반응형 시스템을 위한 올바른 환경 모델의 생성

권령구*, 권기현
경기대학교 컴퓨터과학과
{rkay8496, khkwon}@kyonggi.ac.kr

Generating a Correct Environment Model for Reactive System

Ryoungkwo Kwon*, Gihwon Kwon
Department of Computer Science, Kyonggi University

요 약

반응형 시스템은 외부 환경과 지속적으로 상호작용하는 시스템이다. 반응형 시스템이 요구 사항을 만족하는지를 검사하기 위해서 상호 작용을 수행할 환경 모델을 구축 해야 한다. 환경 모델은 반응형 시스템에게 항상 올바른 입력들을 제공 해야만 검사 결과의 신뢰성을 보장할 수 있기 때문에 올바른 입력들을 포함하는 것은 매우 중요하다. GR(1) Synthesis 는 수학적 정형 기법으로써 논리식으로 표현된 요구 사항을 보장할 수 있는 모델을 자동으로 생성할 수 있다. 이 기법을 이용하여 올바른 환경 모델을 생성하고 반응형 시스템에게 올바른 환경의 입력들을 자동으로 제공 함으로써 반응형 시스템의 행위가 올바른지를 환경 모델 상에서 검사하였다. 또한 사례 연구를 통해 환경 모델을 이용한 검사가 기존 방법에 비해 우수함을 알 수 있었다.

1. 서론

반응형 시스템은 외부 환경의 입력을 받아들이고 자신의 상태를 변경시키거나 외부 환경으로 출력을 생성 한 후 다음 입력을 기다린다. 반응형 시스템은 이러한 환경과의 상호작용을 지속적으로 이루게 된다 [1]. 일반적인 목적을 가지는 많은 시스템(또는 소프트웨어)들이 위와 같은 특성을 가지고 있다.

수 많은 반응형 시스템 중 항공이나 차량에 탑재되는 반응형 시스템에서 오류나 비 정상적인 시스템의 행위로 인해 인명 피해로 직결될 수 있다. 그러므로 반응형 시스템 정확성을 보장하거나 검사하는 것은 중요하다. 어떠한 시스템이 정확하다는 것은 해당 시스템이 기대되는 요구 사항을 올바르게 만족한다고 할 수 있다. 다시 말해, 해당 시스템이 반응형 시스템일 때 외부 환경의 입력들에 대하여 항상 올바른 출력들을 생성할 수 있어야 한다.

테스팅이나 시뮬레이션과 같은 방법들은 시스템이 기대되는 요구 사항을 올바르게 모두 만족하고 있는지를 확인하기 위한 검사 방법이다. 이 방법들은 시스템을 묘사한 모델 또는 실제 구현물에 특정 입력들을 제공하여 시스템의 행위를 검사하는 과정을 거친다. 하지만, 반응형 시스템의 관찰 가능한 행위의 수는 셀 수 없이 많고 개별 행위의 길이는 무한하므로 모든 행위를 완전히 검사한다는 것은 불 가능 하다[2]. 많은 비용을 들여 위 방법들을 수행하더라도 검사하지 못한 시스템의 행위에서 치명적인 문제를 초래할 가능성을 가지고 있다.

행위 검사를 수행하기 위해서는 테스트 케이스들이 요구 된다. 테스트 케이스는 대상 시스템에 대한 입력과 기대하는 출력의 쌍을 명세한 것이다. 하지만, 반응형 시스템에서 행위라는 것은 입력과 출력의 무한한 시퀀스이고 테스트 케이스는 행위의 부분만을 바라보기 때문에, 반응형 시스템의 행위을 검사하기에 충분하지 않다. 또 다른 측면으로 테스트 케이스들의 입력을 어떻게 올바르게 결정할 것인가 이다. 이 점이 중요한 이유는 테스트 케이스가 정확함이 보장 되지 않으면, 테스트의 결과를 신뢰할 수 없다. 만약 수 많은 입력들이 존재하고 그것들 사이에 어떠한 관계가 존재할 때-예를 들어 입력들 간의 의존 관계가 존재하여 개별 입력이 독립적으로 발생하지 않을 경우- 입력들이 실제 발생 가능한 지를 판단해야 한다. 그리고 잘 못된 테스트 케이스를 올바르게 수정하기 위해 무엇이 잘 못되었냐를 알기 위한 노력도 무시할 수 없다. 이러한 특성에 따라 시스템의 행위를 올바르게 검사하기 위해 올바른 테스트 케이스의 생성과 양적인 문제를 결정하는 것이 중요하다. 종합적으로 이러한 측면들은 반응형 시스템 개발에서 정확성과 생산성에 큰 관련이 있다. 만약, 올바른 테스트 케이스의 생성하기 위해 수학적 기반의 자동화된 방법을 적용할 수 있다면 정확성과 생산성 향상에 도움을 줄 수 있을 것이다.

