HyTech 를 이용한 ECML 모델의 검증 및 분석

윤상현 [1], 조재연 [2], 전인걸 [3], 유준범 [1]

[1] 건국대학교 컴퓨터공학부
서울 광진구 화양동 1 번지
{pctkdgus, jbyoo}@konkuk.ac.kr

[2] 한양대학교
서울 성동구 왕십리로 222
{jojaeyeon.kr}@gmail.com

[3] 한국전자통신연구원
대전 유성구 가정로 218 번지
{igchun}@etri.re.kr

요약: ECML 은 한국전자통신연구원에서 제안한 하이브리드 시스템 명세를 위한 언어이며 DEV&DESS 를 기반으로 하고 있다. ECML 은 모델링 및 시뮬레이션 기능을 지원하지만 모델체킹을 추가적으로 지원할 필요가 있다. ECML 은 모델링 성능이 높기 때문에 직접적으로 모델체킹하는 것은 어려운 문제가 있어, 기존의 하이브리드 검증을 위한 모델체커에 맞는 모델로 변환하여 검증하는 방법을 생각해 볼 수 있다. 본 논문에서는 ECML 을 선형 하이브리드 오토마타로 변환한 과정과, 모델체커인 HyTech 를 이용하여 검증한 결과를 보인다. 또한, HyTech 는 기본적으로 GUI 가 제공되지 않는데 검증 결과를 분석하여 사용자가 시각적으로 이해할 수 있도록 도와주는 도구인 HyTechAnalyzer 에 대해 소개한다.

핵심어: ECML, HyTech, 하이브리드 시스템, 모델체킹, 선형 하이브리드 오토마타, HyTechAnalyzer

1. 서론

하이브리드 시스템은 자동차, 항공, 군사 방어 시스템 등의 모델링 및 시뮬레이션에 사용되는 시스템으로서, 연속 시스템(continuous system)과 이산 시스템(discrete system)으로 구성된 동적 시스템이다. 이산 시스템은 주로 하이브리드 시스템의 작동 모드(operation mode)를 모델링하는데 사용되며, 연속 시스템은 하이브리드 시스템 내부 간의, 혹은 외부와의 물리적인 상호작용을 모델링하는데 사용된다. 하이브리드 시스템을 정형 명세하기 위해 하이브리드 오토마타가 [1] 제안되었으며 제약사항을 가진 하이브리드 오토마타인 선형 하이브리드 오토마타 (linear hybrid automata)를 [2] 검증하기 위한 모델체커인 HyTech [3], d/dt [4], PHAVer [5], SpaceEX [6]등이 제안되고 사용되어 왔다.

ECML (ETRI CPS Modeling Language)은 한국전자통신연구원(ETRI)에서 제안한 하이브리드 시스템의 명세를 위한 언어이며 DEV&DESS [7] 형식론(formalism)을 확장한 언어이다. ECML 은 모델링 및 시뮬레이션을 지원하지만 도달 가능성(reachability)과 안전성(safety)과 같은 요구사항에 대한 분석을 하기 위해서는 모델체킹 [8]을 추가로 적용할 필요가 있다. 모델체킹을 하기 위해서는 대상 모델의 모든 변수가 가지는 상태공간을 자동으로 계산하는 과정이 필요하다. 그러나 비선형인 변수를 포함한 ECML 을 직접적으로 모델체킹 하면 실수(real value)로 표현되는 연속 변수에 의해 무한한 상태 공간을 가지는 등 계산이 어려운 문제가 있다 [9]. 따라서 제약사항을 가진 선형 하이브리드 오토마타로 변환하여 모델체킹하는 방법을 생각해 볼 수 있다.

