실시간 모델 체커를 이용한 폴트 트리의 체계적 검증
(Systematic Evaluation of Fault Trees using Real-Time Model Checker)

지 은 경[†] 차 성 덕[††] 손 한 성[†††] 유 준 범[†]
(Eun-Kyoung Jee) (Sung Deok Cha) (Han Seong Son) (Junbeom Yoo)
구 서 룡[††††] 성 풍 현[†††††]
(Seo-Ryong Koo) (Poong Hyun Seong)

요 약 폴트 트리 분석(Fault Tree Analysis)은 산업계에서 가장 널리 사용되는 안전성 분석 기법 중의 하나이다. 하지만, 이 기법은 보통 수작업으로 이루어지며, 분석 결과를 체계적이고 자동적으로 검증할 수 있는 방법이 없다는 약점을 지닌다. 본 논문에서는 실시간 모델 체커인 UPPAAL을 이용하여 안전성이 중요한 소프트웨어의 요구 사항들을 정형 명세하고, 수작업으로 완성된 폴트 트리의 정확성을 검증하는 방법을 제안하고 있다. 제안된 방법을 유용성을 확인하기 위해서 월성 원자력 발전소의 비상 정지 소프트웨어(Wolsung SDS2)에서 사용된 기능 요구 사항들을 예제로서 사용하였다. 폴트 트리는 월성 SDS2에 대한 전문적인 지식을 지니고 폴트 트리를 이용한 안전성 분석을 여러 번 수행해 본 경험이 있는 대학원생들에 의해 작성되었다. 기능 요구 사항들은 UPPAAL의 입력으로서 사용되기 위해서 시제 오토마타의 형태로 수작업으로 변환되었으며, 이 폴트 트리의 정확성을 검증하기 위해서 모델 체킹을 사용하였다. 본 논문에서 제안된 방법을 월성 SDS2 예제에 적용해 본 결과, 수작업으로 작성된 폴트 트리에 존재하는 오류를 찾을 수 있었으며, 이러한 작업을 통하여 제안된 방법이 폴트 트리 분석에 대한 신뢰도를 높이는데 유용함을 발견하였다.

키워드 : 소프트웨어공학, 정형기법, 요구분석기법, 폴트 트리, 모델 체킹

Abstract Fault tree analysis is the most widely used safety analysis technique in industry. However, the analysis is often applied manually, and there is no systematic and automated approach available to validate the analysis result. In this paper, we demonstrate that a real-time model checker UPPAAL is useful in formally specifying the required behavior of safety-critical software and to validate the accuracy of manually constructed fault trees. Functional requirements for emergency shutdown software for a nuclear power plant, named Wolsung SDS2, are used as an example. Fault trees were initially developed by a group of graduate students who possess detailed knowledge of Wolsung SDS2 and are familiar with safety analysis techniques including fault tree analysis. Functional requirements were manually translated in timed automata format accepted by UPPAAL, and the model checking was applied using property specifications to evaluate the correctness of the fault trees. Our application demonstrated that UPPAAL was able to detect subtle flaws or ambiguities present in fault trees. Therefore, we conclude that the proposed approach is useful in augmenting fault tree analysis.

Key words : software engineering, formal method, requirement engineering, fault tree, model checking

· 본 연구는 첨단과학기술 연구센터를 통하여 과학재단의 지원을 받았음.

† 비 회 원 : 한국과학기술원 전자전산학과
ekjee@salmosa.kaist.ac.kr
jbyoo@salmosa.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
cha@salmosa.kaist.ac.kr
††† 비 회 원 : 한국원자력연구소(RAERI), MMIS팀
hsson@nanum.kaeri.re.kr
†††† 비 회 원 : 한국과학기술원 원자력공학과
srkoo@cais.kaist.ac.kr
††††† 비 회 원 : 한국과학기술원 원자력 및 양자공학과 교수
phseong@sorak.kaist.ac.kr

논문접수 : 2001년 8월 31일
심사완료 : 2002년 9월 16일

1. 서 론

폴트 트리 분석(Fault Tree Analysis)은 안전성 분석의 한 방법으로서 시스템의 특정한 위험 요소에 대해 트리를 그려나가면서 그 원인들을 분석하는 도구이다. 폴트 트리 분석은 시스템의 고장 모드와 고장 결과들뿐만 아니라, 시스템의 기능들에 대한 지식을 기반으로 분석을 하는데, 이 분석을 통해서는 특정한 이상 기능에 대해 그것을 일으키는 원인이 되는 컴포넌트들의 조합을 결과로 얻을 수 있다. 폴트 트리 분석은 시스템의 안전성을 분석하는데 유용한 도구로 쓰여져 오고 있으며 시스템의 안전성을 확증하는 자료로 필수적으로 요구되기도 한다. 그러나, 이 방법은 보통 사람의 수작업으로 이루어지며 체계적인 구성 방법이 없는 실정이어서 구성 과정이 자동화 되기 힘들고, 특히 폴트 트리가 표현하는 내용이 부정확할 수 있지만 오류여부를 체계적으로 확인하는 작업이 어렵다. 본 논문에서는 모델 체킹(Model Checking) 방법을 이용하여 폴트 트리의 정확성을 검증하고, 부정확한 부분은 보정할 수 있는 방법을 제안하였다.

모델 체킹은 유한 상태 시스템에 대한 분석방법으로 널리 쓰이는 자동 검증 방법 중 하나이다. 시스템이 만족해야할 속성을 시제 논리 공식으로 표현하고, 시스템의 실제 행위를 병렬 상태-전이 그래프로 나타내면, 모델체커는 시스템이 그 속성을 만족하는지 여부를 자동적으로 검증해 준다. 만일 속성이 만족되지 않는 경우는, 반례를 생성하는데 이 정보는 폴트트리의 검증에 매우 유용하게 쓰일 수 있다. 모델 체커를 통한 검증을 통해 우리는 폴트 트리에서 잘못된 부분을 찾아낼 수 있고, 검증 결과를 분석함으로써 잘못된 부분을 정확하게 보정하는 정보들을 얻을 수도 있다.

본 논문에서는 실시간 시스템의 분석을 대상으로 하며 UPPAAL을 사용하였다. UPPAAL은 실시간 시스템을 모델링, 시뮬레이션하고 검증할 수 있는 통합된 환경을 제공하는 도구로서, 유한한 제어 구조와 실수 클럭을 가지는 비결정적 프로세스들의 집합으로 모델될 수 있는 시스템들에 적합하다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 본 논문의 연구 배경인 폴트 트리 분석과 모델 체킹 방법을 알아본다. 3장에서는 폴트 트리 검증에 모델 체킹 방법을 사용하는 전체 과정을 설명하며, 4장에서는 이 방법을 실제 예제에 적용해 본 과정과 결과를 기술한다. 마지막으로 5장에서는 결론 및 향후 연구에 대해 설명한다.

2. 연구 배경

2.1 폴트 트리 분석

폴트 트리 분석(Fault Tree Analysis)[1]은 1960년초에 제안된 이후, 전자 및 원자력등 산업 분야 전반에서 널리 사용되고 있다. FTA는 본질적으로 위험 요소를 발견하는 도구가 아니라 특정 위험 요소에 대한 원인을 분석하는 방법이다. 이것은 시스템의 위험성 있는 고장 모드와 그 고장 모드의 영향, 시스템의 기능들에 대한 지식을 기반으로 한 top-down방식의 안전성 분석 방법으로서, 분석의 결과는 특정한 이상 기능을 일으키는 원인이 되는 세부 구성요소들 고장의 조합이 된다.

폴트 트리는 시스템의 중요한 위험 상태를 나타내는 root를 가지며, 이 최상위 이벤트(top event)는 먼저 다른 방법에 의해 확인되어야 한다. 최상위 이벤트가 확인되면 그것의 가능한 원인을 드러내기 위해서 시스템을 분석한다. FTA는 위험한 이벤트를 일으키는 원인이 되는 오류들의 조합을 기술하기 위해서 Boolean 논리를 사용한다. 트리의 각 레벨은 상위 레벨의 문제를 일으키는 필요충분한 더 기본적인 이벤트들을 나열한다. 최상위 노드 (top event)와 최하위 잎노드(leaf node) 사이의 중간 노드들 (intermediate nodes)은 기본적인 이벤트들의 단순한 조합 또는 집합이다. 일단 트리가 구성되고 나면 그것은 Boolean식으로 쓰여질 수 있고, 최상위 이벤트를 야기시키기에 충분한 기본 이벤트들의 조합으로 간단히 표현될 수 있다.