본 논문에서는 반응형 시스템에 대한 올바른 입력들을 포함하는 환경 모델을 자동으로 생성하는 연구를 보인다. 본 논문에서 환경 모델이라고 함은 반응형 시스템에 올바른 입력을 제공해 주는 객체로 정의한다. 환경 모델을 자동으로 생성하기 위하여 GR(1)

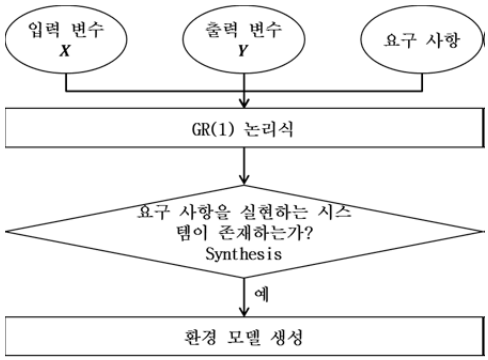
본 연구는 경기도의 경기도지역협력연구센터사업의 일환으로 수행하였음.[2012-0271,모바일 플랫폼 기반의 콘텐츠 응용 소프트웨어 기술 개발]

Synthesis 라는 수학적 기법을 사용 한다. 이 기법은 반응형 시스템이 보장해야 하는 출력과 외부 환경의 입력을 선형 시제 논리(Linear Temporal Logic)[2]의 구문을 제약한 GR(1)(Generalized Reactivity(1)) 논리식으로 작성 한다. Synthesis 알고리즘은 GR(1) 형태의 논리식을 받아들이고 논리식이 실현 가능하다면 오토마타와 같은 모델을 자동으로 생성할 수 있다. 이 모델은 올바른 모든 입력들의 시퀀스와 그것들에 대한 반응형 시스템의 올바른 출력들을 보장 한다. 따라서 이 모델을 반응형 시스템에 올바른 입력들을 발생시키고 행위를 검사하는 것에 사용한다면, 행위 검사를 위한 올바른 입력의 결정에 큰 도움을 줄 것이다.

본 논문의 구성은 다음과 같다. 2 장에서는 GR(1) Synthesis 의 소개와 이 방법을 이용하여 환경 모델을 생성하는 과정을 설명 한다. 3 장에서는 환경 모델을 오토마타로 정의하고 이 것의 의미를 기술 한다. 4 장에서는 2 장에서 설명한 과정을 따라 환경 모델을 생성하여 로봇의 행위를 검사하는 사례 연구와 제안하는 방법의 특징 및 장점을 보인다. 마지막으로 5 장에서 결론 및 향후 연구를 보인다.

2. GR(1) Synthesis

GR(1) Synthesis 에서 GR(1)은 선형 시제 논리의 구문을 제약한 논리식의 형태를 의미하고 Synthesis 는 요구 사항을 실현 가능한 구현물을 곧바로 생성하는 수학 기반의 정형 기법이다. Synthesis 문제는 [1][3]과 같이 많은 연구가 진행 되었는데, 계산 복잡도가 상태 공간에 지수적으로 증가하였다. 그러나 [4]에서 상태 공간에 대해 계산 복잡도를 다항 시간으로 줄인 GR(1) 논리식을 제안 하였다. GR(1) 논리식은 다음과 같은 형태이며, $\varphi = \varphi_e \Rightarrow \varphi_s$, φ_e는 환경의 입력에 대한 논리식이며, φ_s는 반응형 시스템이 보장해야 하는 출력에 대한 논리식이다.