본 논문에서는 ECML 로 명세 된 모델을 모델체커인 HyTech 를 이용하여 모델체킹하는 과정에 대해 설명한다. ECML 로 명세된 모델을 변환 규칙에 따라 선형 하이브리드 오토마타로 변환한다. 그리고 이를 다시 HyTech 의 코드 형태로 만들어 HyTech 에 입력하고 검증 속성을 추가하여 모델체킹을 수행한다. HyTech 는 기본적으로 그래피컬 인터페이스를 제공하지 않아 검증 결과를 분석하기 어려운 점이 있다. 논문에서는 HyTech 의 모델체킹 결과를 분석하여 사용자가 시각적으로 검증 과정을 이해할 수 있도록 도와주는 도구인 HyTechAnalyzer 를 제안하고, 이를 통해 검증 결과를 분석한다.

본 논문의 구성은 다음과 같다. 2 장에서는 논문의 배경지식인 ECML 과 HyTech 와 HyTech 의 입력 모델인 선형 하이브리드 오토마타에 대해서 소개한다. 3 장에서는 2 장에서 소개한 정의를 바탕으로 ECML 을 선형 하이브리드 오토마타로 변환하는 규칙을 설명한다. 4 장에서는 간단한 자동차 예제를 3 장에서 소개한 변환 규칙을 사용하여 변환된 선형 하이브리드 오토마타를 HyTech 에 입력하여 검증한 결과를 보이고, 이를 시각화 해주는 지원도구인 HyTechAnalyzer 를 소개한다. 5 장에서는 논문의 모델 변환 및 모델체킹 과정에 대한 평가를 하고 결론을 내린다.

2. 배경 지식

2.1 ECML

ECML 은 구조 모델(structural model)과 행위 모델(behavioral model)로 구성되어 있다. 그림 1 은 BarrelFiller 모델을 ECML 의 구조 모델로 명세한 것을 보여 준다. 전체 시스템은 하나 이상의 구조 모델로 구성되어 있으며, 구조 모델간에 계층적인 관계를 가지고 있어 전체 모델을 캡슐화(encapsulation) 및 재사용을 지원할 수 있게 되어 있다. 각각의 구조 모델은 포트를 가지고 다른 구조 모델이나 시뮬레이션을 위한 환경모델과 입력 값과 출력 값을 주고 받을 수 있다. ECML 모델의 입출력에 '*[A], [E]*'와 같은 갱신 명세자(update specifier)가 있어 변수가 각각 연속적으로, 이산사건이 발생할 때 값이 갱신된다는 것을 나타내고 있으며, 하나의 구조 모델은 하나 이상의 행위 모델(예: *CBM BarrelFiller*)로 구성될 수 있다. 행위 모델간에도 포트를 통한 입출력 값 전달이 가능하다.

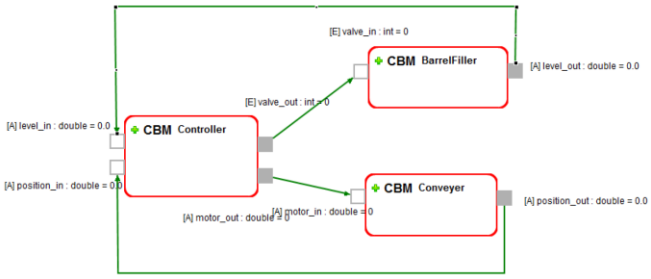


그림 1 ECML 구조 모델의 예 (BarrelFiller 모델)

행위 모델은 구조 모델의 내부적인 상태를 명세하는 데 사용된다. **그림 2** 는 BarrelFiller 를 표현한 행위 모델을 보여주고 있다. BarrelFiller 모델은 '*stop, filling, stopping*'이라는 phase 로 구성되어 있는데 '*stop*'은 완전히 멈춘 상태를 나타내고 '*filling*'은 barrel 에 물을 채우는 상태를, '*stopping*'은 barrel 의 내용물이 일정 수치가 되면 밸브를 잠가서 물의 유입량이 서서히 줄어드는 상태를 나타낸다. 각 phase 에서는 연속 변수 값의 변화율을 정의할 수 있고, 전이 함수에 의해 phase 간의 전이가 발생한다. ECML 행위 모델에 대한 형식적 정의(formal definition)는 다음과 같다.

