NuSCR 의 검증을 위한 Quick Checker 개선
Reformation of Quick Checker for verifying NuSCR

조재연, 윤상현, 유준범

건국대학교 컴퓨터공학부
서울시 광진구 화양동 1 번지
{tm77, pctkdgus, jbyoo}@konkuk.ac.kr

요약: 소프트웨어의 안전성을 향상시키기 위한 방법 중의 하나는 요구사항 명세를 검증하는 것이다. NuSRS[1]는 요구사항 명세 언어인 NuSCR[2]를 지원해주는 도구이며, NuSRS 와 NuSCR 은 KNICS RPS(Reactor Protection System)[3]인 APR-1400 원자로 보호시스템의 소프트웨어를 개발하는데 사용되었다. 원자로 보호 시스템은 안전 필수 시스템으로서, 요구사항 분석 단계부터 안전성 분석이나 정형 검증의 과정을 거쳐야 한다. 이런 요구에 맞추어 NuSRS 는 NuSCR 로 명세 된 요구사항을 NuSCRtoSMV 를 통하여 모델체커인 SMV 의 입력 언어[4]로 자동으로 변환하여 주고 전문가가 안전성 분석을 하는데 도움이 되는 도구인 NuFTA[5]를 포함한다. 그리고 이 두 과정 전에 Quick Check 기능을 사용하여 명세의 문법적인 오류를 검사하도록 하였다. 그러나 기존 Quick Check 기능은 완전성과 무모순성 관점에서 NuSCR 명세를 검증하지 못한 부분이 있었다. 본 연구에서는 완전성과 무모순성의 관점에서 NuSCR 명세를 대상으로 고려해야 할 점을 정의하였다. 그리고 Quick Check 를 CSP(Constraint Satisfaction Problem)를 해결할 수 있는 라이브러리를 사용하여 보완하였으며 적은 노력과 시간으로 NuSCR 명세의 완전성과 무모순성을 보장할 수 있는 도구로 개선하였다.*

핵심어: NuSRS, NuSCR, Quick Checker, Completeness, Consistency, Constraint Satisfaction Problem

1. 서론

소프트웨어의 개발 단계 중 요구사항의 명세는 추후 디자인, 구현, 테스트 단계에 큰 영향을 미친다. 따라서 명확한 요구사항의 명세는 이후 개발 프로세스의 비용을 줄일 수 있다. 특히 안전 필수 소프트웨어(예를 들면, 원자력 발전소나 항공기)는 요구사항 분석 단계에서부터 정형 검증이나 안전성 분석 등의 과정을 통한 안전 논증이 KINS(Korea Institute of Nuclear Safety)와 같은 규제기관에 의해 요구된다. 정형명세를 검증하는 방법으로는 인스펙션과 정형 검증이 있다. 인스펙션은 소프트웨어 요구, 설계, 원시코드 등을 저작자 이외의 다른 전문가가 검사하여 오류를 찾는 검토방법이며, 정형 검증은 수학적 추론을 이용하여 설계의도나 사양이 구현될 수 있는지를 검증하는 체계적인 프로세스로서 주로 정형명세를 대상으로 하여 모델 체킹을 사용한다. 모델 체킹[7]은 명세 자체의 문법적인 오류를 고려하지 않기 때문에 인스펙션을 거친 명세를 대상으로 모델체킹을 해야 한다.

NuSCR[2]은 APR-1400 의 원자력 보호 시스템의 컨트롤러 소프트웨어의 요구사항을 명세하기 위한 요구사항 명세 언어이며 SCR(Software Cost Reduction) [6]을 원자력 분야에 맞게 수정한 것이다. NuSCR 과 이를 지원하는 도구인 NuSRS 는 APR-1400 원자로 보호시스템의 소프트웨어를 개발하는데 사용되었다. NuSRS 에는 인스펙션을 지원하는 Quick Check 기능이 있었다. 그러나 Quick Check 는 NuSCR 명세가 문법을 지키는지 검사하였으나 완전성과 무모순성을 지키는지 검사하기에는 부족한 점이 있었다. 본 논문에서는 이를 해결하기 위한 도구인 Quick Checker 를 소개한다. Quick Checker 는 NuSCR 이 요구하는 문법뿐만 아니라 명세의 완전성(Completeness)과 무모순성(Consistency)을 만족하는지도 검사한다. 2 장에서는 Quick Checker 를 개발하기 위한 배경지식으로 NuSCR, 완전성과 무모순성, 그리고 CSP(Constraint Satisfaction Problem)에 관하여 설명한다. 3 장에서는 2 장에서 이야기한 배경지식들을 토대로 한 Quick Checker 구조를 제시하고, 4 장에서는 APR-1400 원자로 보호 시스템 컨트롤러의 소프트웨어 프로토 타입의 명세를 대상으로 구현된 Quick Checker 의 성능을 확인하며 5 장에서 결론을 맺는다.