(그림 1) 환경 모델을 얻는 GR(1) Synthesis 절차

GR(1) Synthesis 를 통해 환경 모델을 얻는 과정은 아래 그림 1 의 절차를 따른다. GR(1) Synthesis 를 통해 환경 모델을 생성하기 위해 먼저 환경과 반응형 시스템의 입/출력을 이진 변수로 정의 하고, 정의된 변수들 상에서 GR(1) 논리식을 작성한다. Synthesis 알고리즘은 GR(1) 논리식을 입력으로 하여 실현 가능성을 확인한다. 실현 가능하다는 것은 환경과 시스템의 2 인 게임으로 설명할 수 있다. 논리식 φ으로 표현된 모든 가능한 환경의 입력에 대해서 출력을 항상 올바르게 만족할 수 있는 시스템이 존재한다면 이것은 실현 가능하다고 한다. 결국 이러한 시스템을 환경 모델로 생성하게 된다.

3. 환경 모델

앞서 설명한 Synthesis 알고리즘은 GR(1) 논리식 φ이 실현 가능하다면 환경 모델을 생성 한다. 생성된 환경 모델은 올바른 환경의 입력 시퀀스들과 시스템의 올바른 출력을 달성할 수 있는 행위들을 포함 한다. 우리는 검사 대상 반응형 시스템에게 올바른 환경의 입력을 생성해 주기 위해서 환경 모델을 이용한다. Synthesis 알고리즘으로 생성한 환경 모델은 오토마타 $A = (S, X, Y, S_0, \delta, \gamma)$로 정의 한다.

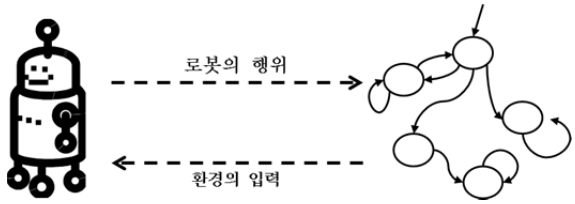
- S – 모든 상태들의 유한 집합.
- X – 입력 변수들의 유한 집합.
- Y – 출력 변수들의 유한 집합.
- $S_0 \subseteq S$ – 초기 상태들의 유한 집합.
- $\delta: S \times 2^X \rightarrow S$ – 상태 전이 함수. $\delta(s, 2^X) = s' \in S$ 일 때, 상태 s에서 다음에 참이 되는 입력 변수들, 2^X에 의해 다음 상태 s'로 전이 한다.
- $\gamma: S \rightarrow 2^Y$ – 상태 라벨링 함수. $\gamma(s) = 2^Y$ 일 때, 상태 s에서 출력 변수들, 2^Y는 참이다.

오토마타로 정의된 환경 모델에서 올바른 입력의 행위는 무한한 입력 시퀀스, $\sigma^e = 2_1^X, 2_2^X, 2_3^X, \ldots$ 이며, 입력 시퀀스 상에서 출력의 행위는 상태들의 무한 시퀀스, $\sigma^s = s_0, s_1, s_2, \ldots$로 정의 된다. 출력의 행위 σ^s는 초기 상태, $s_0 \in S_0$에서 시작하고 상태 전이 함수 δ에 따라, $\forall_{i \geq 0} \cdot \delta(s_i, 2_{i+1}^X) = s_{i+1}$, 상태 전이가 이루어 진다. σ^e에서 $\forall_{i \geq 1} \cdot 2_i^X$은 검사 대상 반응형 시스템에게 생성할 참인 입력 값들의 집합으로 해석 한다. σ^s는 상태 라벨링 함수 $\forall_{i \geq 0} \cdot \gamma(s_i)$에 의해 s_i상태에서 참이 되는 출력 변수들이며 기대하는 시스템의 행위들이다.

환경 모델의 상태 s_i에서 환경 모델은 2_{i+1}^X의 입력들을 반응형 시스템에게 생성하면 상태 s_{i+1}로 전이 한다. 입력들에 따라 반응형 시스템은 어떠한 행위를 할 것이며, 이 행위는 환경 모델의 상태 s_{i+1}에서 출력 변수들과 의미적으로 동일 해야 한다. 만약 그렇다면 반응형 시스템의 행위는 올바르다고 할 수 있다. 동일하지 않다면 실제 반응형 시스템의 행위가 입력에 대하여 올바르게 행동하지 않았음을 보이는 것이다.

