정형 기법을 이용하여 기능적으로 정확한 컨트롤러 개발 사례

김태균, 권기현
경기대학교 컴퓨터과학과
{bigvirus, khkwon}@kyonggi.ac.kr

조지만, 정도균, 이상은
소프트웨어진흥원 SW 공학센터
{kidjoji, jdk, selee} @nipa.kr

Case Study of Developing Functionally Correct Controllers using Formal Method

Taekyun Kim, Gihwon Kwon
Dept. of Computer Science, Kyonggi University

Jiman Cho, Dokyun Jung, Sangeun Lee
SW Engineering Center, National IT Industry Promotion Agency

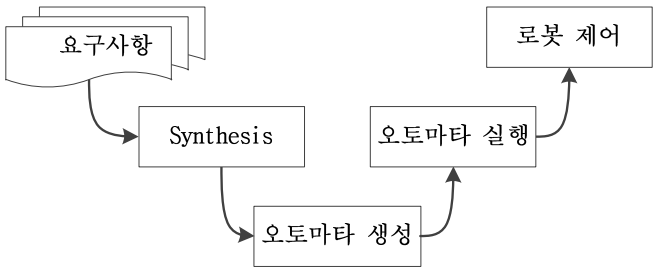
요 약

항공, 자동차, 로봇 등 안전-필수 분야에서 IT 융합 기술이 널리 사용되고 있다. 이들 분야에서 사용되는 IT 핵심 요소 중의 하나가 컨트롤러이다. 컨트롤러의 오 동작은 안전과 매우 밀접하기 때문에, 개발된 컨트롤러의 기능이 정확하다는 것을 입증해야 한다. 본 논문에서는 정형 기법을 이용하여 기능적으로 정확한 컨트롤러 개발 사례를 보인다.

1. 서론

컨트롤러는 항공, 자동차, 로봇 등 안전 필수 분야에서 널리 사용되고 있으며, 안전과 매우 밀접하기 때문에 개발 된 컨트롤러의 기능이 정확하다는 것을 입증해야만 한다. 하지만 컨트롤러의 기능이 정확하다는 것을 입증하기 위해서는 테스팅 혹은 검증 작업 등을 필요로 하며, 그로 인해 테스팅 및 검증에 많은 노력이 필요로 한다.

하지만 논문에서 제안하는 Synthesis 기법은 시스템의 요구사항 또는 명세로부터 시스템의 행위에 부합하는 오토마타/시스템이 자동으로 생성하기 때문에 컨트롤러의 기능이 정확하다는 것을 입증하기 위한 테스팅 또는 검증 과정이 필요하지 않다.



(그림 1) 제안 방법

본 논문의 구성으로는 2 장 LTL Synthesis 에서는 Synthesis 에 대하여 간략히 설명하고, 3 장, 4 장에서는 로봇에 적용한 사례와 결과에 대하여 이야기하고, 마지막 5 장, 결론으로 논문을 마무리 짓는다.

□ 본 연구는 지식 경제부의 소프트웨어공학센터의 사업일환으로 수행하였음. [2012-0203, 고신뢰 융합 SW 를 위한 필수 SW 공학 기술 조사 용역]

2. LTL Synthesis

LTL(Linear Temporal Logic)은 선형 시제 논리로서 시스템 행위에 관한 다양한 요구 사항을 기술하는 명세 언어이다. 행위에 관한 요구 사항을 기술하기 위하여 명제 논리에 사용되는 연산자 ¬, ∨ 뿐만 아니라, ○, □, ◇ 등의 시제 연산자를 제공한다. AP 를 단순 명제 집합이라고 하자. LTL 구문 규칙은 다음과 같다.

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \Diamond\varphi \mid \Box\varphi$$

여기서 $p \in \mathrm{AP}$는 단순 명제이고 ¬, ∨는 각각 부정과 논리합을 나타내는 명제 연산자이다. 그리고 ○, □, ◇는 다음(next), 항상(always), 미래(future)를 나타내는 시제 연산자이다.

LTL 논리식 φ의 의미는 단순 명제 $p \in \mathrm{AP}$에 진리값을 배정한 무한 시퀀스 σ상에서 정의된다. 집합 $\sigma(i)$는 무한 시퀀스 σ의 i번째 위치에서 참인 단순 명제의 집합이다. 무한 시퀀스 σ의 i번째 위치에서 논리식 φ가 만족되었다는 것을 $\sigma, i \vDash \varphi$로 표기하고, 이것을 이용하여 각 연산자의 의미를 정의하면 다음과 같다.

