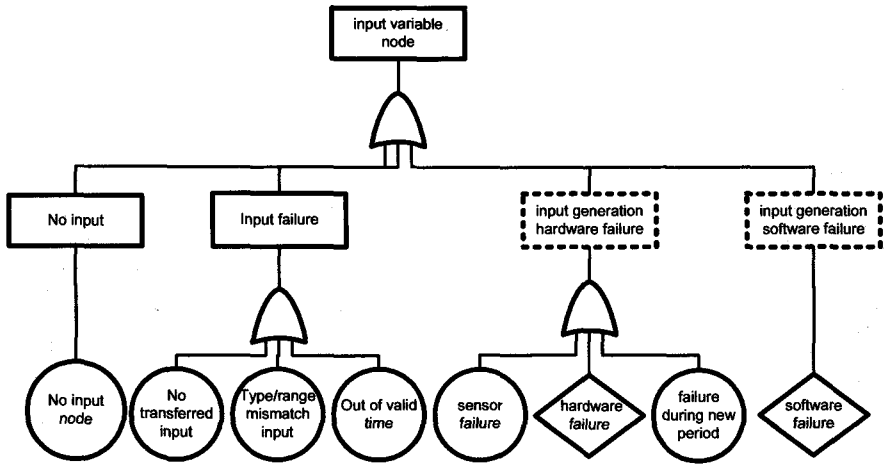


그림 12 입력 변수 노드의 고장 수목 템플릿

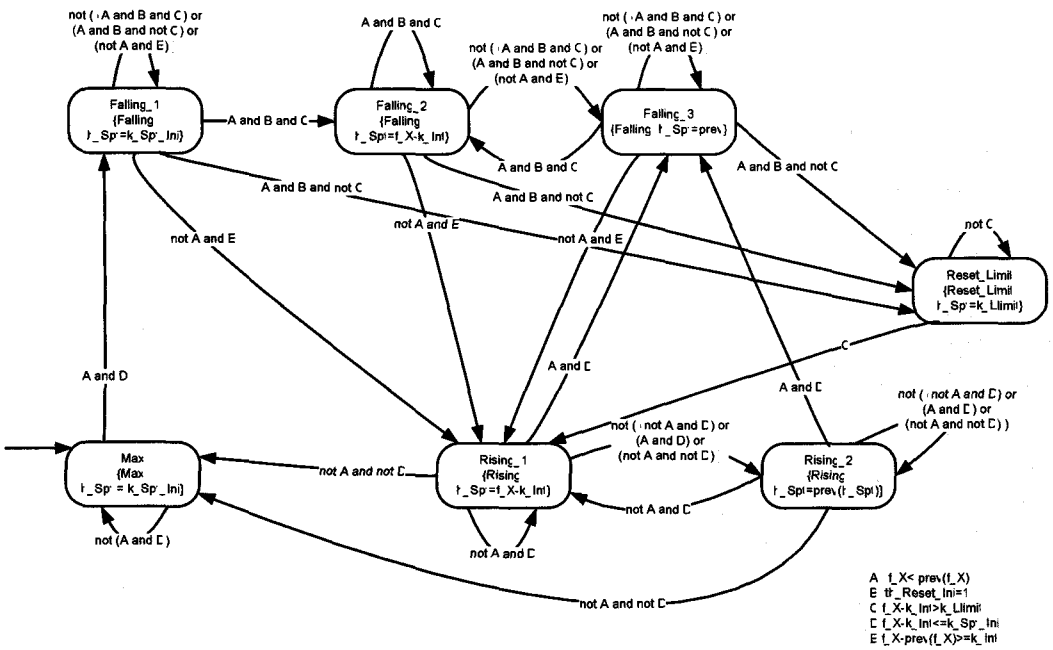


그림 13 h_Spt 히스토리 변수 노드에 대한 확장 FSM

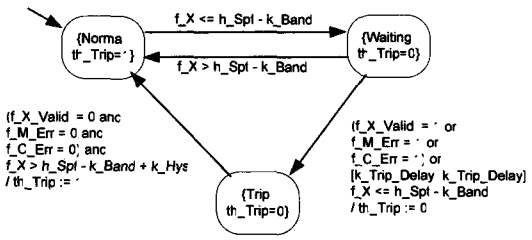


그림 14 th_Trip 시간-히스토리 변수 노드에 대한 확장 TTS

FMEA의 결과나 관련 분야 전문가에 의해서 수행된다. 본 예제에서는 트립이 발생한 경우(th_Trip = 0)를 사고로 설정하고 고장 수목을 전개하였다. 그림 15의 고장 수목의 오른쪽 하단의 회색 노드를 보면 *"Not (f_Valid=0 and f_M_Err=0 and f_C_Err=0 and f_X > h_Spt - k_Band + k_Hys)"*를 만족하면 계속 트립 상태를 유지(th_Trip = 0)한다. 이 조건은 *"f_Valid=0 and f_M_Err=0 and f_C_Err=0 and f_X > h_Spt - k_Band + k_Hys"*를 만족해야만 Trip 상태에서 Normal 상태로 천