4. 사례 연구

사례 연구를 통해 보여줄 대상 반응형 시스템은 Lego MindStorm NXT[5] 로봇이며 자신이 수행해야 할 기대 작업이 주어지고 다양한 센서를 통해 외부 환경의 상태를 감지하여 적합한 행위를 할 책임이 있다. 따라서, 로봇은 반응형 시스템으로써 아주 적합하다. 환경 모델을 이용하여 로봇의 행위를 검사하는 아이디어의 전체 모습은 그림 2 와 같다.



(그림 2) 환경 모델을 이용한 반응형 시스템 검사

우리는 요구 사항으로부터 실제 로봇의 구현물과 환경 모델을 얻을 수 있다. 여기서 구현물이란 요구 사항에 따라 작성된 프로그램 코드이다. 그리고 환경 모델은 GR(1) Synthesis 를 통해 생성하며 구현물에게 올바른 환경의 입력들을 발생 시키고 구현물의 행위를 다시 받아들여 올바른 행위인지를 검사 한다.

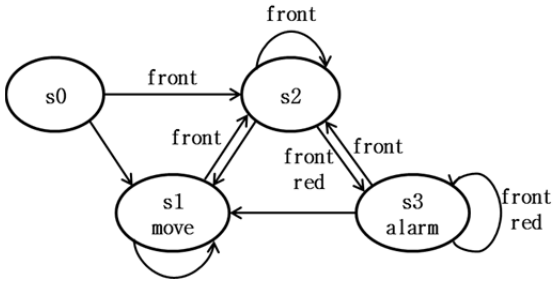
사례 연구의 요구 사항은 다음과 같다. 1) 로봇은 전방으로 이동하거나 멈출 수 있다. 2) 전방에 장애물이 있다면 이동을 멈추고, 장애물이 사라지면 이동한다. 3) 전방에 장애물이 있고 빨간색이 감지 되는 동안은 알람을 울린다. 요구 사항을 요약하면 로봇은 항상 전방으로 이동하다가 장애물이 있으면 멈추고, 빨간색이 감지되면 알람을 울리는 것이다. 요구 사항에서 올바른 환경의 입력을 발생할 때 고려할 부분이 있다. 그것은 환경의 입력들 간의 의존 관계로써, 장애물과 빨간색 입력이 관련 된다. 빨간색이 감지 될 수 있는 경우는 반드시 장애물이 존재할 때에만 가능한 것으로, 빨간색 입력이 장애물과 독립적으로 발생할 수 없다. 이러한 의존 관계를 통해서 검사할 로봇의 행위는 다양하게 존재할 수 있다.

우리는 제 3 자에게 요구 사항을 문서와 구두로 설명한 후 프로그램 코드를 얻었으며, 환경 모델은 직접 생성 하였다. 그리고 생성된 환경 모델에서 로봇에게 환경의 입력을 생성하고 로봇의 행위를 환경 모델에서 자동으로 검사할 수 있도록 하였다. 환경의 입력은 환경 모델 상에서 결정되어 로봇에게 직접 전달하기 때문에, 전방의 장애물을 감지하는 초음파 센서와 색을 판별하는 색깔 센서를 로봇에게 직접 장착할 필요는 없다. JTLV[6]라는 도구를 사용하여 Synthesis 과정을 수행하였으며, 요구 사항이 실현 가능하다면 환경 모델을 생성하였다. Synthesis 의 입력이 되는 논리식 $\varphi = \varphi_e \Rightarrow \varphi_s$은 다음과 같다.

$$\varphi_e = \begin{Bmatrix} (\neg front \wedge \neg red) \\ \wedge \square(\bigcirc red \Rightarrow front) \\ \wedge \square\big((\bigcirc red \wedge front) \Rightarrow \bigcirc front\big) \\ \wedge \square\diamond(TRUE) \end{Bmatrix}$$

$$\varphi_s = \begin{Bmatrix} (\neg move \wedge \neg alarm) \\ \wedge \square\big(\bigcirc front \Leftrightarrow \bigcirc(\neg move)\big) \\ \wedge \square(\bigcirc red \Leftrightarrow \bigcirc alarm) \\ \wedge \square\diamond(move \vee front) \end{Bmatrix}$$