$\sigma, i \vDash p$	iff	$p \in \sigma(i)$
$\sigma, i \vDash \varphi_1 \vee \varphi_2$	iff	$\sigma, i \vDash \varphi_1$ or $\sigma, i \vDash \varphi_2$
$\sigma, i \vDash \bigcirc\varphi$	iff	$\sigma, i+1 \vDash \varphi$
$\sigma, i \vDash \Diamond\varphi$	iff	$\exists_{j \geq i} \cdot \sigma, j \vDash \varphi$
$\sigma, i \vDash \Box\varphi$	iff	$\forall_{j \geq i} \cdot \sigma, j \vDash \varphi$

의미 정의에 따르면 논리식 $\bigcirc\varphi$는 시퀀스의 바로

'다음' 위치에서 φ가 참임을 나타낸다. 또한 $\Diamond\varphi$는 시퀀스에서 '언젠가' φ가 참임을 나타낸다. 그리고 $\Box\varphi$는 시퀀스의 모든 위치에서 φ가 '항상' 참임을 나타낸다.

위에서 소개한 선형 시제 논리를 사용하여 요구사항 또는 명세에 대한 식을 작성하고, 오토마타 혹은 시스템을 자동으로 생성하는 것이 LTL Synthesis 이다. 이러한 LTL Synthesis 는 요구사항으로부터 자동으로 오토마타/시스템이 생성되므로 에러가 존재하지 않는다. 따라서 검증 또는 테스팅 과정이 필요하지 않으므로 시간과 인력 등의 비용을 줄일 수 있다.

하지만 전체 LTL 구문을 사용하여 요구사항을 작성하게 되면 생성되는 오토마타의 크기는 LTL 식 크기 n 승만큼의 크기가 되며, 따라서 이러한 문제를 해결하기 위해 특수한 구조를 가지는 LTL 식을 사용함으로써 오토마타의 크기를 2^n에서 n^3으로 줄일 수 있다. 이러한 식의 구조를 GR(1) 형식이라고 하며, 구조는 크게 환경과 시스템으로 구분된다. GR(1)형식은 $\varphi_e \rightarrow \varphi_s$로 구성 되며 여기서 φ_e는 환경에 대한 가정이며, φ_s는 시스템에 대하여 기대되는 행위를 표현한다. 또한 Synthesis 는 시스템과 환경 사이의 게임으로 여겨 질 수 있으며 이때 주어진 GR(1)이 게임의 승리 조건이 된다. $\varphi_e \rightarrow \varphi_s$는 각각 다음과 같은 LTL 식으로 구성된다.

$$\varphi_e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e, \qquad \varphi_s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$$

환경과 시스템에서 φ_i^e, φ_i^s는 환경의 입력과 시스템의 초기값을 표현하며, φ_t^e, φ_t^s는 환경과 시스템간의 전이관계를 표현한다. 마지막으로 φ_g^e, φ_g^s는 환경을 위한 목적과 시스템에서 기대되는 목표를 표현한다.

LTL Synthesis 알고리즘은 논리식 φ을 입력 받은 후에, 먼저 주어진 논리식이 실현 가능한지를 검사한다. 만약 실현 가능하다면, 알고리즘은 컨트롤러를 구현하는 오토마타를 생성한다. 생성된 오토마타는 요구 사항을 만족하기 위해서 로봇을 제어하는 컨트롤러로서 오토마타 형식은 $A = (X, Y, Q, Q_0, \delta, \gamma)$ 이다. 여기서,

- X 는 입력(환경) 변수 집합
- Y 는 출력(시스템) 변수 집합
- $Q \subseteq \mathbb{N}$ 는 상태 집합
- $Q_0 \subseteq \mathbb{N}$ 는 초기 상태 집합
- $\delta: Q \times 2^X \rightarrow 2^Q$ 는 전이 관계
- $\gamma: Q \times 2^X \rightarrow 2^Y$ 는 출력(상태 라벨링) 함수이다.