정의 1: $BM = (X, Y, S, Trans^E, Trans^S, Cond^S, Rate, Out^C, Out^D, Out^E, Out^S)$.

- X: 입력 변수의 집합. $X=X^C \times X^D \times X^E$
  - X^C: 연속 값을 가진 입력 변수들의 집합
  - X^D: 이산 값을 가진 입력 변수들의 집합
  - X^E: 이산 사건 입력 변수들의 집합
- Y: 출력 변수의 집합. $Y=Y^C \times Y^D \times Y^E$
  - Y^C: 연속 값을 가진 출력들의 집합
  - Y^D: 이산 값을 가진 출력들의 집합
  - Y^E: 이산 사건 출력들의 집합
- P: phase 의 집합, phase 마다 $s \in S$, $x \in X$ 에 대한 invariant condition 과 *Rate* 를 정의
- S: 상태의 집합. $S=P \times S^C \times S^D$
  - S^C 는 연속적인 상태들의 집합
  - S^D 는 이산적인 상태들의 집합
- $Trans^E$: $S \times X \rightarrow S$: 외부 사건 전이 (external event transition) 함수
- $Trans^S$: $S \times X^C \times X^D \rightarrow S$: 내부 상태 전이 (internal state transition) 함수
- $Cond^S$: $S \times X^C \times X^D \rightarrow Bool$: 내부 상태 전이 함수를 발생시키기 위한 조건 함수
- $Rate$: $S \times X^C \times X^D \rightarrow S^C$ 연속 변수의 변화율 함수
- Out^C: $S \times X^C \times X^D \rightarrow Y^C$: 연속 값의 출력 함수
- Out^D: $S \times X^C \times X^D \rightarrow Y^D$: 이산 값의 출력 함수
- Out^E: $S \times X^C \times X^D \rightarrow Y^E$: 외부 사건 전이에 대한 이산 사건 출력 함수
- Out^S: $S \times X \rightarrow Y^E$: 내부 상태 전이에 대한 이산 사건 출력 함수

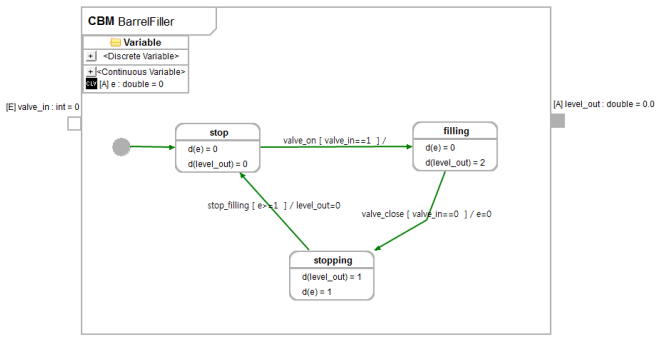


그림 2 ECML 행위 모델의 예 (*BarrelFiller*)

2.2 HyTech 와 선형 하이브리드 오토마타

HyTech [3]는 하이브리드 시스템을 대상으로 하는 모델체커이다. 하이브리드 시스템은 정형 모델인 하이브리드 오토마타로 명세하는데, HyTech 는 제한된 하이브리드 오토마타 모델인 선형 하이브리드 오토마타(Linear Hybrid Automata, LHA)와 안전성 요구사항을 입력으로 받고 이를 자동으로 검증한다. 하이브리드 시스템들은 하이브리드 오토마타로 표현할 수 있지만, 선형 하이브리드 오토마타로 표현되지 않는 경우가 많다. 이런 경우에는, approximation 이나 abstraction 을 통해 하이브리드 오토마타를 선형 하이브리드로 변환 후 검증해야 할 수도 있다 [9]. **그림 3** 은 ECML 행위 모델로 표현된 '*BarrelFiller*'를 선형 하이브리드 오토마타로 표현한 것이다. '*stop, filling, stopping*'의 control mode 들로 구성되어 있으며 jump condition 을 만족하면 control mode 간의 전이가 발생 한다. 선형 하이브리드 오토마타에 대한 형식적 정의는 다음과 같다.