* 본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT 연구센터 지원산업의 연구결과로 수행되었음(NIPA-2010-c1090-0903-0004, NIPA-2010-C1080-1031-0003) 이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2010-0002566)

2. 배경 및 관련 연구

2.1 NuSCR

NuSCR 은 디지털 원자로 보호시스템의 소프트웨어 요구사항을 명세하기 위해 SCR 을 수정한 언어로써 FOD, FSM, TTS, SDT 등으로 이루어져 있다. 그림 1 은 KNICS RPS(Reactor Protection System) BP (Bistable Process)의 명세 중 *g_VAR_OVER_PWR* 를 명세하기 위한 FOD 를 보여주고 있다[8]. 'g'는 그룹 노드를 나타내는 접두사로서 FOD 의 이름 앞에 붙는다. FOD 는 history variable node, timed history variable node, function variable node 로 구성되는데 각각 '*h*', '*th*', '*f*'의 접두사를 가지며 FSM (Finite State Machine), TTS (Time Transition System), SDT (Structural Decision Table)로 정의된다. FOD 는 시스템의 기능들을 객체화 시켜서 다이어그램 형태로 만든 것으로, 노드와 전이들로 이루어져 있다. 노드들의 역할은 입력, 출력 또는 처리하는 객체나 기능을 나타내며 기능 하나당 하나의 노드가 할당된다. 입력과 출력은 하나의 변수로 취급되며, 처리기능은 FOD, SDT, FSM, TTS 들 중의 하나가 된다. 전이는 노드들 간의 데이터 이동경로를 포함하고 기능의 처리순서를 나타낸다. 노드는 도형으로, 전이는 벡터로 각각 표현된다. 노드와 노드가 전이로 이어졌다면, 노드간에 데이터 이동이 일어난다. 전이를 주는 노드와 전이를 받는 노드가 있는데, 두 노드는 의존성이 있어 전이를 주는 노드는 받는 노드보다 먼저 처리가 되어 처리된 값을 전달한다. 전이를 받는 노드는 전이를 통해서 값을 받는다.

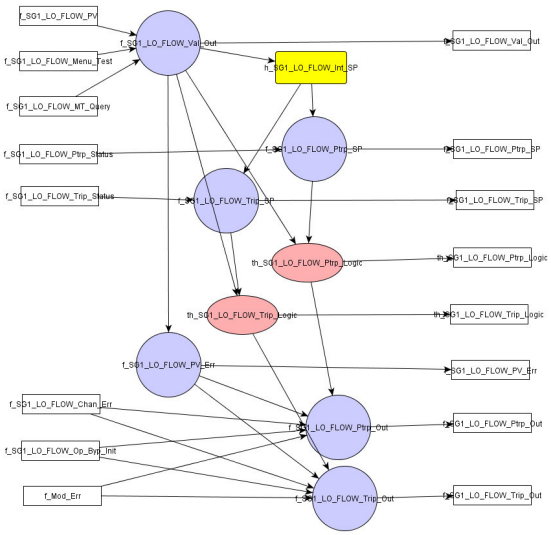

그림 1 g_VAR_OVER_PWR 에 대한 FOD

FSM 으로 정의되는 history variable node 는 시스템의 상태를 표현한다. FSM 은 상태와 전이로 이루어져 있다. NuSCR 은 대상이 되는 컨트롤러가 전원이 들어오면 전원이 나갈 때까지 계속 작동하는 소프트웨어를 명세하기 때문에 초기상태가 있지만 종료상태는 없다. FSM 의 전이는 전이의 시작상태와 대상상태 그리고 전이가 이루어지기 위한 조건과 전이가 이루어질 때의 출력 값에 대한 할당부분을 가지고 있다. 시스템의 처리가 시작되면 초기상태로부터 시작해서 연결되어 있는 모든 전이들의 조건을 검사하여 조건을 만족하는 전이가 있으면 대상상태로 넘어가며 시스템의 한 주기가 지날 때마다 단 하나의 전이만을 실행한다. 현재 상태에서 다른 상태로 연결되는 전이들의 조건들을 하나도 만족하지 못한다면 현재 상태에 머문다.

Timed history variable node 는 TTS 로 정의된다. TTS 의 기본적인 형태는 FSM 과 같지만 시간조건이 있는 전이가 포함될 수 있다는 점이 다르다. 시간 조건은 시스템의 주기시간이 몇 번이나 지나야 전이가 이루어 질 수 있는지 조건을 주는데 사용된다. 그림 2 에서 *Waiting* 에서 *Trip* 상태로 이어지는 전이의 조건은 시간조건인 *k_VAR_OVER_PWR_Trip_Dly* 만큼의 단위 시간 동안 전이의 조건을 계속해서 만족해야만 *Trip* 상태로 넘어갈 수 있음을 의미한다.

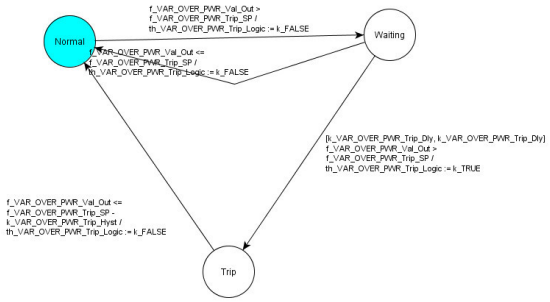

그림 2 th_SG1_LO_FLOW_Trip_Logic 에 대한 TTS

Function variable node 는 그림 3 과 같이 SDT 로 정의되며 시스템의 수학적 기능을 명세 하는 데 사용된다. SDT 는 조건 부분과 행동부분이 있는 테이블 형태로 되어 있다. SDT 의 조건부분은 조건 식을 기술하는 열과, 해당 조건의 진리(True – 'T', False – 'F', Don't care – '-')를 기술하는 열로, 행동부분은 행동 식과 조건부분의 조건들의 조합에 따른 행동이 정의되는 열로 구성되어 있다.