생성된 오토마타는 시스템이 요구된 행위를 만족하기 위해서 따라야 하는 전략을 구현해 놓은 것이다. 입력은 환경의 요구 사항인 φ_e를 만족하는 시퀀스 $X_0, X_1, \cdots, X_j \in 2^X$ 이다. 이러한 입력 하에서 이 오토마타의 실행(run)은 상태들의 시퀀스 $\sigma = q_0 q_1 q_2 \cdots$ 이다. 이 시퀀스는 초기 상태 $q_0 \in Q_0$ 에서 시작하며, 시퀀스의 모든 위치인 $j \geq 0$에서 $q_{j+1} \in \delta(q_j, X_j)$ 이다. 이다. 실행 σ 의 해석 또는 의미는 $Y_0, Y_1, \cdots$이다. 이 때 $Y_i = \gamma(q_i)$로서 실행 σ 의 i 번째 상태에 배정된 라벨(출력 또는 반응) 값이다. 이 라벨들의 시퀀스가 시스템이 따라야 할 경로 및 반응이다. 앞에서 언급한 바와 같이 환경에서 명시된 조건을 위반하는 입력 시퀀스가 주어진다면, 다시 말해서 φ_e의 어떤 부분이라도 위반하는 입력 시퀀스가 있다면, 생성된 오토마타는 더 이상 적절하지 않으며 시스템의 정확한 행위를 보장할 수 없다.

3. 사례 연구

본 장에서는 LTL Synthesis 를 이용하여 요구사항을 만족하는 오토마타를 생성하여 실제 로봇에 적용한다. 먼저, 시스템의 요구사항 또는 명세를 분석하여 GR(1)형식으로 변환한다. 그 후 Synthesis 알고리즘을 통하여 요구사항을 만족하는 오토마타를 생성 후, 생성된 오토마타를 실행한다.

실험 환경으로는 4 개의 영역이 가진 맵 환경과 실험 로봇으로는 레고 마인드 스톰 NXT2.0 이다. 로봇의 요구사항은 다음과 같다.

- 요구사항 1: 로봇은 r_1, r_2, r_3, r_4 지역을 순차적으로 방문하고, r_1, r_3 지역에서 $help$를 만나면 로봇은 정지하고 $alarm$을 울리며, r_2, r_4 지역에서 $fire$을 만나면 로봇은 정지하고 $beep$을 울린다.

로봇은 각 영역을 순찰하되 r_1, r_2, r_3, r_4 순으로 방문한다. 만약 r_1, r_3 영역에서 $help$를 만나면 거기에 멈춰서 $alarm$을 울린다. 만약 r_2, r_4 영역에서 $fire$를 만나면 거기에 멈춰서 $beep$를 울린다. 어느 영역에서나, 위급 상황을 나타내는 $help$가 해지되면 순찰을 재개한다.

(a) 맵 환경


(b) 실험 로봇

(그림 2) 실험 환경

로봇의 요구 사항을 전 장에서 소개한 LTL Synthesis 기법을 적용하기 위해서는 요구사항을 GR(1)형식으로 표현하여야 한다. 다음은 첫 번째 요구사항을 GR(1)형식으로 표현하는 과정을 설명한다.

우선 로봇이 환경으로부터 알아야 하는 정보로는 로봇이 4 개의 지역을 순찰 중 화재를 발견했음을 나타내는 $fire$ 와 구조 신호를 발견했음을 나타내는 $help$ 이다. 로봇의 환경 요구 사항을 나타내면 다음과 같다.

$$(\neg fire \wedge \neg help) \quad \cdots\cdots (1)$$

$$\Box\left(\neg(r_2 \vee r_4) \rightarrow \neg\bigcirc fire\right) \quad \cdots\cdots (2)$$

$$\Box\left(\neg(r_1 \vee r_3) \rightarrow \neg\bigcirc help\right) \quad \cdots\cdots (3)$$

$$\Box\left(\neg\left(\bigcirc help \wedge \bigcirc fire\right)\right) \quad \cdots\cdots (4)$$

(1)은 환경의 초기 조건으로 로봇이 화재와 구조 신호를 발견 못 했음을 나타낸다. (2), (3), (4)는 환경의 전이 조건으로 (2), (3)은 r_2, r_4이외의 영역에서는 화재를 발견하지 않는다는 것과 r_1, r_3이외의 영역에서는 구조 신호를 발견하지 않는 다는 것을 나타

내며, (4)은 화재와 구조 신호를 동시에 발견하지 못한다는 것을 나타낸다.