정의 2: *HA={X, V, flow, inv, init, E, jump, Σ, syn}* [9]

- *X*: 변수의 집합, $X = X^C \times X^D$
  - X^C: 연속 값을 가진 변수들의 집합
  - X^D: 이산 값을 가진 변수들의 집합
- *V*: control mode 의 집합
- *flow*: flow condition, 각 control mode 에 할당, 해당 control mode 에서 변수의 변화율 함수
- *inv*: invariant condition, 각 control mode 에 할당, 해당 control mode 에 머무르기 위해서 변수의 값이 지켜야 할 조건
- *init*: initial condition, 각 control mode 에 할당, *init(v)=true* 일 때 해당 control mode 에 진입할 수 있음
- *E*: control switch 의 finite multiset
  - $e = (v,v')$: control switch
  - v: source control mode
  - v': target control mode
- *jump(e)*: jump condition, control switch 에 변수 값의 조건
- Σ: event 또는 synchronization label σ(e)의 집합, control switch 에 할당
- $syn(\sigma_{HA}(e_{HA}), \sigma_{HA'}(e_{HA'}))$: labeling function, $(\sigma_{HA}(e_{HA}) = \sigma_{HA'}(e_{HA'})) = true$ 이면 e_{HA} 와 $e_{HA'}$는 동시에 전이
  - HA, HA': 선형 하이브리드 오토마톤
  - $\Sigma_{HA} \subset HA, \Sigma_{HA'} \subset HA'$
  - $e_{HA} \in \Sigma_{HA}$, $e_{HA'} \in \Sigma_{HA'}$

정의 2 에서 *X* 의 경우, [9]에는 이산 변수와 연속 변수에 대한 구분을 따로 하지는 않았지만, HyTech 에서 이산 변수와 연속 변수를 구분할 수 있기 때문에 나누었다.

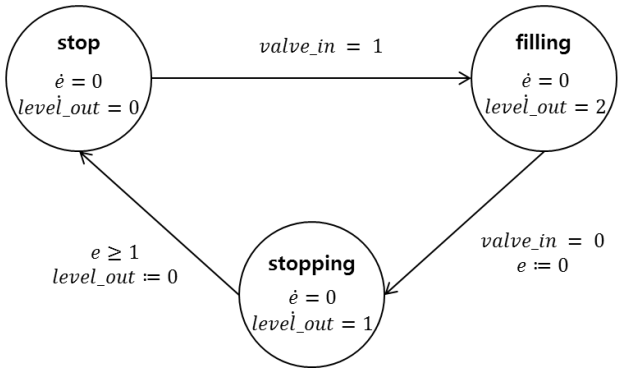


그림 3 LHA 로 명세한 *BarrelFiller* 모델

3. ECML 모델의 선형 하이브리드 오토마타 변환

이 장에서는 ECML 의 행위 모델을 선형 하이브리드 오토마타로 변환하기 위한 규칙을 설명한다. ECML 의 단일 모델을 대상으로 변환규칙을 소개한 후 여러 개의 단일 모델이 포트로 연결되어 있는 결합 모델 (coupled model)에 대한 변환 규칙에 대해서도 이야기 한다.

3.1 ECML 단일 모델

ECML 의 요소들을 선형 하이브리드 오토마타로 일대일로 대응시켜 가면서 변환 규칙을 설명한다. ECML 과 선형 하이브리드 오토마타는 몇 가지 의미상의 차이(semantic gap)가 있다. 대표적인 차이점은 두 가지를 들 수 있는데 첫째로, ECML 은 포트로 모델 외부에서 입력을 받아 올 수 있지만 선형 하이브리드 오토마타는 closed system 이기 때문에 외부입력이 없다. 둘째로, ECML 의 전이는 결정적(deterministic)이지만 선형 하이브리드 오토마타의 control switch 는 비결정적(non-deterministic)이라는 차이가 있다. 변환 규칙은 이러한 의미상의 차이들을 고려하여 정의 1 과 정의 2 의 용어들을 사용하며, 선행 연구인 [10]을 기반으로 하고 있다.