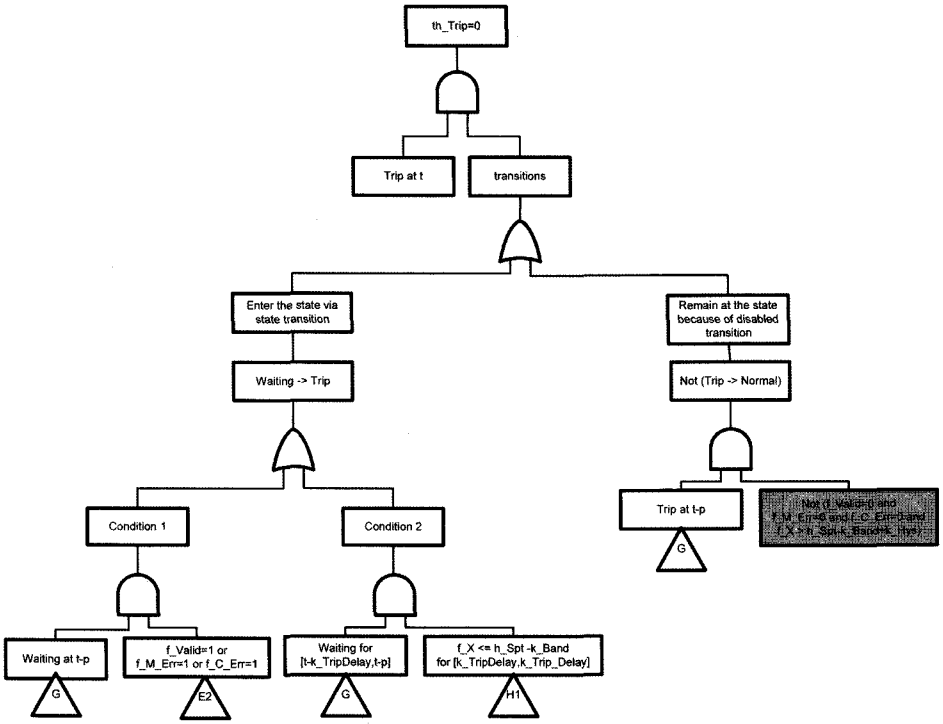


그림 15 오류가 있는 트립 논리의 고장 수목

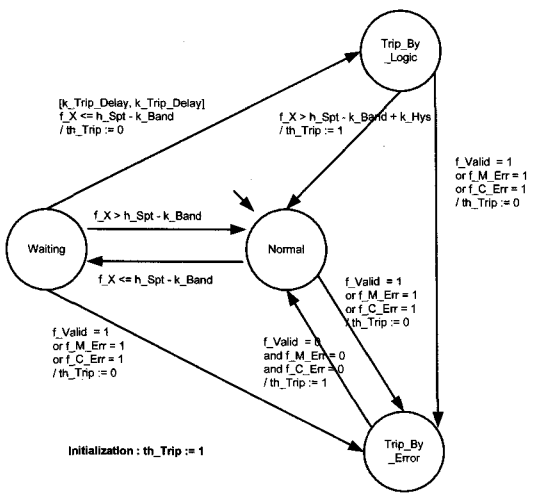


그림 16 수정된 th_Trip 시간-히스토리 변수 노드의 TTS

이할 수 있음을 의미한다. 이를 다시 나눠서 분석하면 모듈 오류(f_M_Err)나 채널 오류(f_C_Err) 등이 모두 정상일때도 추가적으로 *"f_X > h_Spt - k_Band + k_Hys"*가 만족해야만 트립이 해지됨을 유추할 수 있다. 이 것은 안전성과 관련된 불필요한 과중한 제약이다. 따라서, 그림 16과 같이 트립을 입력 변수 f_X 값의 이상에 의한 Trip_By_Logic과 모듈 오류나 채널 오류에 의해서 발생한 Trip_By_Error로 나누어야 한다. 본 연구에서 생성된 소프트웨어 고장 수목을 원자력 분야 전문가가 검토한 결과 매우 유용하다고 확인하였다.

5. 결 론

본 논문에서는 NuSCR 정형 명세 언어로 작성된 소프트웨어 요구 명세로부터 고장 수목을 체계적으로 생성하고 분석하는 방법에 대하여 기술하였다. 본 연구에서는 소프트웨어의 동작과 구조를 동시에 반영하는 통합된 고장 수목을 생성함으로써 소프트웨어의 안전성을 높이고자 하였다. 이를 위해 NuSCR로 작성된 요구 명세의 구성 요소에 대한 템플릿을 정의하고, 이들 템플릿을 사용하여 고장 수목을 구성하는 방법을 제안하였다. 또한, 제안된 방법의 유용성을 살펴보기 위해서 현재 원전계측제어시스템 개발 사업단에서 개발 중인 APR1400 모델의 원자로 보호 시스템의 일부 트립 논리에 대한 고장 수목을 생성하고 분석을 수행해 보았으며, 이 과정을 통해 시스템내에서 발생할 수 있는 중요한 오류를 발견하고 수정할 수 있었다. 이러한 과정은 원자력 분야 전문가의 검토를 통해 유용성을 확인하였다. 본 논문에서 제안한 방법을 통해 요구 명세의 작성과 함께 안전

성을 분석할 수 있게 되어, 개발 초기부터 안전성 분석이 이뤄지는 통합된 개발-검증 환경을 구축할 수 있을 것으로 기대된다.

본 연구에서는 NuSCR 로 기술된 소프트웨어의 고장 수목을 제안하였다. NuSCR이 다른 안전성이 중요한 시스템의 주요 특성을 가지고 있어서 대부분의 시스템에서 응용이 가능하지만, 일반적인 소프트웨어 고장 수목이라고 하기에는 부족하다. 따라서, 일반적인 소프트웨어의 고장 수목의 모델에 대해서 연구로 확장할 수 있다.