논리식 φ는 요구 사항을 따르는 환경과 시스템의 행위가 포함되어 있다. 환경 논리식 2, 3 행은 입력들 간의 의존 관계를 표현하는 것으로써 이진화된 선형 시간 t에서 빨간색이 발생 되었다는 것은 시간 $t-1$에서 전방에 장애물이 있어야 한다는 것을 의미한다. 3 행에서는 선형 시간 t에서 빨간색이 발생되었다면 마찬가지로 시간 t에서도 장애물이 존재해야 함을 의미한다. 환경과 시스템 공통으로 1 행은 환경의 입력과 로봇 출력의 초기 값을 지정하며 4 행은 환경과 로봇의 궁극성 조건으로써, 환경은 명시적인 궁극성 조건이 없지만, 로봇은 무한한 행위 상에서 전방에 장애물이 있지 않는 한 이동하는 행위가 무한히 자주 나타나야 하는 것을 의미 한다. Synthesis 알고리즘을 통해 위 요구 사항, 논리식 φ이 실현 가능함을 확인하였고 그림 3 의 환경 모델을 생성하였다. 환경 모델에서 상태는 로봇의 출력을 나타내며 간선은 환경 모델이 발생할 입력이다.



(그림 3) 생성된 환경 모델

제 3 자로부터 구현된 프로그램 코드를 환경 모델을 통해 검사하였다. 최초에 구현된 코드는 다음과 같다.

```
While(true) {
        if(obstacle) {
                stop();
                if(red) {
                        alarmOn();
                }
        } else {
                move();
        }
}
```

환경 모델 상태 3 에서 상태 1 로 전이할 때, 로봇의 잘못된 행위를 확인 하였다. 환경 모델 상에서는 상태 3 에서 로봇은 알람을 울리고 멈춰 있다가 상태 1 로 전이 하면서 환경 모델은 장애물과 빨간색의 입력을 해제 하면, 로봇은 알람을 끄고 이동해야 한다. 하지만, 코드에서는 멈춰서 알람을 울리다가 모든 입력이 해제 되었을 때 이동을 하면서 알람을 울렸다. 다음은 이 정보를 바탕으로 코드를 수정한 결과이며 두 군데에 알람을 울리지 않는 코드를 추가 하였다.

```
While(true) {
        if(obstacle) {
                alarmOff();
                stop();
                if(red) {
                        alarmOn();
                }
```

```
        } else {
                alarmOff();
                move();
        }
}
```

위 결과를 통해 테스트 케이스와 환경 모델을 비교해 볼 수 있다. 단순 입/출력 쌍으로서 테스트 케이스를 작성한다면 아래의 총 4 가지 케이스를 식별할 수 있다. 단, 세 번째 테스트 케이스는 사실상 입력들간의 의존관계에 의해 사실상 무의미 하다. 이와 같은 테스트 케이스들로 검사를 수행하게 되면 잘 못된 코드에서도 모두 기대하는 출력을 얻을 수 있다.

- $(front, red, \neg move, alarm)$,
- $(front, \neg red, \neg move, \neg alarm)$
- $(\neg front, red, move, alarm)$
- $(\neg front, \neg red, move, \neg alarm)$

하지만, 환경 모델로 검사하면 반응형 시스템의 잘못된 행위를 발견할 수 있다. 환경 모델상에서 올바른 입력의 행위 $\sigma^e = \{front, red\}, \{\neg front, \neg red\}, \ldots$와 출력의 행위 $\sigma^s = s_2, s_3, s_1, \ldots$가 주어졌을 때, 환경 모델의 실행 $\sigma = s_2, \{front, red\}, s_3, \{\neg front, \neg red\}, s_1, \ldots$ 이 정의 된다. 결국 무한한 실행 상에서 $\gamma(s_1)$ 의 결과가 반응형 시스템이 입력 $\{\neg front, \neg red\}$ 을 통해 실제 수행한 행위가 일치하지 않았기 때문에 테스트 케이스들에서는 확인할 수 없었던 잘못된 행위를 발견했다고 설명 할 수 있다.