3.1.1 변수

ECML 의 입출력 변수들(*X, Y*)은 선형 하이브리드 오토마타의 변수들(X_{HA})로 변환할 수 있다. 두 형식론 모두 연속, 이산 변수를 지원하기 때문에 각각 대응 시킬 수 있다. ECML 의 X^E 의 경우, 이산 변수(X_{HA}^c)로 할당한다.

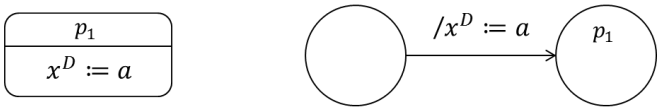
3.1.2 Phase

ECML 의 phase(*P*)는 선형 하이브리드 오토마타의 control mode(*V*)와 대응된다. $P \in \{p_1, p_2, \dots, p_i\}$ 이고 $V = \{v_1, v_2, \dots, v_i\}$ 이면, $v_1 := p_1, v_2 := p_2, \dots, v_i := p_i$ 와 같이 대응시킨다. 다만, ECML 은 Moore type 과 Mealy type 의 모델을 지원하는 반면에 선형 하이브리드 오토마타는 Mealy type 만 사용할 수 있는 문제점이 있다. Mealy type 의 phase 는 내부에 invariant, rate 만을 정의할 수 있고 전이의 출력 함수에 값을 정의할 수 있는 반면에, Moore type 의 phase 는 내부에 변수의 값을 할당할 수 있다는 차이점이 있다. Moore type 의 ECML 모델은 Mealy type 으로 일치시킨 후에 변환을 수행해야 한다.

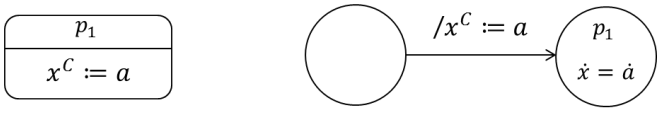
3.1.3 Moore type 의 ECML phase 의 변환

기본적으로 ECML 을 Mealy type 으로만 정의 하는 것이 가장 좋은 방법이지만 제약 사항을 두고 Moore type 의 모델을 선형 하이브리드 오토마타로 변환할 수 있는 방법이 있다. **그림 4** 는 Moore type 의 ECML phase 를 선형 하이브리드 오토마타로 변환한 것을 보여주고 있다. **그림 4** 의 '*a*'는 변수를 나타내는데, (a)에서처럼 이산 변수의 값을 Moore type 으로 정의하였을 경우 해당 control mode 로 들어오는

*jump(e)*에 값을 할당 하면 된다. (b)와 같이 연속 변수의 값을 Moore type 으로 정의하였을 경우, 해당 control mode 로 들어가는 *jump(e)* 에 값을 할당해주고 control mode 의 내부에 '$\dot{x} = \dot{a}$' 와 같이 연속 변수에 대한 flow 를 정의해 주면 된다. 그러나, control mode 내에 (b)와 같이 flow 에 flow 를 할당하는 표현은 HyTech 에서는 가능하지만, 검증 도구마다 지원여부가 결정되는 문제가 있다. 또한, 'a'가 외부 입력 변수일 경우, 이 방법으로는 변환이 어렵다.



(a) 이산 변수에 대한 Moore type의 변환



(b) 연속 변수에 대한 Moore type의 변환

그림 4 Moore type 의 ECML phase 의 변환

3.1.4 Initial condition

ECML 모델은 phase 에, 선형 하이브리드 오토마타는 control mode 에 모든 변수들의 상태를 initial condition 으로 가질 수 있다. 각각의 대응 되는 phase, control mode 마다 고려해야 한다.