Conditions	1	2	3
f_SG1_LO_FLOW_Val_Out > k_SG1_LO_FLOW_PV_Max	T	-	F
f_VAR_OVER_PWR_Val_Out < k_VAR_OVER_PWR_PV_Min	-	T	F
Action	**1**	**2**	**3**
f_SG1_LO_FLOW_PV_Err := true	O	O	
f_SG1_LO_FLOW_PV_Err := false			O

그림 3 f_SG1_LO_FLOW_PV_Err 에 대한 SDT

2.2 완전성과 무모순성

완전성(Completeness)은 어떤 논리식이 해당 공리 체계에서 의미론적으로 참이면, 증명 가능하다는 것을 의미한다. 무모순성(Consistency)은 해당 공리로부터 논리식과 논리식의 부정이 동시에 증명 가능하지 않은 성질이다. NuSCR 의 모델의 조건들은 특정 상태에서 일어날 수 있는 행동을 결정한다. 모델을 검증하기 위해서 조건들의 영역을 하나의 공리체계로 가정하고 완전성과 무모순성의 관점을 이용한다. NuSCR 의 모델 M 의 어떤 하나의 상태 S 에서 진리치를 판별할 수 있는 조건의 개수를 n 이라 하고, $p_k(1 \le k \le n)$를 각 조건의 진리치라고 하자. 조건의 진리치의 집합을 Σ 라고 하면 $\Sigma = \{p_1, p_2, \ , p_n\}$ 이다. p 의 값은 true 또는 false 이며, Σ의 원소들의 개수는 n 개이므로, $n(^{\forall}\Sigma) = 2^n$이다. m 은 M 에서 명세 되어 있는 Σ의 개수, $\Sigma_i(0 < i \le m)$를 M 에 명세 되어있는 Σ 이라고 하고, $\Sigma_r(1 \le r \le 2^n)$를 실행 시 조건들의 진리치의 집합인 Σ으로 가정하자. 그리고 $P_i(1 \le i \le m)$는 Σ_i와 Σ_r를 비교한 진리치라고 가정하자. 모든 실행의 경우인 $^{\forall}\Sigma_r$에 대해서, $\bigvee_{i=1}^{i=m} P_i = \text{true}$를 만족한다면 어떤 이라도 NuSCR 의 요구사항 내에서 증명할 수 있으므로, 완전성을 만족한다. $\forall_{i \ne j}(P_i \wedge P_j) = \text{false}$ 라면 어떤 조건들의 조합도 중복되지 않고 한 조합에서의 서로 다른 행동을 일으키지 않으므로 무모순성을 만족한다[9].

2.3 Constraint Satisfaction Problem

CSP(Constraint Satisfaction Problem)[10]는 몇 가지의 제약사항이나 제한 영역을 만족해야 되는 상태들의 집합을 찾는 수학적 문제들이다. CSP 는 다음과 같이 구성된다.

정의 1: CSP 객체 O = <V, A, C> 는 다음과 같다.
- V 는 변수들의 집합이다.
- A 는 값들의 영역이다.
- 아래의 조건에 한해서 C 는 제약사항들의 집합, $\{C_1, \ , C_q\}$ 이다.
  다음의 경우, 각 제약사항 $C_i \in C$는 $\langle s_i, \rho_i \rangle$ 는 쌍이다.
  - s_i는 변수들의 크기가 m_i인 집합이며, 제약사항의 범위라고 한다.
  - ρ_i는 A 의 m_i진 관계이며 제약사항의 관계라고도 한다.

NuSCR 의 모델에서 어떤 한 상태에서 전이를 결정하거나 행동을 결정할 때, 두 가지 이상의 조건(또는 조건의 조합)이 동시에 만족되는 경우가 생긴다면 무모순성을 위반하게 된다. 따라서 해당 경우가 생기지 않도록 하기 위해서 CSP 해결 기법을 NuSCR 의 각 모델의 조건에 적용하여 해결하였다. 조건들은 CSP 의 C 즉, 제약사항에 해당하며 진리치를 포함한다. 해당 식의 변수의 집합은 CSP 의 V 에 해당한다. 해당 조건의 변수의 범위들을 토대로 유한 영역을 정한다. 이는 A 에 해당한다. 그림 4 예제는 NuSCR 의 조건들에 해당된다. 이 예제에서는 n_variable_1, n_variable_2 는 변수, ①, ②, ③번이 C 인 제약사항, ④, ⑤번이 A 인 값들의 영역에 해당한다. 제약 사항인 ①, ②, ③을 동시에 충족시킬 수 없으므로 이는 무모순성을 위반하게 된다. 본 연구에서는 CSP 를 사용할 수 있는 라이브러리를 사용하여 FSM, TTS, SDT 등에 적용하였다.

```
①  n_variable_1 = 1;
②  n_variable_2 < 1;
③  n_variable_1 = n_variable_2;
④  -6 < n_variable_1 < 5;
⑤  -3 < n_variable_2 < 8;
```

그림 4 NuSCR 의 조건 예

3. Quick Checker

3.1 명세 검사

Quick Checker 사용자가 작성한 NuSCR 명세가 NuSCR 문법과 완전성 및 무모순성을 만족하는지에 대해 검사한다. 본 연구에서는 기존 Quick Check 를 완전성과 무모순성의 조건을 충족시킬 수 있도록 각 노드 별로 고려 해야 할 점들을 정의 하고 이를 지원하기 위해 기존의 Quick Check 를 보완하여 Quick Checker 를 개발하였다.