2.2 실시간 모델 체킹

모델 체킹은 유한 상태 병행 시스템을 검증하는 자동화된 방법이다. 병행 시스템의 전체 상태 그래프는 유한한 Kripke 구조[2]로 타내어지고, 이 구조가 특별한 시제 논리공식(시스템이 만족해야 할 속성)의 모델이 되는지를 자동적으로 결정한다. 실수 변역을 가진 시간상에서의 모델 체킹은 상태 전이 그래프와 정량적인 시간 정보를 가진 논리 공식들을 결합하는 것이다[3].

시간 제약이 정확도에 중요한 시스템들을 실시간 시스템이라고 한다. 실시간 시스템 어플리케이션의 예로는 비행기 제어기, 산업 기계들과 로봇 등을 들 수 있다. 이러한 시스템들은 보통 어려운 실시간 제약을 만족해야만 하는데, 항공기의 착륙 장치가 언젠가는 내려간다.' 라고만 말하는 것으로는 충분하지 않다. 그 이벤트가 다른 이벤트들과 관련해서 언제 일어날 수 있는지에 대한 상한과 하한이 있다. 이러한 시스템들에서의 버그들은 모호하고, 수정하는데 많은 비용이 들 수 있기 때문에 (생명을 위협하는 것이 될 수도 있다.) 정확성이 중요하

다. 그러나 반응 시스템에 대한 시제 증명에 대한 전통적인 방법들은[4,5,6,7] 시간에 대한 정성적인 성질들만 유지한 채, (예, 언젠가는 p가 만족한다.) 정량적인 시간에 대해서는 추상화했다. 그러나 Alur외 몇 명의 연구진에 의해 실수 변역을 가지는 정량적인 시간 모델에 대한 모델 체킹 알고리즘이 제안되고 그 알고리즘이 결정적임이 증명되었다[3]. 그 이후에도 정량적 시간 모델에 대한 모델 체킹에 관한 많은 연구들이 이어졌다[8, 9,10,11].

실시간 시스템을 모델 체킹하기 위해서는 긴밀 시간 모델(dense time model)을 바탕으로 해서 시스템을 타임드 오토마타 등으로 모델하고, 시제 논리로 속성을 기술한 후 그들에 대해 모델 체킹을 수행한다. SPIN, SMV등 기존의 전형적인 비실시간 모델 체커들은 따로 클럭 변수들을 가지지 않으나 실시간 모델 체커에서는 클럭 변수들을 명시하고 그것들을 이용하여 모델에서의 시간 제약과 속성에서의 시간 제약을 표현한다. 실시간 모델 체커는 시스템 모델과 속성이 시간 제약을 포함하기 때문에, 영역 기반 모델 체킹 방법 등 기존의 모델 체커들과 다른 접근 방법이 사용된다. 안전성이나 제한된 존속성에 대한 속성들을 검증하려고 할 때, 시간에 대한 제약 등을 표현하기 위해서 시제 논리(timed logic)가 사용된다. 본 논문에서는 실시간 모델체커들 중에서 효율성이 좋고 사용하기 편리한 UPPAAL[12]을 사용하여 검증하였다.

UPPAAL은 실시간 시스템을 모델링하고, 시뮬레이션하고 검증하는 도구인데, Aalborg 대학과 Uppsala대학의 연구결과로 개발되었다. 이 도구는 유한한 제어 구조와 실수 값을 가지는 클럭들을 가진 비결정적인(non-deterministic) 프로세스들의 집합으로 모델 될 수 있는 시스템들에 적합하다. 이 프로세스들은 채널과 공유 변수들을 통해 통신한다. 시간요소가 중요한 실시간 제어기와 통신 프로토콜 등이 전형적인 어플리케이션들이다.

UPPAAL은 시스템의 행위를 클럭과 데이터 변수들을 가지는 확장된 오토마타들의 네트워크로써 기술하는 명세 언어 부분(Editor)과, 시스템의 디자인 단계동안 시스템의 가능한 동적 실행들을 조사할 수 있게 해주는 시뮬레이터(Simulator) 부분, 시스템의 심볼릭 상태 공간을 조사함에 의해 불변식(invariant)과 제한된 존속성(bounded-liveness properties)을 검사하는 모델 체커(Verifier)의 세 부분으로 구성되어 있다. 모델링과 디버깅을 쉽게 하기 위해서 UPPAAL은 속성이 시스템 모델에서 만족되지 않았을 경우 처음 상태부터 그때까지의 경로를 보여주는 진단 경로를 자동적으로 생성해 준다.

UPPAAL은 이제까지 Bang과 Olufsen 파워 제어기[13], 충돌 회피 프로토콜[14,15], 기어박스(Gearbox)제어기[16], LEGO®MINDSTROMS™시스템, TDMA 프로토콜 시동 매커니즘[17], 필립스 오디오 제어 프로토콜[18] 등을 포함한 여러 산업 사례 연구에 사용되었다.

3. 폴트 트리를 보정하기 위한 전체 과정

본 장에서는 폴트 트리를 보정하기 위해서 모델 체킹 방법을 사용하는 전체적인 과정을 소개한다. 모델체커에서의 디자인 모델과 그 부분에 해당하는 폴트 트리가 주어졌을 때, 다음과 같은 과정이 수행된다.

1. 폴트 트리의 노드들에 대해 검증 공식을 만든다
2. 시스템 모델과 이 검증 공식에 대해 모델 체커로 검증을 수행한다
3. 검증 결과를 분석한다
4. 폴트 트리의 수정할 부분을 보정한다

3.1 공식 만들기

3.1.1 공식의 구문론(syntax)

먼저 폴트 트리의 노드들에 대해 공식을 만든다. 어떤 모델 체커가 사용되느냐에 따라 사용할 수 있는 공식의 형태가 약간씩 다르다. 본 논문에서는 시간 제약을 쉽고 정확하게 기술하기 위해서 실시간 모델 체커 UPPAAL을 사용하며, UPPAAL에서 검증될 수 있는 속성의 형태는 다음과 같다.

$$\varphi ::= \forall\Box\beta \mid \exists\Diamond\beta$$

$$\beta ::= \alpha \mid \neg\beta \mid (\beta) \mid \beta\vee\beta \mid \beta\rightarrow\beta$$

여기서 α는 클럭 변수나 데이터 변수에서의 기본 공식 또는 프로세스 내에서의 위치이다. 기본 공식은 각 변수들(클럭 변수 또는 데이터 변수)의 정수값에서의 범위를 나타내거나 (예, $1\le x\le 5$) 또는 두 변수 사이의 차이에 대한 범위를 나타낸다.(예, $3\le x-y\le 7$). 직관적으로, $\forall\Box\beta$ 는 모든 경로에 대해서 모든 상태에서 가 만족되어야 한다는 제약으로, 이 공식이 만족되기 위해서는 모든 도달 가능한 상태들에서 β 공식이 만족되어야 한다. 비슷하게, $\exists\Diamond\beta$ 식은 임의의 상태에서 언젠가는 β 가 만족되어야 한다는 제약이다.

UPPAAL에서 'A[]'는 '$\forall\Box$'을 나xk고, 'E<>' 는 ,' $\exists\Diamond$'을 나타내는데, 그 예를 들면 다음과 같다:

- E<> (p1.cs and p2.cs) - 시스템이 프로세스 p1과 p2가 둘다 cs위치에 있는 상태에 도달할 수 있어야 한다.
- A[] (c2 > 30 imply a==0) - 모든 상태에서, 클럭 변수 c2가 30보다 크면, 데이터 변수 a가 0 이어야 한다.

3.1.2 검증 속성의 일반적 형태

폴트 트리를 검증하기 위한 공식은 일반적으로 두 가지의 형태를 띠게 된다. 결과는 Property is satisfied 또는 Property is not satisfied 중 하나이며, 그 결과에 대한 해석은 다음과 같다.