반면에 다음 아래는 로봇의 시스템 요구사항은 다음과 같이 나타낼 수 있다.

$$(\neg beep \wedge \neg alarm) \cdots (5)$$

$$(r_1 \wedge \neg r_2 \wedge \neg r_3 \wedge \neg r_4) \cdots (6)$$

$$\bigwedge_{i=1}^{4} \square(r_i \rightarrow r_{(i+1) mod 4}) \cdots (7)$$

$$\square \left(\begin{array}{l} \left(r_1 \bigwedge_{i \neq 1} \neg \bigcirc r_i \right) \\ \vee \left(r_2 \bigwedge_{i \neq 2} \neg \bigcirc r_i \right) \\ \vee \left(r_3 \bigwedge_{i \neq 3} \neg \bigcirc r_i \right) \\ \vee \left(r_4 \bigwedge_{i \neq 4} \neg \bigcirc r_i \right) \end{array} \right) \cdots (8)$$

$$\square \left(\neg \bigcirc fire \rightarrow \neg \bigcirc beep \right) \cdots (9)$$

$$\square \left(\neg \bigcirc help \rightarrow \neg \bigcirc alarm \right) \cdots (10)$$

$$\square \left(\bigcirc fire \wedge (r_2 \vee r_4) \rightarrow \bigcirc beep \right) \cdots (11)$$

$$\square \left(\bigcirc help \wedge (r_1 \vee r_3) \rightarrow \bigcirc alarm \right) \cdots (12)$$

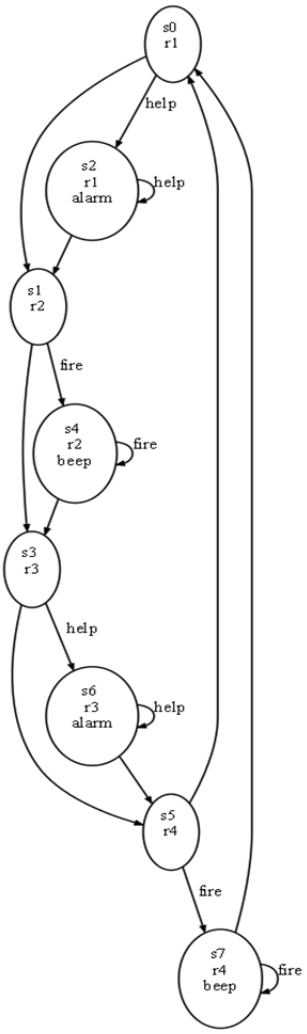
$$\square \left(\bigcirc beep \rightarrow \bigwedge_{i \in 2,4} \left(\bigcirc r_i \leftrightarrow r_i \right) \right) \cdots (13)$$

$$\square \left(\bigcirc alarm \rightarrow \bigwedge_{i \in 1,3} \left(\bigcirc r_i \leftrightarrow r_i \right) \right) \cdots (14)$$

$$\bigwedge_{i \in 1,2,3,4} \square \diamond (r_i \vee fire \vee help) \cdots (15)$$

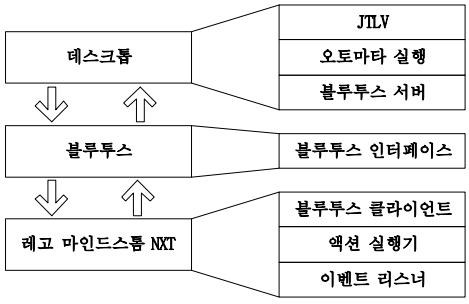
(5), (6)은 시스템의 초기 조건으로 $beep$와 $alarm$이 발생하지 않는다는 것과 로봇은 r_1에서 시작한다는 것이다. (7)부터 (14)은 시스템의 전이 조건을 나타낸다. (7)는 로봇이 영역에서 영역으로 이동관계를 나타낸다. (8)은 로봇은 항상 하나의 영역에만 위치함을 나타낸다. (9), (10)은 로봇이 화재를 발견하지 않으면 $beep$를 발생하지 않는 것과 구조 신호를 발견하지 않으면 $help$을 발생하지 않는 것을 나타낸다. (11), (12)는 로봇이 r_2, r_4에 있으면서 화재를 발견하면 로봇은 $beep$를 발생 해야 함과 로봇이 r_1, r_3에 있으면서 구조 신호를 발견하면 로봇은 $alarm$를 발생 해야 하는 것을 나타낸다. (13), (14)는 $beep$와 $alarm$이 발생되면 로봇은 그 자리에서 머물러야 함을 나타내며, 마지막 (15)은 시스템의 목표 조건으로 로봇이 화재 또는 구조 신호를 발견하지 않으면 계속해서 r_1, r_2, r_3, r_4을 방문해야 함을 나타낸다.