3.1.1 FOD

FOD 는 입력 노드, 출력 노드 그리고 나머지 제어 노드(FOD, TTS, FSM, SDT 를 의미함)로 구성되어 있다. 각 노드들의 이름은 출력하는 값의 이름과 같다. 제어 노드는 단 하나의 출력 노드와 연결될 수 있다. 2 개 이상의 제어 노드로부터 하나의 출력 노드로 전이가 된다면, 어떤 값이 출력되는지 결정할 수 없다. 이 때, 두 제어 노드가 서로 모순되므로 무모순성을 만족시키지 못한다. 제어 노드들은 입력 노드의 값을 받고 출력 노드로 값을 출력한다. 그림 5 에서 입력 노드인 input_1 으로부터 제어 노드인 output_1, ouput_2 로 값을 받고, 제어 노드인 output_1, output_2 는 각각 출력 노드인 output_1 과 output_2 로 연결된다. 입력 노드로부터 나오는 전이는 하나 이상 있어야 하며, 제어 노드로만 연결되어 있어야 한다.

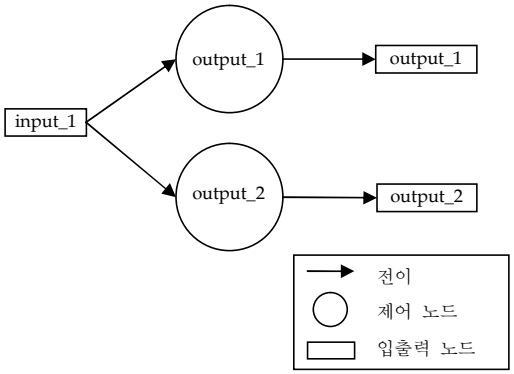

그림 5 제어노드와 입출력 노드가 올바르게 연결된 FOD

그림 6 에서 입력 노드인 input_1 에서 전이가 없으므로 입력 값이 어떻게 처리되는지 요구사항 레벨에서 알 수 없다. 따라서 입력 값은 무효하게 되고 완전성을 만족시키지 못한다.

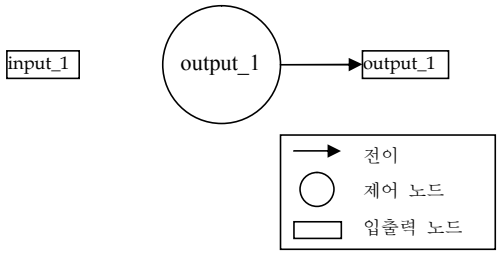

그림 6 입력노드의 전이가 없는 FOD

입출력 노드는 자신이나 다른 입출력 노드로 연결되지 않아야 한다. 만약 그림 7 처럼 입력 노드와 출력 노드가 전이로 연결이 되어 있다면, 이 두 노드간의 관계는 명확하지 않다. 따라서 완전성을 만족시키지 못한다.

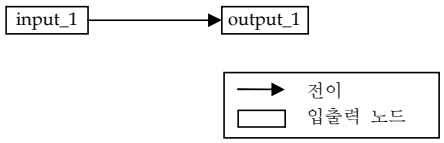

그림 7 입출력 노드끼리만 연결된 FOD

또한 제어 노드와 그로부터 나오는 전이가 연결된 출력 노드는 이름이 같아야 한다. 이름이 같지 않다면 출력과 제어 노드로부터 나오는 출력이 다르기 때문에 무모순성을 만족시키지 못한다. 하나의 제어 노드에서 나가는 전이는 하나 그리고 들어오는 전이는 반드시 하나 이상 가지고 있어야 한다. 어떤 제어 노드로 들어오는 전이가 없다면, 그 제어 노드의 실행을 보장할 수 없다. 따라서 모든 제어 노드의 실행의 완전성을 보장 할 수 없다.

3.1.2 FSM, TTS

FSM 과 TTS 는 상태와 전이들로 구성되어 있고, 초기상태를 갖고 있으며 종료상태는 가지고 있지 않다. 완전성의 관점에서 초기상태에서 갈 수 없는 상태들은 하나도 없어야 한다. 그림 8 에서 초기상태인 start 에서 갈 수 없는 S1 같은 상태들이 있다면, 그 상태로의 이동을 보장 할 수 없으므로, 완전성에 어긋난다.

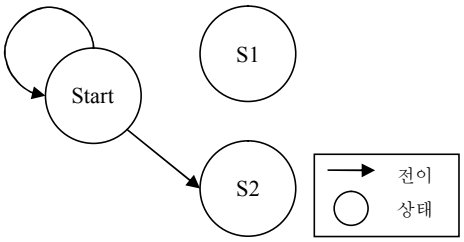

그림 8 초기상태로부터 갈 수 없는 상태가 있는 경우

그림 9 의 상태 S2 의 경우처럼 이름이 중복된 상태들이 있다면, 같은 조건에 서로 다른 상태로의 전이나 행동을 취하게 된다면 모순이 된다. 따라서 이런 모순의 문제를 없애기 위하여 같은 이름을 가진 상태를 허용하지 않는다.