1) 노드 이벤트의 존재성 검증

어떤 노드 N에 대해서, 그 노드 이벤트가 일어나는 상황을 나타내는 공식을 p_N 이라고하면,

$$\forall\Box(\neg PN)\Box\ (p_N) \qquad (1)$$

이라는 모델체킹 공식은 '시스템의 모든 상태에 대해 공식 p_N 이 거짓이다.'라는 의미의 제약식으로서, 노드 이벤트가 정말 발생가능한 것인지를 체크한다. 검증 결과에 따라 다른 해석이 부여된다.

• 속성이 만족되었을 때 (Property is satisfied) - 검증 속성이 모델에서 만족되었을 때, 이것은 노드 N 이벤트가 결코 일어나지 않음을 의미한다. 즉, 최상위 노드 이벤트를 일으키는 가능한 원인 중 하나인 N 이 발생하는 상태까지 시스템이 결코 도달하지 않는다는 이야기이다. 폴트트리 이벤트의 기술에 오류가 있음을 알 수 있다.

• 속성이 만족되지 않았을 때 (Property is not satisfied) - 검증 속성이 모델 체킹에서 만족되지 않았다면, 그것은 노드 N 에 의해 표현된 상황이 정말 일어날 가능성이 있으며 폴트 트리의 노드 N 이 적절히 기술되었다는 것이다. 최상위 위험 상황에 대한 가능한 원인을 기술하고 있으므로, 이 부분에 대해 폴트 트리를 제대로 그렸다는 확인을 한 것이다. 이 경우에 이렇게 확인된 위험 원인을 제거하기 위해서 시스템에 적절한 조치가 가해져야 할 것이다. 검증 결과로부터 또 하나 알 수 있는 사실은 속성이 만족되지 않았을 때, 모델 체커는 시스템의 처음 상태로부터 속성이 만족되지 않는 상태까지의 하나의 경로(path)를 제공한다. 이런 오류 경로를 자세히 분석함으로써, 어떤 경우에 N 노드의 상황이 벌어지는지를 알 수 있고, 그 상황에 이르지 않기 위해서 어떤 조치를 취해야 하는가 등 위험요소를 제거하기 위한 정보들도 얻을 수도 있다. UPPAAL 은 특히 시뮬레이션 정보를 스크린을 통해 그래픽하게 볼 수 있게 제공해 준다.

2) 노드들 조합의 정확성 검증

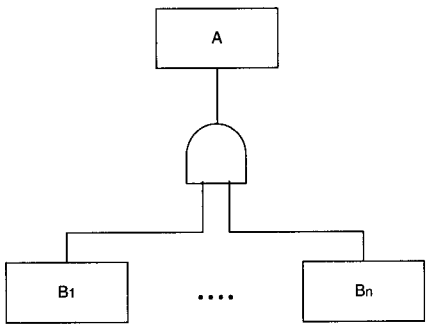
본 논문에서는 폴트 트리에 대해서 지연 계산법 (duration calculus) 시맨틱[19]을 따른다. 지연 계산법 시맨틱의 정의에 따라, 우리는 폴트 트리가 정확하게 구성되었는지를 체크한다. 이를 위해서는, 노드들을 조합하는 게이트의 종류에 따라 다른 공식을 사용하여야 하는데, 다음과 같은 검증공식들을 적용할 수 있다.

AND gate

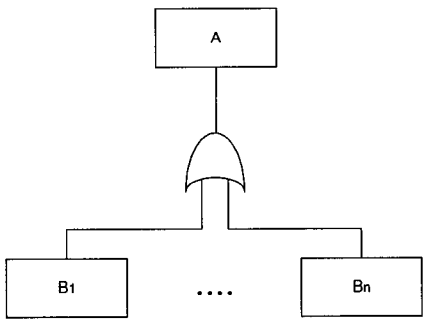
$$\forall\Box((B_1\wedge\cdots\wedge B_n)\leftrightarrow A) \qquad (2)$$

OR gate

$$\forall\Box((B_1\vee\cdots\vee B_n)\leftrightarrow A) \qquad (3)$$



(a) AND-게이트



(b) OR-게이트

그림 1 게이트로 연결된 폴트 트리들

• 속성이 만족되었을 때 (Property is satisfied) - 이 결과는 게이트에 의한 노드들의 결합이 정확하다는 것을 말해 준다. 예를 들어, 공식 2, □ ((B1 … Bn) A) 가 시스템에서 만족되었다면, 이것은 그림 1(a)의 노드 B1 … Bn 들이 AND 게이트에 의해 연결되어 A 의 원인이 되는 것이 정확하다는 것을 보증하는 것이다. 즉, A가 만족되는 것과 B1 부터 Bn 까지가 동시에 만족되는 것과는 같은 것이다 라는 것이 검증된 것이다.

• 속성이 만족되지 않았을 때 (Property is not satisfied) - 검증이 실패한 데는 여러 가지 이유가 있을 수 있다. 게이트에 의한 연결이 부적절한 경우 즉, 하위 노드들과 상위 노드간에 연결 시맨틱이 맞지 않는 경우가 있고, 폴트 트리가 이벤트의 내용을 잘못 기술하

고 있는 경우도 있고, 이벤트의 내용이 불충분하게 기술되어 모호함이 잘못의 원인이 되는 경우도 있다.시스템에서 속성이 만족되지 않았을 때, 우리는 시스템의 처음 상태로부터 속성이 만족되지 않는 때까지의 시뮬레이션 트레이스를 얻을 수 있다. 이 시뮬레이션 트레이스는 면밀히 분석되어야 한다. 이 분석을 통해서 때때로 우리는 폴트 트리의 부정확한 부분이나, 불충분한 부분들을 발견할 수 있고 경우에 따라서는 모델 자체의 오류를 발견할 수도 있다. 속성이 만족되지 않은 경우, 시뮬레이션 트레이스로부터 폴트 트리를 보정하기에 충분한 정보를 얻을 수 있으면, 폴트 트리의 불완전한 부분들을 보완할 수 있을 것이다.폴트 트리 노드가 부정확할 수 있는 가능한 이유들과 그에 대한 폴트 트리의 보정 방법을 찾아보면 다음과 같다

■ 경우 1 : 노드 내용이 부정확하게 기술되었다
- 폴트 트리에서 부정확한 부분을 확인하고, 공식을 수정해서 다시 만든다
- 바뀐 공식을 모델 체커를 이용해 검증한다
- 검증 결과 바뀌어진 공식이 정확하다면, 이 공식의 내용에 맞게 원래 폴트 트리 노드의 내용을 다시 쓴다

■ 경우 2 : 공식에서 가정의 내용이 충분히 기술되지 않았다
- 특정한 폴트 트리노드에서 가정되고 있는 조건들을 확인한다
- 가정된 조건들을 공식에 AND 연산자를 사용하여 추가한다
- 바뀐 공식을 모델 체커를 통해 검증한다
- 바뀐 공식이 만족되면, 바뀐 공식에 따라 폴트 트리 노드의 내용을 다시 쓴다
- 바뀐 공식이 만족되지 않으면, 수정 작업을 다시 처음부터 시작한다

■ ...

폴트 트리를 보정하기 위해 필요한 모델 및 속성에 관한 정보가 자세하게 정리되어 있다면, 폴트 트리의 정확성 여부에 관한 상세한 분석이 가능할 것이다. 그러나 모델 체킹 방법이 폴트 트리를 정확하게 하는데 도움을 줄 수는 있지만, 폴트 트리에 존재할 지도 모르는 모든 오류를 자동화된 방법으로 찾아주는 것이 아니라는 점 또한 분명하다.