참 고 문 헌

[1] W. Velseley, F. Goldberg, N. Roberts, and D. Haasl, *Fault Tree Handbook*, NUREG 042, US Nuclear Regulatory Comission, US, 1981.

[2] T. Kim, S. Cha, D. Kim, and H. Chung, "NuSCR-software requirements specification language," in *Proceedings of the 5th Formal Methods Workshop (FMW 2001)*, June 2001.

[3] J. Yoo, T. Kim, S. Cha, J. Lee, and H. Son, "A formal software requirements specification method for digital nuclear plants protection systems," *Journal of Systems and Software*, vol. 74, pp. 73-83, 2005.

[4] N. Leveson and P. Harvey, "Analyzing software safety," *IEEE Transactions on Software Engineering*, vol. 9, pp. 569-579, Sept. 1983.

[5] N. Leveson, S. Cha, and T. Shimeall, "Safety verification of ada programs using software fault trees," *IEEE Software*, vol. 8, pp. 48-60, July 1991.

[6] S. Min, Y. Jang, S. Cha, Y. Kwon, and D. Bae, "Safety verification of ada95 programs using software fault trees," in *Proceeding of SAFECOMP 1999*, pp. 226-238, 1999.

[7] C. Garret, S. Guarro, and G. Apostolakis, "The dynamic flowgraph methodology for assessing the dependability of embedded software systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 5, pp. 824-840, 1995.