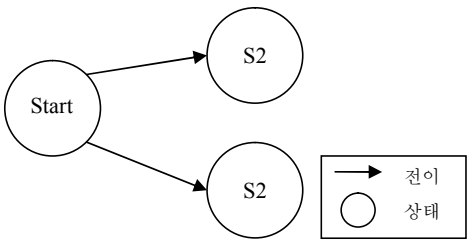

그림 9 같은 이름을 가지는 상태가 여러 개 있는 경우

어떤 하나의 상태는 자신을 시작 상태로 하는 0 개 이상의 전이를 가질 수 있다. 하나의 상태로부터 나오는 전이가 2 개 이상일 경우, 상태에서 다음 상태로 넘어갈 때, 해당 전이의 조건을 보고 어떤 전이를 실행할 것인지 결정해야 한다. 그림 10 의 FSM 예제는 2 개 이상의 전이의 조건들이 서로 충돌하는 예제이다. S1 의 상태에 있을 경우, n_variable_1 의 값이 5

보다 크면 어떤 전이를 실행해야 할 지 알 수 없다. 전이가 시작되는 상태를 start, 전이가 끝나는 상태를 end 라 하고 해당 전이의 조건의 진리치를 C(start, end)라 하자. start 에서 전이가 일어난다면 end 로 가는 전이의 조건은 true 가 되며 나머지 전이의 조건은 false 가 되어야 한다. 즉, n_variable_1 > 5 일 때, $C(start, end) \wedge \sim C(start, end) = false$, $\sim C(start, end) \wedge C(start, end) = false$)이므로, 이는 무모순성을 만족하지 못한다. 따라서 이를 검사하기 위해 위에서 언급한 SCP 를 해결하는 라이브러리를 사용하였다. TTS 에서는 시간 조건으로 최소 시간과 최대 시간을 갖고 있는데, 최소 시간이 최대 시간보다 짧다면 이것도 모순이 된다.

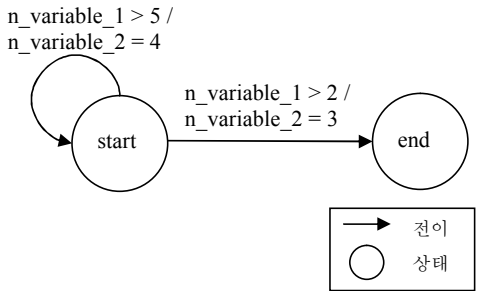

그림 10 FSM 의 예 4

3.1.3 SDT

SDT 는 조건들과 조건에 따른 행동을 결정하는 테이블 형태로 정의된다. 이전의 Quick Check 에서는 SDT 의 조건들의 진리치의 모든 조합에 대한 기술이 없어도 이를 검사하지 않아 완전성을 보장하기 어려웠다. 또한 조건들간에 중복된 조합이 있어도 이를 검사하지 않아 무모순성을 보장하기 어려웠다.

표 1,2 에서 C1, C2 는 조건이며, A1, A2 는 행동이다. 는 true, F 는 false 그리고 해당 열의 진리치가 만족 될 때 O 는 취할 행동을 나타낸다. 어떤 행동을 결정하기 위해서 그 행동의 모든 조건(행동이 정의된 열에 해당하는 조건)은 true, false, 또는 don't care 의 값이 정의되어 있어야 한다. 그렇지 않다면 조건에 의한 행동을 결정할 수 없다. 표 1 의 3 번째 열에서 C1 의 조건이 정의되어 있지 않으므로 행동을 결정할 수 없다. SDT 의 행동 부분에서는 어떤 한 열의 행동이 2 가지 이상 정의된다면, 두 행동은 서로 모순이 된다. 표 1 의 5 번째 열의 경우, A1 의 행동과 A2 의 행동이 정의 되어 있어 두 행동은 서로 모순을 일으킨다. 이 때 어떤 행동을 해야 할지 결정할 수 없다. 조건들의 임의의 조합이 SDT 내에 없다면 해당 조건들의 조합은 SDT 내에서 증명될 수 없다. 표 1 에서 (C1, C2)에 정의된 유효한 조건은 (T,F) 뿐이며, (F,F),(F,T),(T,T)는 제시되어 있지 않다. 이 경우 (F,F),(F,T),(T,T)일 때의 행동을 결정할 수 없으므로 완전성을 만족시키지 못한다. 또한 (T,F)가 4,5 열에 중복으로 제시되어 있다. 이렇게 되면 (T,F)가 만족될 때 4 열의 행동을 실행해야 할 지, 5 열의 행동을 실행해야 할지 알 수 없다. 따라서 무모순성을 만족시킬 수 없다.

C1	n_variable_1>3		T	T
C2	n_variable_2>3	T	F	F
A1	n_variable_1=6		O	O
A2	n_variable_1=8			O

표 1 조건의 진리치의 완전성과 행동의 무모순성을 지키지 못한 SDT 의 예

표 2 는 행동들이 서로 겹치는 부분이 없고 조건들이 모든 진리치의 조합이 정의되었으므로 완전성과 무모순성을 만족하는 것처럼 보인다. 하지만 n_variable_1 의 값이 4 보다 클 경우, C1 이 F, C2 가 T 인 경우는 조건이 서로 모순이기 때문에 만족 가능한 변수의 값을 찾을 수 없다. 이러한 모순 문제를 찾기 위해서 SDT 에서 CSP 해결 라이브러리를 사용하여 무모순성을 검증하였다. SDT 의 조건들은 조건의 진리표를 통하여 진리치가 결정될 경우, 무모순성과 완전성에 어긋나는지 조사한 뒤, 조건들의 조합을 검증한다.