4. 사례 연구

본 논문에서 제안된 방법이 산업계에서 개발되거나 운영중인 실제 시스템에도 적용 가능함을 보이기 위하여 월성 원자력 발전소[20]에서의 소프트웨어 정지 계통(Shutdown System 2 : SDS 2)의 일부 기능을 선택하였다. 이를 위하여 자연어로 기술된 월성 시스템의 프로그램 기능 요구명세(Program Functional Specification : PFS)를 기반으로 이 시스템의 기능을 잘 이해하는 기술진이 폴트 트리분석을 하였으며, 그 결과를 확인하기 위하여 PFS를 기반으로 만들어진 timed 오토마타들로 이루어진 시스템 모델에 대한 모델 체킹을 통해 검증 및 보정하는 작업을 수행하였다. SDS2는 원자로의 상태를 계측하면서 원자로의 상태가 비정상적이 되면 정지 신호를 발생시키는데, 본 시스템에 있는 여섯개의 정지 신호 중, 본 논문에서는 시간 제약을 포함하는 Primary Heat Transport Low Core Differential Pressure(PDL) 정지 신호 관련 부분을 예제로 선택하였다. PDL 정지 신호 처리부(PDL trip part)는 센서들로부터 압력과 파워 값들을 받아들여서 계산하고 비교한 후, 비정상적인 상태로 판단되면 Trip출력을 내보내고 그렇지 않은 정상적인 상태일 때는 NotTrip 출력을 내보낸다.

4.1 실시간 모델 체킹

타임드 오토마타 모델의 일부분에 대해 살펴보자. 그림 2은 PDLCond 프로세스이다. PFS에서 PDLCond에 관한 명세의 일부분을 보면 다음과 같다. 이 자연어 명세와 이에 해당하는 visual timed automata의 GUI기반의 그리고 텍스트기반의 모델은 다음과 같다.

If the D/I is open, select the 0.3% FP conditioning level. If Ψ_{LOG} < 0.3% FP - 50mV, condition out the immediate trip. If Ψ_{LOG} 0.3% FP, enable the trip.

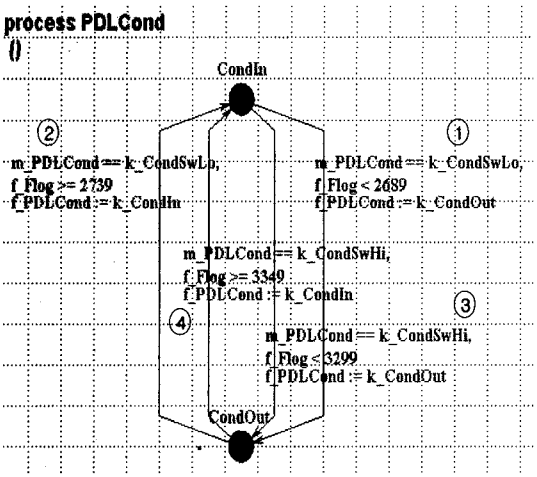


그림 2 PDLCond 프로세스

```
process PDLCond{
state CondIn, CondOut;
init CondIn;
trans CondIn → CondOut {
    guard m_PDLCond == k_CondSwLo, f_Flog < 2689;
    // (Ψ_LOG < 0.3% FP - 50mV) → ①
    assign f_PDLCond := k_CondOut;
    },
    CondOut → CondIn {
    guard m_PDLCond == k_CondSwLo, f_Flog >= 2739;
    // (Ψ_LOG == 0.3% FP) → ②
    assign f_PDLCond := k_CondIn;
    },
    CondIn → CondOut {
    guard m_PDLCond == k_CondSwHi, f_Flog < 3299;
    // (Ψ_LOG < 0.5% FP - 50mV) → ③
    assign f_PDLCond := k_CondOut;
    },
    CondOut → CondIn {
    guard m_PDLCond == k_CondSwHi, f_Flog >= 3349;
    // (Ψ_LOG == 0.5% FP) → ④
    assign f_PDLCond := k_CondIn;
    };
}
```

m_PDLCond는 (접두어 m_은 모니터 변수를 의미한다) 조건 레벨에 대한 모니터 변수로서, m_PDLCond의 값이 k_CondSwLo이면, 0.3% FP 조건 레벨이 선택된 것을 의미하고, k_CondSwHi이면 5% FP 조건 레벨이 선택된 것을 의미한다. f_Flog는 Ψ_{LOG} 를 나타내는 함수이다. 프로세스 PDLCond는 두 가지 상태를 가지고 있는데 (CondIn과 CondOut), 이것은 f_PDLCond 변수의 값이 k_CondIn 또는 k_CondOut 두 가지일 수 있다는 것과 같은 의미이다. 0.3% FP 로그 파워 레벨은 2739mv이고, 5% FP 로그 파워 레벨은 3349mv이다.

If $_{LOG}$ < 0.3% FP - 50mV,
condition out the immediate trip.

위의 명세에서 이 부분은 그림 2의 PDLCond 프로세스에서 맨 오른쪽의 전이에 해당하며, 구체적인 변수들을 사용해서 다음과 같은 표현될 수 있다.

If m_PDLCond == k_CondSwLo and f_Flog
< 2689(=2739-50),
then f_PDLCond := k_CondOut.PDLCond

프로세스의 다른 전이들도 이와 같은 식으로 모델되었다. 이런 식으로 PFS의 PDLTrip 부분이 timed automata로 기술되었다. PDLTrip 부분 명세를 모델하기 위해 PDLCond(2) 프로세스 이외에도 FaveC(4), Flog(2), mFlog(7), mPDLCond(2), mPHTD(3), PDLDly(4), PDLSnrDly(2), PDLSnrI(2), PDLTrip(2), pTimer(2) 프로세스들이 있다. 프로세스 이름 옆의 괄호의 숫자는 각 프로세스의 스테이트 수를 나타낸다. PDLTrip에 대한 전체 스테이트 수는 2^{15} 여개 정도이다.

4.2 폴트 트리 분석

PDLTrip 부분에 대한 폴트 트리가 그림 3에 나타나 있다. 이 폴트 트리는 월성 시스템에 대한 자연어 명세 PFS에서 한 페이지 분량의 PDLTrip에 대한 부분을 보고 그린 것이다. 이 시스템 명세에 대해 여러 번의 검토를 거쳤고 상세한 지식을 가지고 있는 박사과정 학생을 포함한 세 명의 석박사 과정 학생 그룹이 2시간 정도에 걸쳐 그린 그림이다.

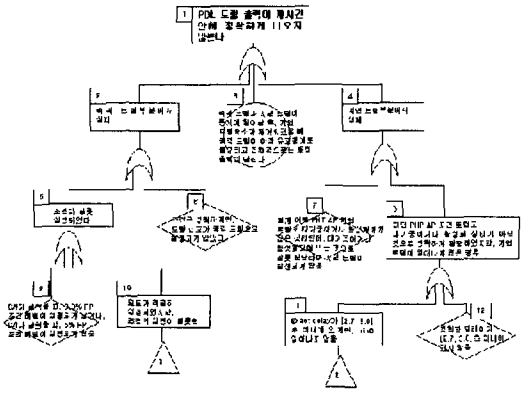


그림 3 월성 PDLTrip 부분에 대한 폴트 트리 일부

최상위 이벤트는 "PDL 트립 출력이 제시간 안에 정확하게 나오지 않는다." 이다. 폴트 트리 분석은 최상위 이벤트가 확인된 상황에서 시작한다. 최상위 위험 이벤트를 찾는 것은 폴트 트리 분석이 아닌 다른 방법에 의해 행해지며, 이 논문의 범위가 아니다. 우리는 월성 시스템의 다른 단계에서 이미 확인된 최상위 위험 이벤트를 가지고 분석을 시작한다.

그림 3의 폴트 트리를 보면, 최상위 이벤트의 원인으로 즉각 트립부분에서 실패, 지연 트립부분에서 실패, 즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립 요소가 제거되었을 때 즉각 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다. 의 세가지가 OR 게이트로 연결되어 있다. 여기서 트립 출력이 열린다는 것은 트립시키라고 신호가 나온것을 의미하고, 트립 출력이 닫힌다는 것은 트립이 아닌 상태를 나타낸다.

4.3 폴트 트리의 검증 및 보정

본 논문에 제시된 방법을 적용한 예로 그림 3의 노드 3을 검사하고자 한다. 첫 번째로 노드 3을 나타내는 공식을 만들어야 한다. 그리고 나서 그것을 모델 체커로

검증하며, 검증 결과에 대해 분석하게 된다.

4.3.1 첫 번째 케이스 : 노드 3

4.3.1.1 공식 만들기와 속성의 검증

"즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립요소가 제거되었을 때, 즉각 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다."

• "즉각 트립이 일어나다." - 모델에서 보면(그림 2와 8 참조) f_PDLSnrI 값이 k_SnrTrip이고, f_PDLCond 값이 k_CondIn일 때 즉각 트립이 일어나므로 이 부분은 다음과 같은 공식으로 표현될 수 있다.