로봇의 행위를 명세한 논리식을 JTLV 를 통하여 실현 가능한지를 검사한다. 이때 만약 실현 가능하다면 로봇의 행위를 명세한 논리식에 해당하는 오토마타가 생성된다. 아래 그림 3 은 요구사항에 해당하는 오토마타이다. 오토마타를 살펴보면 각 요구사항에 부합한다는 것을 알 수 있다. 예를 들어 요구사항 1 의 경우 지역 r_1, r_3에서 $help$를 만나면 $alarm$을 발생하며, r_2, r_4에서 $fire$를 만나면 $beep$을 발생함을 알 수 있다.



(그림 3) LTL Synthesis 통하여 생성된 오토마타

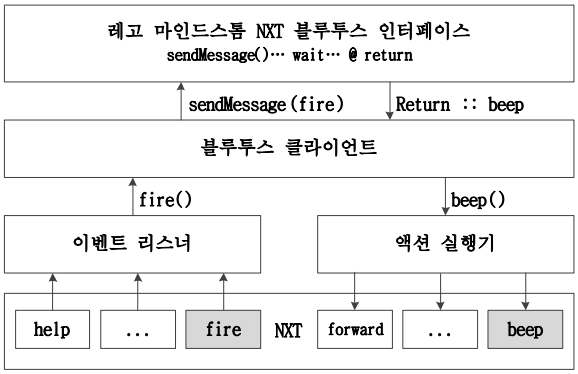
위 그림 3 와 같이 생성된 오토마타를 로봇에 적용하기 위하여 아래 그림 4 와 같이 로봇을 제어 할 수 있는 시스템을 구성하였다.



(그림 4) 로봇 제어 시스템 구성

로봇 제어 시스템은 크게 데스크톱, 블루투스, 레고마인드스톰 NXT 로 구성이 된다. 데스크톱은 JTLV 에 GR(1)형식의 LTL 파일을 입력으로 받아들여 실현 가능 여부를 판단 하여 실현 가능하다면 오토마타를 생성한다. 그리고 생성된 오토마타를 실행하는 부분과 로봇과 상호작용을 하기 위한 블루투스 서버로 구성된다. 블루투스는 로봇이 환경으로부터 입력 받는 센서 값을 데스크톱으로 전송하고, 로봇의 행위를 제어하기 위한 명령을 로봇으로 전송하는 역할을 하게 된다. 마지막으로 NXT 에서는 환경으로부터 입력 받

는 센서 값을 인식하기 위한 이벤트 리스너와 데스크톱으로부터 전송되는 로봇의 액션을 수행하기 위한 액션 실행기, 데스크톱과 상호작용을 하기 위한 블루투스 클라이언트로 구성된다.



(그림 5) NXT 시스템 구성

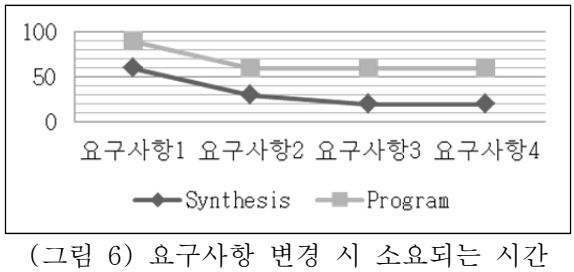
예를 들어 위 그림 6 과 같이 로봇이 r_2에서 $fire$를 발견하게 되면 이벤트 리스너에서는 $fire$에 해당하는 데이터를 블루투스 인터페이스 통해 데스크톱으로 전송하고, $fire$를 발견했을 때 수행하여야 하는 로봇의 다음 상태의 액션을 블루투스를 통하여 로봇에 전송한다. 그 후 로봇의 액션 실행기에서는 블루투스 통해 전송된 $beep$라는 액션을 수행한다.

4. 적용 및 결과

이번 장에서는 위의 3 장에서 제안한 일련의 과정에 대하여 3 개의 요구사항을 적용 하였다. 또한 LTL Synthesis 로부터 생성된 오토마타/시스템을 이용하여 구현된 로봇과 전통적인 로봇 프로그램으로부터 구현된 로봇간의 차이에 대하여 두 가지 항목으로 이야기 한다.