C1	n_variable_1>3	T	T	F	F
C2	n_variable_1>4	T	F	T	F
A1	n_variable_1=6	O	O		
A2	n_variable_1=8			O	O

표 2 조건절이 모순을 일으키는 SDT 의 예

3.2 SDT, TTS, FSM 의 문법 검사

이전 Quick Check 는 행동 구문과 조건 구문을 구별하지 않고 같은 문법을 적용하여 검사하였다. 따라서 Quick Checker 에서는 행동 구문과 조건 구문의 문법검사를 다르게 하였다. 행동구문에서는 해당 노드의 출력 변수에 대한 할당만을, 조건 구문에서는 변수간의 비교나 연산만을 가지고 있어야 하므로 이를 지키는지의 여부를 검사하였다.

그림 11 은 각 spec 의 조건, 행동식들을 위한 EBNF(Extended Backus-Naur Form) [11]구문이다. 기존 NuSRS 의 Quick Check 문법과는 약간 다르게 작성하였다. 행동문 검사는 비단말 기호인 action_stmt 로부터 시작한다. 조건문 검사는 비단말 기호인

complex_condition 으로부터 시작한다. action_stmt 혹은 complex_condition 으로부터 시작되는 문법으로부터 단말 기호까지 탐색하여 문법을 만족하는지 검사하였다. 해당 구문에서 ID 는 변수 이름이며, Number 는 숫자, true, false 는 진리치를 의미한다.

```
action_stmt := assign_stmt, {';' , assign_stmt} | ';' |;
assign_stmt := ID, ':=', complex_condition;
complex_condition := logical_or_exp;
logical_or_exp := logical_and_exp, {'|',
                    logical_and_exp};
logical_and_exp := equality_exp, {'&', equality_exp};
equality_exp := relational_exp, {('='|'!='),
                    relational_exp};
relational_exp := additive_exp, {('<'|'>'|'<='|'>='),
                    additive_exp};
additive_exp :=multiplicative_exp, {('+'|'-'),
                    multiplicative_exp};
multiplicative_exp := unary_exp,{('*'|'/'|'%')},
                    unary_exp};
unary_exp :=('-'|'!'|), primary_exp;
primary_exp := Number|'true'|'false'|ID|
                    ('(',complex_condition,')');
```

그림 11 EBNF

3.3 Quick Checker 의 클래스 설계

Quick Checker 는 기존의 Quick Check 를 보완하고 유지보수성을 늘리기 위해 클래스 구조를 변경하여 개발하였다. 그림 12 은 이전에 있었던 Quick Checker 의 구성도이다. 그림 12 의 Quick Check Panel 은 모델 검사를 하기 위한 GUI 및 검증 기능을 구현한 클래스이며 Parser 를 사용하여 모델의 식에 사용되는 문법을 검사한다. Parser 는 JavaCC[12]로 구현되었다. Parser 는 각 Checker 에서 호출하여 해당 모델의 조건 조건과 행동을 구분하지 않고 검사만 하였으며, FOD 를 검사하지 못한 곳이 있었다. 또한 모델을 검사할 때 하나의 함수에서 각기 다른 타입의 모델을 검사하였다. 에러를 검사하는 코드마다 해당 에러메시지의 문자열을 에러메시지를 담당하는 변수에 할당하여 GUI 부분으로 보냈다.

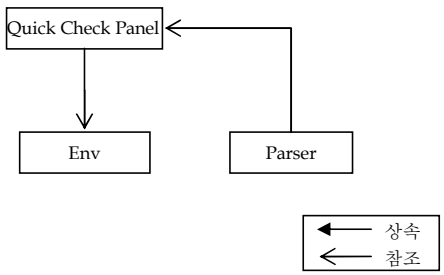

그림 12 이전 Quick Check 구성도

그림 13 은 본 연구에서 구현한 Quick Checker 구성도이다. Quick Checker 는 Checker 의 인터페이스를 구현한 FOD Checker, SDT Checker, FSM Checker 를 사용하여 검증 대상에 따라서 Checker 를 호출하도록 하였다. 또한 Quick Check Error 로 에러 유형을 모아 놓고 Quick Check Panel 로 출력하도록 구현하였다. 따라서 Quick Check Error 는 Quick Checker 실행 시 일어날 수 있는 에러의 유형과 메시지를 정적 상수 형태로 담고 있다. Parser 는 JavaCC 로 구현하였으며, 그림 11 을 기준으로 한 문법 검사 이외에 문법 구조에 맞는 파스 트리를 생성한다. 각 Checker 에서는 Parser 로부터 생성된 파스 트리를 Node2XCSP 에 전달하여 CSP 를 해결할 수 있는 라이브러리에 입력할 문서를 생성한다. 문서를 생성할 시 검사할 조건의 변수의 모든 범위에서 CSP 를 해결하는 것은 실제로 불가능하기 때문에 각 변수가 가질 수 있는 값들의 일부 범위를 제약사항으로 지정한다.