→ f_PDLSnrI == k_SnrTrip and f_PDLCond == k_CondIn

• "지연 트립이 발생되고 나서 지연 트립 요소가 제거되다." - 그림 4를 보자. 지연 트립이 발생되고, [0.9, 1.1]사이의 시간이 흐른 뒤에 센서값들에 대해 판단을 해서 모든 압력 신호들(네개의 P들)이 지연 트립 임계값보다 크고, $AVEC$ 파워가 전체의 70% FP 아래로 떨어지면 즉 압력과 파워값들이 정상으로 돌아오면 지연 트립 출력이 닫힌다. 지연 트립 출력이 열렸다가 닫힌다는 것은 모델에서 PDLDly프로세스가 Wating상태에서 Normal상태로 이동하면서 f_PDLTrip값이 k_Trip에서 k_NotTrip으로 바뀌는 것으로 표현된다. 그러나, f_PDLTrip == k_NotTrip and f_PDLDly == k_InDlyNorm 인 조건은 지연 트립이 아예 발생하지 않은 처음 상태를 나타내기도 하기 때문에, 지연 트립이 일어났다가 정상으로 돌아온 상태를 표현하기엔 불충분한 식이다. 일단 지연 트립이 발생되면 클럭변수에 의해 시간의 흐름이 체크된다.

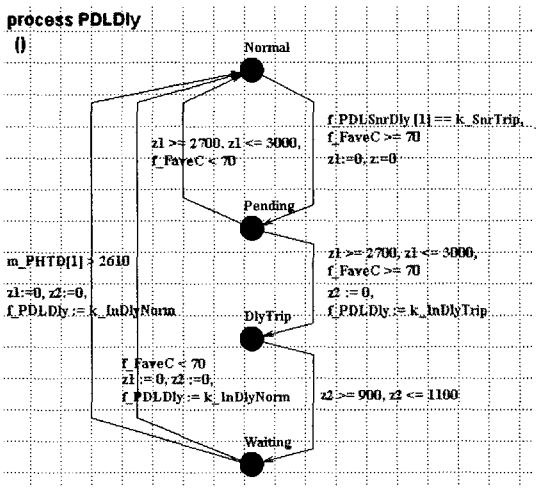


그림 4 PDLDly 프로세스

이 식이 처음 상태의 조건과 구별되기 위해서는 클럭 변수 에 대한 z 가 0보다 크다는 조건이 추가되면 된다. 따라서, 노드 3에서 뒷부분의 내용에 해당하는 공식은 다음과 같이 된다.

→ f_PDLDly == k_InDlyNorm and z > 0 and f_PDLTrip == k_NotTrip

따라서, 노드 3의 발생을 나타내는 공식 p_3는 다음과 같다.

p_3 : f_PDLDly == k_InDlyNorm and z>0
and f_PDLSnrI == k_SnrTrip
and f_PDLCond == k_CondIn
and f_PDLTrip == k_NotTrip

노드 3이 나타나는 상황의 존재 가능성을 체크하는 공식은

∀□(p_3) :

∀□ (¬(f_PDLDly == k_InDlyNorm and z>0
and f_PDLSnrI == k_SnrTrip
and f_PDLCond == k_CondIn
and f_PDLTrip == k_NotTrip))

가 된다.

4.3.1.2 검증 결과의 분석

검증 결과는 Property is not satisfied 이다. 검증 결과 속성이 만족되지 않는 경우는 속성을 만족시키지 않는 반례에 대한 시뮬레이션 트레이스를 볼 수 있다. UPPAAL 시뮬레이터의 모습이 그림 5에 보여진다.

UPPAAL의 시뮬레이터는 "Enabled Transitions", "Simulation Trace" 등의 몇 가지 기능들을 제공하는데, 시뮬레이터의 맨 왼쪽 위에 있는 "Enabled Transitions" 셀은 시뮬레이션 각 단계에서 모든 프로세스의 전이들 중에서 현재 활성화된 전이들을 모두 표

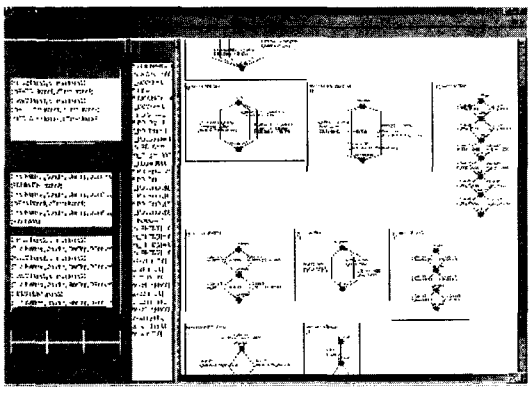


그림 5 그래픽 시뮬레이션 트레이스

시하며, 맨 왼쪽 아래의 "Simulation Trace" 셀은 각 프로세스들의 현재 상태들로 이루어진 튜플(tuple)들을 표시한다. 가운데 칼럼은 변수들의 값들을 기록하고 있고, 맨 오른쪽의 셀은 각 프로세스의 현재 상태를 빨간색으로 표시하고, 시뮬레이션 단계가 진행됨에 따라 상태의 변화를 나타내준다.

노드 3에 대한 검증 결과는 f_PDLTrip이 k_NotTrip이고, f_PDLSnrI가 k_SnrTrip, f_PDLCond가 k_CondIn, f_PDLDly이 k_InDlyNorm이며, $z > 0$인 상태가 시스템 행위 내에 존재함을 보여준다. 위의 검증에 대한 전체 시뮬레이션 트레이스는 다음과 같다. 전체 시스템은 PDLTrip, PDLDly, PDLCond, PDLSnrI1, PDLSnrDly1, mFlog, mPHTD1, Flog, FaveC, mPDLCond, pTimer로 구성된다. 설명의 편의를 위해 맨 왼쪽에 번호를 매겼다. 각 라인의 튜플은 전체 시스템의 각 프로세스의 현재 상태들로 이루어진 것이며, 짝수줄의 (프로세스이름.trans#) 튜플은 활성화된 전이를 나타낸다. 이 시뮬레이션 트레이스에 대한 자세한 분석을 통해 다음과 같은 진행 상황을 알 수 있다.

(PDLTrip, PDLDly, PDLCond, PDLSnrI1, PDLSnrDly1, mFlog, mPHTD1, Flog, FaveC, mPDLCond, pTimer)
1: (Trip, Normal, CondIn, **ISnrTrip**, **DSnrTrip**, mFlog0, mPHTD0, FlogDftHi, FaveC0, CondSwLo, pTimer0)
2: (PDLSnrDly1.trans3)
3: (Trip, Normal, CondIn, **ISnrTrip**, **DSnrNotTrip**, mFlog0, **mPHTD0**, FlogDftHi, FaveC0, CondSwLo, pTimer0)
4: (mPHTD1.trans1, pTimer.trans1)
5: (Trip, Normal, CondIn, **ISnrTrip**, DSnrNotTrip, mFlog0, **mPHTD1**, FlogDftHi, FaveC0, CondSwLo, pTimer0)
6: (PDLSnrI1.trans3)
7:(Trip, Normal, CondIn, **ISnrNotTrip**, DSnrNotTrip, mFlog0, **mPHTD1**, FlogDftHi, FaveC0, CondSwLo, pTimer0)
8: (mPHTD1.trans4, pTimer.trans2)
9:(Trip, Normal, CondIn, ISnrNotTrip, DSnrNotTrip, mFlog0, **mPHTD0**, FlogDftHi, **FaveC0**, CondSwLo, pTimer0)
10: (FaveC.trans1, pTimer.trans2)
11:(Trip, Normal, CondIn, ISnrNotTrip, DSnrNotTrip, mFlog0, mPHTD0, FlogDftHi, **FaveC1**, CondSwLo, pTimer0)
12: (FaveC.trans3, pTimer.trans2)
13:(Trip, Normal, CondIn, ISnrNotTrip, **DSnrNotTrip**, mFlog0, mPHTD0, FlogDftHi, **FaveC2**, CondSwLo, pTimer0)
14: (PDLSnrDly1.trans2)
15:(Trip, Normal, CondIn, ISnrNotTrip, **DSnrTrip**, mFlog0, mPHTD0, FlogDftHi, FaveC2, CondSwLo, pTimer0)
16: (PDLDly.trans1)
17:(Trip, Pending, CondIn, ISnrNotTrip, **DSnrTrip**, mFlog0, mPHTD0, FlogDftHi, FaveC2, CondSwLo, pTimer0)
18: (PDLDly.trans2)
19:(Trip, **DlyTrip**, CondIn, ISnrNotTrip, **DSnrTrip**, mFlog0, mPHTD0, FlogDftHi, **FaveC2**, CondSwLo, pTimer0)
20: (FaveC.trans4)
21:(Trip, **DlyTrip**, CondIn, ISnrNotTrip, **DSnrTrip**, mFlog0, mPHTD0, FlogDftHi, **FaveC1**, CondSwLo, pTimer0)
22: (PDLDly.trans3)
23:(Trip, **Waiting**, CondIn, ISnrNotTrip, DSnrTrip, mFlog0, mPHTD0, FlogDftHi, FaveC1, CondSwLo, pTimer0)
24: (PDLDly.trans6)
25:(**Trip**, **Normal**, CondIn, ISnrNotTrip, DSnrTrip, mFlog0, mPHTD0, FlogDftHi, FaveC1, CondSwLo, pTimer0)
26: (PDLTrip.trans4)
27:(**NotTrip**, **Normal**, CondIn, **ISnrNotTrip**, DSnrTrip, mFlog0, mPHTD0, FlogDftHi, FaveC1, CondSwLo, pTimer0)
28: (PDLSnrI1.trans1)
29:(**NotTrip**, Normal, CondIn, **ISnrTrip**, DSnrTrip, mFlog0, mPHTD0, FlogDftHi, FaveC1, CondSwLo, pTimer0)