[8] M. Yau, S. Guarro, and G. Apostolakis, "Demonstration of the dynamic flowgraph methodology using the titan ii space launch vehicle digital flight control system," *Reliability Engineering and System Safety*, vol. 49, pp. 335-353, 1995.

[9] M. Yau, G. Apostolakis, and S. Guarro, "The use of prime implicants independability analysis of software controlled systems," *Reliability Engineering and System Safety*, vol. 62, pp. 23-32, 1998.

[10] C. Garrett and G. Apostolakis, "Automated hazard analysis of digital control systems," *Reliability Engineering and System Safety*, vol. 77, pp. 1-17, 2002.

[11] Y. Papadopoulos, J. McDermid, R. Sasse, and G. Heiner, "Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure," *Reliability Engineering and System Safety*, vol. 71, pp. 229-247, 2001.

[12] Y. Papadopoulos and M. Maruhn, "Model-based synthesis of fault trees from matlab-simulink models," in *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN '01)*, pp. 77-82, IEEE, Jul. 2001.

[13] K. Vemuri, J. Dugan, and S. Sullivan, "Automatic synthesis of fault trees for computer-based systems," *IEEE Transactions on Reliability*, vol. 48, pp. 394-402, Dec. 1999.

[14] K. J. Sullivan, J. B. Dugan, and C. Coppit, "The Galileo fault tree analysis tool," in *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, (Madison, Wisconsin), pp. 232-235, Jun. 1999.

[15] KNICS, "Knics (korea nuclear instrumentation and control system research and development center)," 2001.

[16] D. Parnas and J. Madey, "Functional documentation for computer systems engineering," Tech. Rep. Technical Report CRL No. 273, Telecommunications Research Institute of Ontario, McMaster University, 1991.

[17] E. Yourdon and L. Constantine, *Structured design: fundamentals of a discipline of computer program and systems design*, Prentice-Hall, 1986.

[18] T. Henzinger, Z. Manna, and A. Pnueli, "Timed transition systems," in *Proceedings REX Workshop on Real-Time: Theory in Practice*, vol. 600 of *Lecture Notes in Computer Science*, pp. 226-251, Springer-Verlag, 1992.

[19] AECL, *Software Work Practices -Procedure for the specification of Software Requirements for Safety Critical Software*, Sep. 1991.

[20] T. Kim, *Property-based Theorem Proving and Template-based Fault Tree Analysis of NuSCR Requirements Specification*, Ph.D Dissertation, KAIST, 2005.

[21] M. Hsueh, T. Tsai, and R. Iyer, "Fault injection techniques and tools," *IEEE Computer*, vol. 30, pp. 75-82, Apr. 1997.

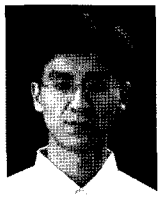
[22] M. Sullivan and R. Chillarege, "Software defects and their impact on system availability - A study of field failures in operating systems," in *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, pp. 2-9, 1991.

[23] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: The MEFISTO tool," in *Proceedings of the 24th Annual International Symposium on Fault-Tolerant Computing*, (Los Alamitos, CA, USA), pp. 66-75, IEEE Computer Society Press, Jun. 1994.

[24] H. Ammar, S. Yacoub, and A. Ibrahim, "A fault

model for fault injection analysis of dynamic UML specifications," in *Proceedings of the ISSRE 2001*, 2001.

[25] J. Christmansson and R. Chillarege, "Generation of an error set that emulates software faults based on field data," in *Proceedings of the Twenty-Sixth International Symposium on Fault-Tolerant Computing*, 1996.



김 태 호

1995년 성균관대학교 정보공학과 학사 1997년 한국과학기술원 전산학과 석사 2005년 한국과학기술원 전자전산학과 전산학전공 박사. 2005년~현재 한국전자통신연구원 임베디드SW연구단 선임연구원. 관심분야는 소프트웨어 공학, 정형 명세 및 검증, 프로그램 정적 분석, 소프트웨어 프로세스

유 준 범

정보과학회논문지 : 소프트웨어 및 응용
제 32 권 제 2 호 참조

차 성 덕

정보과학회논문지 : 소프트웨어 및 응용
제 32 권 제 2 호 참조