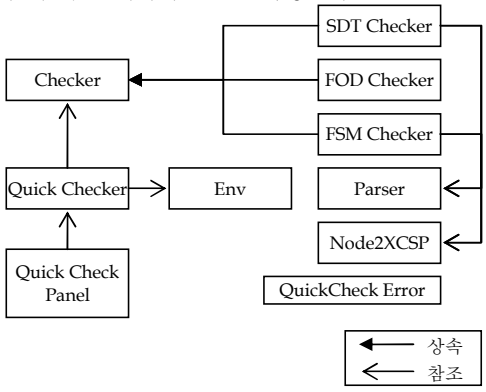

그림 13 현재 Quick Checker 구성도

그림 14 는 NuSRS 의 자료를 표현하기 위한 클래스 구성도이다. 기존의 클래스들은 NuSRS 의 GUI 에 의존해 있었으나 본 연구에서는 클래스들을 독립적으로 사용하기 위하여 GUI 의존부분을 제거했다. Env 는 NuSCR 의 자료구조를 갖고 있다. Env 는 자료구조의 루트인 FOD 객체를 참조하며 FOD 객체는 다른 노드객체인 FOD, FSM, SDT 들을 참조한다. NuSpec 은 일반적인 모델이 가져야 할 정보를 갖고 있다. Diagram 은 FOD, FSM 이 Diagram 형태로 나타낼 수 있는 정보를 갖고 있으며, Variable 은 NuSCR 의 변수정보를 나타낸다. Transition 클래스는 일반적인 전이를 나타내며 FSM Transition 클래스는 FSM 의 전이를 나타낸다. NuNode 는 FOD 에서 볼 수 있는 노드들을 나타내며, IONode 는 입출력을 나타내는 노드이다. RefNode 는 FSM, SDT, FOD 등의 모델을 나타내는 노드이며, NuSpec 과 1:1 대응한다. FSM 은 FSM 모델을 나타내는 클래스이며, TTS 또한 포함한다. SDT 는 SDT 모델을 나타내고, FOD 는 FOD 모델을 나타낸다.

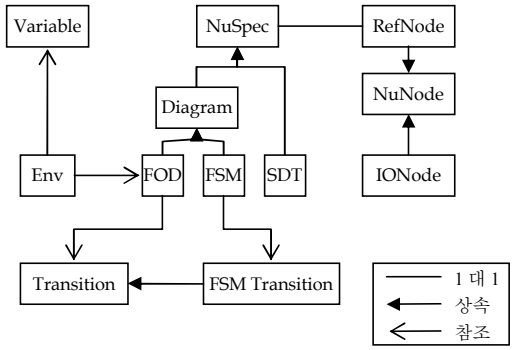

그림 14 NuSRS 자료 클래스 구성도

4. 사례연구

Quick Checker 를 구현하여 APR-1400 원자로 보호 시스템의 프로토타입 명세를 NuSCR 언어로 기술한 예제(*g_BP*)를 검사하여 이전 Quick Check 와 구현한 Quick Checker 를 비교하였다. Quick Checker 는 FOD, FSM, TTS, SDT 등과 변수들을 검사하였다. FOD 모델에서는 노드간의 연결, 제어 노드와 입출력 노드의 관계 등을 검사하며, FSM 과 TTS 에서는 상태간 연결 관계, 시작상태로부터의 연결 여부, 전이의 조건과 행동, 잘못된 변수 사용여부 등을 검사하며, SDT 에서는 변수 사용, 테이블의 진리치, 조건, 행동, 잘못된 변수 사용여부 등을 검사한다.

오류 유형	개수
전이없음	11
정의되지 않은 변수	66
초기상태로부터 도달할 수 없음	2

표 3 기존 Quick Check 가 *g_BP*(Bistable Process)를 검사한 결과

오류 유형	개수
전이없음	11
정의되지 않은 변수	66
초기상태로부터 도달할 수 없음	2
입력이 없는 노드	1
완전성 오류	14
무모순성 오류	18