• Line 1-3: 즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립요소가 제거되는 상황을 나타낸다. 시스템의 요소 중 PDLSnrDly의 상태가 (PDLSnrDly1.trans3) (line 2)에 의해서 DSnrTrip에서 DSnrNotTrip으로 바뀌었다.

• Line 3-7: 입력 값의 하나인 mPHTD 값을 읽는다 (line 4). 이 값이 즉각 트립을 결정하는 PDLSnrI의 상태를 IsnrTrip에서 (line 5) IsnrNotTrip으로 (line 7) 바꾸어 놓는다. 즉 입력이 트립 설정치보다 커서 정상적인 상태로 전이한다. 이 때에 일어나는 상태의 전이 (PDLSnrI1.trans3) 는 line6에 기술되어 있다.

• Line 7-9: mPHTD 값을 읽는다(line 8). 이 값이 mPHTD1에서 mPHTD0로 바뀌었으므로 원래 상태 즉, 즉각 트립을 발생시켜야 하는 상태로 되돌아 간다.

• Line 9-13: 또 다른 입력인 FaveC 값을 두 차례 읽는다(line 10, 12). 입력된 FaveC의 값이 FaveC0에서 FaveC1로 상승하였으며, 다시 FaveC2로 상승하였다. FaveC의 값이 FaveC2의 상태이며, mPHTD의 값도 mPHTD0이므로 지연트립이 발생할 수 있는 제한 조건이 만족되었다.

• Line 13-15: FaveC 값의 영향으로 PDLSnrDly의 상태가 지연 트립요소가 제거되었던 DSnrNotTrip에서 다시 DsnrTrip으로 바뀐다. 이 때에 일어나는 상태의 전이(PDLSnrDly1.trans2)는 line14에 기술되어 있다.

• Line 15-19: <그림 4>의 모델에서 제시하고 있는

것과 같이 지연 트립이 발생한다.

• Line 19-21: 다시 FaveC 값을 읽는다(line 20). 이 값으로 인해서 지연 트립이 정상상태로 바뀔 수 있는 상태가 된다.

• Line 21-25: <그림 4>의 모델에서 설명된 것과 같이 잠시 기다리는 시간([0.9,1.1])이 흐른 뒤의 상태를 거쳐 지연 트립이 정상상태로 바뀐다.

• Line 25-27: 이 영향으로 PDLTrip 상태가 전이(PDLTrip.trans4) (line 26)에 의해서 NotTrip으로 바뀐다. 즉시 트립 조건은 이미 활성화되어 있다.

• Line 29: 즉시 트립 조건이 활성화된 상태임에도 불구하고, PDLTrip 프로세스가 NotTrip상태에 머무른다. 이것은 안전하지 못한 상태이며, 중요한 위험이 될 수 있다.

PDLTrip이 NotTrip이라는 것은 즉시 트립과 지연 트립 둘다 아니라는 것을 나타내는데, 실제 시스템을 수행시켜 보면 Trip상태에서 즉시 또는 지연 트립 중 어느 하나만 Trip상태를 벗어나도 PDLTrip프로세스가 NotTrip으로 이동하게 되는 것이다. PDLTrip의 상태를 결정하는 요인은 즉시 트립과 지연 트립 두가지가 있는데, 이 두가지 요인의 결과는 한 곳 (PDLTrip)에 나타나기 때문에 나타나는 현상이다.

이러한 과정을 통해, 우리는 노드 3이 표현하는 즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립요소가 제거되었을 때, 즉각 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다 라는 이벤트, 즉 노드 3이 나타내는 위험 요인이 시스템 내에서 정말 존재함을 확인하였다. 이 정보는 폴트 트리 내의 이 노드가 제대로 찾아지고 구성되어졌다는 확신을 준다. 실제 존재할 수 있는 이러한 위험 요소를 제거하기 위해 시스템에 적절한 조치가 취해져야 할 것이다.한편, 시뮬레이션 트레이스에 대한 분석을 통해 폴트 트리의 보정을 위해 작성한 시스템 모델의 정확성을 검증해 볼 수 있다는 효과도 얻을 수 있다. UPPAAL 상의 시스템 모델이 시스템을 정확하게 반영하고 있는가를 시뮬레이션을 통해 검증함으로써 좀 더 정확한 폴트 트리 보정을 할 수 있을 것이다. 본 연구에서도 UPPAAL이 제공한 반례에 해당하는 시뮬레이션 트레이스 중 8번 전이에 의해 PDLSnrI의 상태가 IsnrNotTrip에서 IsnrTrip으로 즉각 바뀌지 않는 점을 발견하였다. 이는 UPPAAL 상의 모델이 시스템을 정확하게 반영하고 있지 못한 경우 또는 UPPAAL의 결함으로 생각할 수 있는 부분이다. 이 문제는 본 논문의 방향에 큰 영향을 미치지 않으므로 여기서는 더 이상 언급하지 않도록 한다.

4.3.1.3 폴트 트리의 보정

위의 분석과 함께 우리는 몇 가지 점들을 더 살펴봄으로 폴트 트리를 더 정확하게 수정할 수 있다. 폴트 트리에 대한 검증결과를 분석하는 과정에서 얻을 수 있는 정보들을 이용해 수작업으로 폴트 트리를 보정한다. 수정된 폴트 트리가 정확한가 하는 점은 수정된 폴트 트리에 대해서 다시 모델 체커를 이용한 검증을 적용함으로 증명될 수 있다.

1. 노드의 이벤트 내용이 부정확하게 기술되었다.

UPPAAL이 제공해 준 반례는 노드 3이 기술하고 있는 것처럼 Trip에서 NotTrip으로의 전이가 지연 트립 조건이 비활성화 되어서가 아니라 즉시 트립 조건이 비활성화되었기 때문에 이루어진 경우를 보여준다. 앞에서 노드 3 이벤트가 일어난다는 것은 확인했고, 여기서 알 수 있는 것은 노드 3이 기술하면서 예측했던 상황이 아니라 다른 상황에 의해 그러한 결과가 나왔다는 것이다. 노드 3을 나타내는 p_3이 예측했던 상황말고도 또 다른 상황을 나타내고 있다는 것을 의미한다. 자세한 분석을 통해서, 우리는 노드 3의 이벤트 기술 내용이 좀 더 일반적으로 수정되어야 한다는 것을 알 수 있다. 지연 트립과 즉시 트립이 둘 다 일어난 다음에, 지연 트립이 비활성화되서 전체 트립이 닫히는 경우도 있지만, 즉시 트립이 비활성화되어서 전체 트립이 닫히는 경우도 있다. 이들을 모두 포함하기 위해서

"노드의 내용은 즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립요소가 제거되었을 때, 즉각 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다."