- 요구사항 2: 로봇은 r_1, r_2, r_3, r_4 지역을 순차적으로 방문하고, r_1, r_2, r_3 지역에서 $fire$를 만나면 로봇은 정지하고 $beep$을 울린다.
- 요구사항 3: 로봇은 r_4, r_3, r_2, r_1 지역을 순차적으로 방문하고, r_2, r_4 지역에서 $help$를 만나면 로봇은 정지하고 $alarm$을 울리며, r_1, r_3지역에서 $fire$을 만나면 로봇은 정지하고 $beep$을 울린다.
- 요구사항 4: 로봇은 r_4, r_3, r_2, r_1 지역을 순차적으로 방문하고, $r_4, r_3, r_2,$ 지역에서 $help$를 만나면 로봇은 정지하고 $alarm$을 울리며, r_1지역에서 $fire$을 만나면 로봇은 정지하고 $beep$을 울린다.

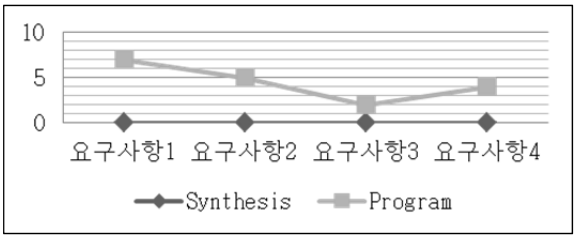
1) 로봇의 요구사항 변경 시 소요되는 시간



(그림 6) 요구사항 변경 시 소요되는 시간

그림 6 을 보듯이 로봇에 첫 번째 요구사항을 적용한 시간은 Synthesis 와 전통적인 프로그램 방법으로 구현된 시스템은 약 60 분과 90 분의 시간이 소요 되었다. 하지만 동일한 환경에서 요구사항을 수정하여 적용하는 경우 Synthesis 가 전통적인 프로그램 방법보다 적은 시간이 사용됨을 확인 할 수 있었다.

2) 오류의 수



(그림 7) 요구사항 변경 시 발생하는 오류의 수

전통적인 프로그램으로 구현된 시스템에서는 그림 7 을 보듯이 2 개에서 7 개의 오류가 잠재하고 있었다. 이러한 오류는 대부분 조건 문에서 잘못된 조건식에 의해 발생된 오류였다. 하지만 Synthesis 로부터 생성된 시스템에서는 요구사항이 올바르게 명세가 된 경우에는 오류를 발견 할 수가 없었다.

5. 결론

본 논문에서는 정형기법 중 하나인 LTL Synthesis 알고리즘을 이용하여 정확한 컨트롤러를 개발하는 방법에 대하여 제안하였다. 또한 4 가지 요구사항을 변경하면서 Synthesis 와 전통적인 프로그램으로 구현된 시스템 간의 차이를 비교 하였을 때 다음과 같은 놀라운 결과를 얻을 수 있었다.

제안방법을 이용하여 개발 할 경우 전통적인 프로그램 기법 보다 요구되는 소요 시간 및 오류의 수가 현저히 줄어듦을 보일 수 있었으며, 또한 컨트롤러의 정확성을 확인하기 위한 테스팅 또는 검증 등의 시간이 소요되지 않았음 에도 불구하고 요구사항에 올바른 행위를 하는 컨트롤러를 구축할 수가 있었다.

하지만 이러한 정형기법은 개발자가 쉽게 접근하기 어려운 문제점을 가지고 있다. 따라서 향후 연구로는 논문에서 제안한 LTL Synthesis 를 보다 쉽게 접근하기 위한 요구사항 명세 방안과, 멀티 로봇 적용 방안에 대하여 연구를 진행 할 것이다.

참고문헌

[1] H. Kress-Gazit, T.Wongpiromsarn and U.Topcu, “Correct, Reactive Robot Controlfrom Abstraction and Temporal Logic Specifications,“ IEEE Robotics andAutomation, Vol.18, No.3, pp.65-74, 2011.

[2] Nir Piterman, Amir Pnuli , and Yaniv Sa’ar. ”Synthesis of Reactive Designs”, In Proc, 7th International conference on Verification, Model Checking and Abstract interpretation, 2006

[3] 권령구, 권기현, “.LTL Synthesis 를 통한 반응형 시스템의 신뢰성 향상”, 한국정보과학회정보과학회지, 제 30 권 제 2 호, pp.24-30, 2012.

[4] http://jtlv.ysaar.net/ “A Framework for Developing Temporal Verification Algorithm.”