표 4 현재 Quick Checker 가 *g_BP*(Bistable Process)를 검사한 결과

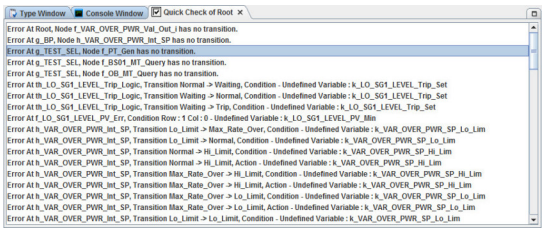

그림 15 기존 Quick Check 모듈

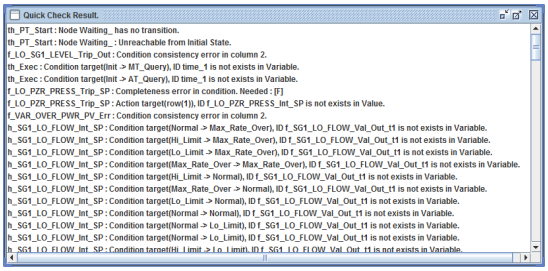

그림 16 현재 Quick Checker 모듈

그림 15 과 그림 16 는 Quick Check 와 Quick Checker 를 실행한 장면이고, 표 2 와 표 3 은 이전 Quick Check 와 새로 만든 Quick Checker 의 성능을 비교한 것이다. 표 2, 3 에서의 전이 없음은 FOD, FSM, TTS 등에서 어떤 노드나 상태의 전이가 없다는 것이고, 정의되지 않은 변수는 SDT, TTS, FSM 등에서 조건구문이나 행동구문에 기술한 변수를 NuSCR 명세의 변수들 중에서는 찾을 수 없는 것이다. 초기상태로부터 도달할 수 없음은 FSM, TTS 에서 그림 10 의 S1 처럼 시작상태로부터 도달할 수 없는 노드가 존재한다는 것이다. 이전 Quick Checker 에서는 APR-1400 원자로 보호 시스템의 프로토타입 명세로부터 전이 없음, 정의되지 않은 함수, 초기상태로부터 도달할 수 없는 경우의 오류만이 검출되었다. 따라서 SDT Table 의 진리치가 완전성과 무모순성을 지킬 수 있도록 작성 되어 있는지, 수식이 서로 문제를 일으키는지 검사할 수 없었다. 개선된 Quick Checker 에서는 표 3 의 내용처럼 완전성 오류와 무모순성 오류, 그리고 입력이 없는 노드를 추가로 검사하였다. 완전성 오류와 무모순성 오류는 일부 SDT 내용 작성에 오류가 있음을 나타내는 것이고, 입력이 없는 노드는 FOD 에서 제어 노드가 전이를 받고 있지 않도록 작성되었다는 것을 의미한다. 결론적으로 Quick Checker 는 이전에 잡아내지 못했던 오류를 잡을 수 있게 되었다.

5. 결론

안전 필수 시스템은 소프트웨어 요구사항분석 단계부터 명세에 대한 검증을 거친다. 본 연구에서는 정형명세 언어인 NuSCR 로 명세 된 소프트웨어 요구사항모델을 대상으로 완전성과 무모순성의 관점에서 생각해봐야 할 점들을 정의하고 이를 자동으로 검사하기 위한 도구인 Quick Checker 를 제시하였다. 사례연구를 통해 이전에 NuSRS 에 포함되어있던 기능인 Quick check 에서 검사하지 못했던 부분들을 찾아 낼 수 있음을 보여 Quick Checker 의 성능이 이전보다 개선되었음을 보였다. 이를 이용하여 이후의 개발 단계에서 수행하는 NuSCRtoSMV 를 이용한 모델체킹이나 NuFTA 를 이용한 안전성분석의 과정을 좀 더 정확하게, 그리고 적은 비용으로 수행할 수 있을 것으로 기대된다.

참고문헌

[1] Jaemyung Cho, Junbeom Yoo, Sungdeok Cha "NuEditor – A Tool Suite for Specification and Verification of NuSCR" In proceeding of Second ACIS International Conference on Software Engineering Research, Management and Applications (SERA2004), pp298-304, LA, USA, May 5-7, 2004.

[2] Junbeom Yoo, Taihyo Kim, Sungdeok Cha, Jang-Su Lee, Han Seong Son, "A Formal Software Requirements Specification Method for Digital Nuclear Plants Protection Systems," Journal of Systems and Software, Vol.74, No.1, pp73-83, 2005.

[3] KNICS.http://www.knics.re.kr/english/eindex.html.

[4] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Guinchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking" In Proceeding of International Conference of Computer-Aided Verification (CAV 2002). Copenhagen, Denmark, July 27-31, 2002

[5] Sanghyun Yun, Dong-Ah Lee and Junbeom Yoo, "NuFTA: A CASE Tool for Automatic Software Fault Tree Analysis," Transactions of the Korean Nuclear Society Spring Meeting 2010, pp.855-856, May 27-28, Pyeongchang, Korea, 2010.

[6] C Heitmeyer, "Software Cost Reduction" In J. J. Marciniak, editor, Enc. of Software Engineering. 2nd edition, 2002.

[7] K. L. McMillan, Edmund M. Clarke, "Model Checking" Foundations of Software Technology and Theoretical Computer Science Lecture Notes in Computer Science, 1997, Volume 1346/1997, 54-56, DOI: 10.1007/BFb0058022

[8] KAERI (Korea Atomic Energy Research Institute), "SRS for Reactor Protection System.", KNICS-RPS-SVR131-01 Rev.00, 2005

[9] Constance L.Heitmeyer, Ralph D. Jeffords, and Bruce G.Labraw "Automated Consistency Checking of Requirements Specifications" ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 5 , Issue 3 (July 1996), Pages: 231 – 261, 1996

[10] Andrei A. Bulatov, Andrei A. Krokhin and Peter Jeavons, "Constraint Satisfaction Problems and Finite Algebras", Lecture Notes in Computer Science, 2000, Volume 1853/2000, 272-282, DOI: 10.1007/3-540-45022-X_24

[11] LM Garshol, 2003, "BNF and EBNF: What are they and how do they work" [Online], Available at http://www.garshol.priv.no/download/text/bnf.html, [Last access: 2010-12-26]

[12] Viswanathan Kodaganallur, "Incorporating Language Processing into Java Applications: A JavaCC Tutorial," IEEE Software, vol. 21, no. 4, pp. 70-77, July/Aug. 2004, doi : 10.1109/MS.2004.16