에서

"즉각 트립과 지연 트립이 동시에 일어난 후, 둘 중 어느 한 트립요소가 제거되었을 때, 다른 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다."

로 수정되어야 한다.

2. 기본 이벤트에 대한 공식이 너무 복잡하다

이 경우 우리는 폴트 트리에 대한 또 다른 보정에 대해 생각해 볼 수 있다. 기본 이벤트에 대한 공식이 너무 복잡한 경우는 그 기본 이벤트가 실제 여러 개의 더 기본적인 이벤트들의 조합을 포함하고 있을 수 있으며 그 경우에는 폴트 트리를 좀 더 세분화할 필요가 있다. 1.에서 수정된 노드 3 이벤트는 다음과 같이 더 기본적이면서 OR 게이트로 연결되는 두개의 기본 이벤트로 나누어질 수 있다.

"즉각 트립과 지연 트립이 동시에 일어난 후, 지연 트립요소가 제거되었을 때, 즉각 트립이 아직 유효함에도

불구하고 전체적으로는 트립 출력이 닫힌다."

(노드 3.1)

OR

"즉각 트립과 지연 트립이 동시에 일어난 후, 즉각 트립요소가 제거되었을 때, 지연 트립이 아직 유효함에도 불구하고 전체적으로는 트립 출력이 닫힌다."

(노드 3.2)

노드 3.2는 다음과 같은 공식으로 표현된다.

A[] (f_PDLDly == k_InDlyNorm and z >0 and f_PDLTrip == k_NotTrip and f_PDLSnrI == k_SnrTrip and f_PDLCond == k_CondIn)

보는 것과 같이, 이 속성은 이벤트를 표현하기 위해서 많은 'and'들을 포함한다. 이 노드는 여러개의 자식 노드들로 분리될 필요가 있을지 모른다. 이것은 폴트 트리 구성이 좀 더 세분화 되어야 할 필요를 의미한다. 실제로 위의 공식은 and로 이어진 다음의 두 부공식으로 나뉘어질 수 있다.

f_PDLSnrI == k_SnrTrip and f_PDLCond == k_CondIn

f_PDLTrip == k_NotTrip and f_PDLDly == k_InDlyNorm and z > 0

각 공식은 노드 3.2의 자식 노드가 된다.

이 공식들이 and' 연산자에 의해 합성되므로, 자식 노드들은 폴트 트리에서 AND 게이트에 의해 연결되게 된다. 이런 식으로 우리가 잎노드로부터 공식을 만들었을 때, 그 공식의 길이가 너무 길거나 하면, 그 잎노드가 좀 더 기본적인 이벤트들로 세분화될 필요가 있지 않은지 살펴보아야 한다. 이러한 방법은 모델 체킹에 의해 폴트 트리를 보정할 수 있는 하나의 방법 -폴트 트리가 더 세분화 되도록 돕는 방법-이다. 그러나, 물론 많은 연결 연산자를 가진 모든 긴 공식들이 나뉘어질 수 있는 것은 아니다. 기본 이벤트 자체가 복잡해서 긴 공식을 필요로 하는 경우가 있기 때문이다.

노드 3에 대한 보정 결과가 그림 6 에 보여진다.

4.3.2 두 번째 케이스 : 노드 12

4.3.2.1 공식 만들기와 속성의 검증

노드 12 : "정확한 딜레이가 [2.7,3.0]초 이내에 있지 않음."

위험상황에서 지연 트립 신호를 내보내는 PDLDly 프로세스는 압력 센서와 파워 값을 점검해서 그들 값이 비정상적인 경우, 바로 정지 신호 출력을 내보내지 않고 일단 [2.7,3.0]초간 정상 수행을 지속한다. 노드 12는 딜레이가 [2.7,3.0]초 범위가 아닐 때 트립 신호를 내는데 실패할 수 있음을 나타내고 있다. 노드 12의 내용을 시

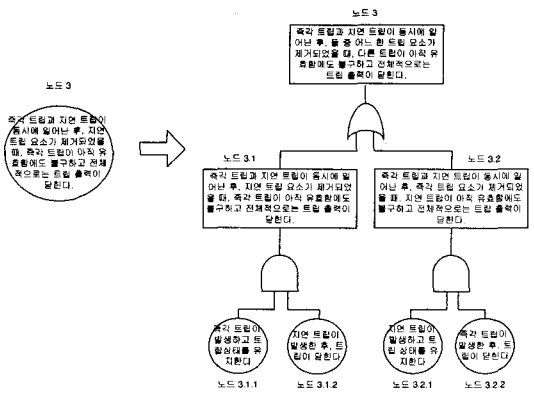


그림 6 폴트 트리의 노드 3에 대한 보정 결과

제 논리공식으로 표현하면 다음과 같다. UPPAAL에서 정수만 표현할 수 있기 때문에 2.7은 27로 3.0은 30으로 표현했으며, z는 클럭 변수이다.

→ (z < 27 or z > 30) and f_PDLTrip == k_NotTrip

딜레이가 [2.7,3.0] 범위를 벗어났기 때문에 f_PDLTrip이 Trip이 될 수 없고, NotTrip임을 나타낸다. 노드 12의 경우가 존재할 수 있는지 다음의 공식을 통해 검증한다.

$\forall\Box(p_{12})$:

$\forall\Box$(((z < 27 or z > 30) and f_PDLTrip ==k_NotTrip))

4.3.2.2 검증 결과의 분석

검증 결과는 **"Property is not satisfied"** 이다. 이 경우 UPPAAL이 보여주는 시뮬레이션 트레이스는 클럭 변수 z는 0이고(시간이 흐르지 않은 상태), PDLSnrI가 ISnrTrip에서 IsnrNotTrip으로 이동했기 때문에 f_PDLTrip이 k_NotTrip으로 되는 경우를 보여준다. 즉, 클럭 변수의 변화, 시간 딜레이가 시스템에 끼치는 영향을 보기 위한 검증을 하였는데, 시간은 하나도 흐르지 않은 상태에서 다른 요인(센서값이 정상으로 돌아옴)에 의해 f_PDLTrip이 k_NotTrip이 된 경우를 보여준 것이다. 검증 결과가 의미 없는 트레이스는 보여주는 경우는 보통 검증 공식이 적절하지 않기 때문이다. 이 경우 분석을 해 보면, 지연 트립에서 압력과 파워값이 비정상적인 경우에 [2.7,3.0]의 딜레이가 의미가 있는데, 그 상황을 제대로 기술하지 않았기 때문에 즉 노드 12가 상황에 대한 가정을 생략했기 때문에 공식이 불명확해진 것임을 알 수 있다. 생략된 가정들을 포함해서 노드 12는 다음과 같이 수정되어야 한다.

$\forall\Box(p_{12})$:

∀□ ((f_PDLSnrDly == k_SnrTrip and f_FaveC >= 70 and (z<27 or z>30) and f_PDLTrip==k_NotTrip))

수정된 공식에 대해 검증을 수행한 결과는 Property is satisfied 이다. 즉, 모델에서 딜레이가 [2.7,3.0]범위를 벗어나서 PDLTrip 출력이 오류가 나는 경우는 없다는 것을 보여준다.

4.3.2.3 폴트 트리의 보정

위의 분석을 통해 폴트 트리 노드 12에 대한 처음 공식이 불명확하다는 것을 확인했다. 검증 공식은 노드의 자연어 기술로부터 나오는 것이기 때문에 공식이 불명확하다는 것은 노드의 기술이 불명확하다는 것을 말해준다. 위의 경우는 노드 12의 자연어 기술이 가정을 정확히 명시하고 있지 않기 때문에 불명확함이 발생된 경우이므로, 노드 12의 내용은 다음과 같이 가정을 포함하도록 수정되어야 한다.