본 연구에서 환경 모델을 이용하여 반응형 시스템의 행위를 검사할 때 몇 가지 장점을 확인할 수 있었다. 첫 번째는 Synthesis 알고리즘을 통해 올바른 환경의 입력 시퀀스들을 자동으로 생성할 수 있다. 두 번째는 환경 모델의 무한한 실행 시퀀스들을 포함하기 때문에 테스트 케이스로는 어려운 부분도 검사 가능하다. 세 번째는 사람이 검사 과정에서 입력들을 직접 만들거나 수동으로 발생할 때 실수의 가능성을 줄일 수 있다. 네 번째는 가상의 환경모델을 통해 검사하였기 때문에 실제 센서 장치들이 필요하지 않았으므로, 전체 시스템이 실제로 완전히 구축되지 않아도 이른 시점에 검사를 수행하고 잘못된 행위를 확인하여 수정할 수 있다. 종합적으로 이러한 장점들이 반응형 시스템의 정확성과 생산성 향상에 도움을 줄 수 있다.

5. 결론 및 향후 연구

본 논문에서 GR(1) Synthesis 통해 환경 모델을 생성하고 이것을 이용해 반응형 시스템의 행위를 검사하는 방법을 보였다. 반응형 시스템은 지속적으로 외부 환경의 입력을 받아들여 적합한 행위를 하는 상호작용을 하며, 기대 했던 요구 사항을 항상 만족할 수 있어야 한다. 따라서, 반응형 시스템의 정확성을 달성하기 위해 테스팅이나 시뮬레이션과 같은 방법을 적용하여 행위를 검사해야 한다. 하지만, 반응형 시스템의 가능한 행위의 수는 무한하므로 완전히 검사한다는 것은 사실상 불가능하다. 또한 검사를 수행하기 위한 테스트 케이스들은 행위의 부분만을 검사할 수 있으므로, 많은 양의 테스트 케이스들이 요구 된다. 그리고 잘못된 테스트 케이스로 인해 테스트의 결과는 신뢰할 수 없게 된다. 이러한 특성으로 인해 테스트 케이스를 수정 또는 추가하는 일련의 작업들로 인해 반응형 시스템의 정확성과 생산성이 저하 된다. 하지만 Synthesis 알고리즘은 GR(1) 논리식으로 작성된 요구 사항을 받아들여 실현 가능성을 확인하여 올바른 모든 환경의 입력을 생성 할 수 있는 환경 모델을 생성할 수 있다. 생성된 환경 모델을 이용하여 자동으로 올바른 환경의 입력을 반응형 시스템에 전달한다면 무한한 행위의 관점에서 검사 가능하며 사람의 실수나 잘못된 테스트 케이스가 만들어진 가능성을 제거할 수 있다. 궁극적으로 GR(1) Synthesis 를 통해 생성된 올바른 환경 모델을 이용하여 반응형 시스템의 행위를 검사함으로써 정확성 및 생산성 향상에 기여 할 수 있다.

본 논문에서 제안한 방법은 GR(1) Synthesis 와 환경 모델을 이용하는 초기 연구이며, 비교적 간단한 로봇의 행위를 검사하는 사례 연구를 보였다. 이 연구를 더욱 진전하기 위해 보다 복잡하고 환경의 많은 입력들이 존재하는 로봇의 시나리오들을 개발해야 할 것이다. 그리고 로봇 분야에 국한되지 않고 외부 환경과 상호 작용이 활발하며 안전성이 중요한 분야(GUI, 논리 회로 등)에 적용하여 반응형 시스템의 정확성과 생산성 향상에 이 연구가 가치 있다는 것을 보일 것이다.

참고문헌

[1] Pnueli. A, Rosner. R, “On the synthesis of a reactive module”, In Proc. 16th ACM Symp. Princ. of Prog. Lang., pp. 179–190 (1989)

[2] E. M. Clarke, O. Grumberg, D. A. Peled, “Model Checking”, MIT Press, 1999.

[3] A. Church, “Logic, arithmetic and automata”, In Proc. 1962 Int. Congr. Math., pages 23–25.

[4] Piterman. N, Pnueli. A, Sa'ar. Y, “Synthesis of Reactive(1) Designs”, In Proc. 7th Intl Conference on Veri¯cation, Model Checking, and Abstract Interpretation (VMCAI'06), LNCS vol. 3855, pp. 364-380, 2006.

[5] Lego.com Mindstorms, http://mindstorms.lego.com

[6] Pnueli. A, Sa'ar. Y, Lenore D. Zuck, “Jtlv: A Framework for Developing Verification Algorithms”, CAV 2010: 171-174