정확한 딜레이가 [2.7,3.0]초 이내에 있지 않음 (이전의 노드 12)

→ 어떤 P가 지연 트립 임계값 아래에 있고, AVEC가 70%FP를 초과했을 때, 정확한 딜레이가 [2.7,3.0]초 이내에 있지 않음 (새로운 노드 12)

5. 결론 및 향후 연구

본 논문에서는 모델 체킹 방법이 폴트 트리의 정당성을 검증하고 보정하는데 유용하게 사용될 수 있음을 제안한다. 먼저 폴트 트리를 검증하기 위한 두가지 공식 형태를 제안하였다. 하나는 노드 이벤트의 존재 가능성을 검증하기 위한 형태이고, 다른 하나는 게이트에 의한 노드들의 결합의 정확성을 검증하기 위한 형태이다. 공식을 만든 다음에는 이 공식을 실시간 모델 체커 UPPAAL을 이용하여 검증한다. 이 속성들의 검증 결과는 우리에게 폴트 트리의 정확성에 대한 확신할 수 있는 정보를 주기도 하고, 속성이 만족 되지 않았을 경우, 모델 체커에서 제공해 주는 반례들을 통해 폴트 트리의 잘못된 부분을 보정할 수 있는 정보를 주기도 한다. 이 정보들을 가지고 우리는 폴트 트리를 좀더 정확한 형태로 고칠 수 있다.

어떤 최상위 위험 이벤트에 대해 완전한 하나의 폴트 트리가 존재하는 것이 아니므로, 폴트 트리의 모든 부분을 정확하게 고친다는 것은 실제 거의 불가능한 일이다. 그러나, 이 논문에서는 오류 가능성을 포함하고 있는 폴트 트리에 대해 이렇다할 만한 검증이나 보정 방법이 없는 상황에서 모델 체킹이라는 자동화된 방법을 사용해서 검증 및 보정에 대한 가능성을 제시하였다.

모델 체킹 방법은 디자인 단계에서 시스템이 어떤 속성들을 만족시키는지를 검증하여 시스템 디자인을 정확하게 만드는 목적으로 주로 사용되어 왔는데, 본 논문에서는 모델 체킹 방법이 폴트 트리 분석을 보완하는데에도 사용될 수 있음을 보였다. 월성의 PDL 정지 신호 관련 부분에 대한 사례 연구를 통해서 이 방법이 유용하게 사용될 수 있음을 보였다.

본 논문에서 우리는 여러가지 상황에 대해 폴트 트리를 보정할 수 있는 몇가지 조정 방법을 제시하였다. 그러나 이것들로는 충분하지 않으며, 향후 연구를 통해서 더 많은 사례에 대한 보정 방법이 제시될 필요가 있다. 또한, 본 논문에서는 노드의 자연어 기술을 시제 논리 공식으로 바꾸는 과정은 정확하다고 가정하였었는데, 이 과정은 분석가의 능력에 의존적으로 달라질 수도 있으므로, 좀 더 체계적인 변환 방법이 제시될 필요가 있다.

참 고 문 헌

[1] W. E. Vesely. "Fault Tree Handbook," Technical Report NUREG-0492 [0942?], US Nuclear Regulatory Commission, 1981.

[2] E. M. Clarke, Jr., O. Grumberg and D. A. Peled, "Model Checking," MIT Press, 1999.

[3] R. Alur, C. Courcoubetis, and D. L. Dill. "Model-Checking in Dense Real-time," *Information and Computation*, 104(1), 1993. preliminary version appeared in Proc. 5th LICS, 1990.

[4] A. Pnueli. "The temporal logic of programs," In *In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46-77, 1977.

[5] S. Owicki, and L. Lamport. "Proving liveness properties of concurrent programs," *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[6] E. A. Emerson and E. M. Clarke. "Using branching-time temporal logic to synthesize synchronization skeletons," *Science of Computer Programming*, 2, 1982.

[7] Z. Manna, and A. Pnueli. "The anchored version of the temporal framework," In *In Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science 354*. Springer-Verlag, 1989.

[8] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. "Symbolic Model Checking for Real-Time Systems," *Information and Computation*. 111(2), 1994.

[9] K. G. Larsen, P. Petterson and W. Yi. "Model-Checking for Real-Time Systems," In *Invited paper. In Proceedings of 10th International Funderamentals of Computing Theory,* Dresden, Germany, August 1995. LNCS 965, pages 62-88, Horst Reichel(Ed.).

[10] S. Yovine. "Model-Checking Timed Automata," In G. Rozenberg and F. Vaandrager, editors, In *In Embedded Systems, Lecture Notes in Computer Science,* 1998. invited paper.

[11] T. A. Henzinger, O. Kupferman, and M. Y. Vardi. "A space-efficient on-the-fly algorithm for real-time model checking," In *Proceedings of the Seventh International Conference on Concurreny Theory (CONCUR 1996),* 1996. LNCS 1119, Springer-Verlag, 1996, pp. 514-529.

[12] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson and W. Yi. "UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems," In *In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems,* New Brunswick, New Jersey, October 1995.

[13] K. Havelund, A. Skou, K. G. Larsen and K. Lund. "Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL," In *In Proceedings of the 18th IEEE Real-Time Systems Symposium,* San Francisco, California, USA, December 1997.

[14] H. E. Jensen, K. G. Larsen and A. Skou. "Modelling and Analysis of a Collision Avoidance Protocol using SPIN and UPPAAL," In *In Proceedings of the 2nd SPIN Workshop,* Rutgers University, New Jersey, USA, August 1996.

[15] L. Acetoa, A. Bergueno and K. G. Larsen. "Model Checking via Reachability Testing for Timed Automata," In *In Proceedings of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Gulbenkian Foundation, Lisbon , Portugal 1998. LNCS 1384, pages 263-280, Bernhard Steffen (Ed.)

[16] M. Lindahl, P. Pettersson and W. Yi. "Formal Design and Analysis of a Gear-Box Controller," In *Proc. of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* number 1384 in Lecture Notes in Computer Science, pages 281-297. Springer-Verlag, March 1998.

[17] H. Lönn and P. Pettersson. "Formal Verification of a TDMA Protocol Startup Mechanism," In *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems,* pages 235-242. December 1997.

[18] K. G. Larsen, P. Pettersson and W. Yi. "Diagnostic Model-Checking for Real-Time Systems," In *Proc. of Workshop on Verification and Control of Hybrid Systems III,* number 1066 in LNCS, pages 575-586. Springer-Verlag, October 1995.

[19] Kirsten M. Hansen, Anders P. Ravn, and Victoria Stavridou. "From Safety Analysis to Software Requirements," *IEEE Transactions on Software Engineering,* 24(7):573-584, July 1998.

[20] Program Functional Specification, SDS2 Programmable Digital Comparators, Wolsong NPP 2,3,4," Technical Report 86-68300-PFS-000 Rev.4, AECL CANDU, May 1994.

지 은 경

1999년 KAIST 전산학과 학사. 2001년 KAIST 전산학과 석사. 2001년 ~ 2002년 몽골 울란바타르 대학 조교수. 관심분야는 소프트웨어공학, 정형기법

차 성 덕

정보과학회논문지 : 소프트웨어 및 응용 제 29 권 제 5 호 참조

손 한 성

1993년 서울대학교 원자핵공학과 학사. 2000년 KAIST 원자력공학과 박사. 관심분야는 소프트웨어 안전성, 정형기법, 원자력계측제어

유 준 범

1999년 홍익대학교 컴퓨터공학과 학사. 2001년 KAIST 전자전산학과 전산학전공 석사. 2001년 ~ 현재 KAIST 전자전산학과 전산학전공 박사과정. 관심분야는 소프트웨어공학, 안전성분석, 정형기법

구 서 룡

1998년 한국과학기술원 원자력공학과 학사. 2000년 한국과학기술원 원자력공학과 석사. 2000년 ~ 현재 한국과학기술원 원자력공학과 박사과정. 관심분야는 소프트웨어 확인 및 검증 기법, 정형기법

성 풍 현

1977년 서울대학교 공과대학 원자핵공학 학사. 1984년 MIT 원자핵공학 석사. 1987년 MIT 원자핵공학 박사. 1991년 ~ 현재 한국과학기술원 원자력 및 양자공학과 교수. 관심분야는 소프트웨어 기반 safety critical digital system 신뢰도 연구, 원자력발전소 운전원 지원 시스템 연구, System Integration