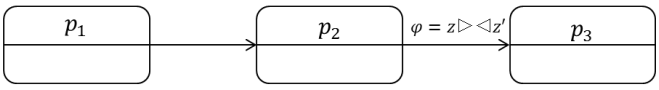
3.1.5 Rates

ECML 의 각 phase 에서 연속 입력, 연속 상태들에 대한 rate 는 선형 하이브리드 오토마타의 각 control mode 에서 flow 로 정의 될 수 있다. Rate 함수, $rate(s,X^C,X^D)$={s | $ds/dt=a(dx/dt)+b$, $s\in S^C$, $x\in X^C$, $\{a,b\}\in \mathbb{R}$}, 에 대하여 flow 함수는 $flow(v)$={s | $ds/dt=a(ds/dt)+b$, $s\in X_{HA}$, $\{a,b\}\in \mathbb{R}$}와 같이 정의할 수 있다.

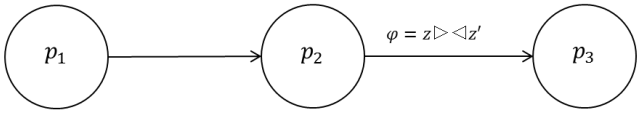
3.1.6 타입을 고려하지 않는 전이변환

ECML 모델의 모든 전이($Tran^E$, $Tran^S$)를 선형 하이브리드 오토마타의 control switch(E)로 변환 한다. 그리고 전이 조건 함수를 분석하여 jump condition 을 정의할 필요가 있다. 전이 조건은 단위 명제(atomic proposition)들의 conjunction 으로 표현될 수 있다. 단위 명제의 정의는 다음과 같다. $\triangleright\triangleleft = \{=, \leq, <\}$는 비교연산자들이고, $z := x \mid n \mid z + n \mid z \times n$ 는 x가 변수이고 n이 상수인 일차식인 단위명제 이며 $\varphi = z \triangleright\triangleleft z'$ 도 단위 명제라고 할 수 있다. $z \in X^C \cup X^D \cup S^C \cup S^D$ 이기 때문에, 연속/이산 값에 대한 차이와 비교연산자의 특징을 고려한 변환을 해야 한다.

그림 5 는 ECML 의 전이를 선형 하이브리드 오토마타로 변환한 ECML 의 전이를 보여주고 있다. ECML 의 $Tran^E(S,X)=p_2$ 에 대해, source phase P=p_1 이고, $cond= \varphi_1 \wedge \varphi_2 \wedge \ldots \varphi_m$, $Out^E=(S,X^C,X^D)$ ={$(n_1, n_2, \ldots n_m) \mid n_k \in \{X^C \cup X^D \cup S^C \cup S^D\}$, {i,k}∈R} 이면, 변환된 jump condition 은 $jump(p_1,p_2)=\{cond_1 \wedge y_1' = n_1 \wedge cond_2 \wedge y_2' = n_2 \wedge \ldots \wedge cond_i \wedge y_i' = n_i\}$ 이다.



(a) ECML 모델의 전이



(b) LHA로 변환된 전이

그림 5 ECML 전이 변환의 예

3.1.7 Urgent location

ECML 의 결정적인 전이를 선형 하이브리드 오토마타의 비결정적인 control switch 로 변환하기 위해서는 먼저 HyTech 의 urgent flag (*asap*)를 사용하는 것을 생각해 볼 수 있다. **그림 6** 에서 HyTech 모델의 behavior trajectory 는 $(l,x):(q_1,1) \rightarrow (q_2,1) \rightarrow (q_3,1)$ 이며, x=1 인 상태에서 q_2 에 넘어와서 jump condition 을 만족하여 바로 q_3 로 바뀌게 된다. 그러나, HyTech 에서는 **그림 7** 과 같이 x=0 인 상태에서 q_2 로 넘어오면, jump condition 을 만족하기 전에 flow 에 의해 연속 변수의 값이 증가한다. 이 경우에는 urgent flag 가 있더라도 control mode 의 전이가 바로 발생하지 않는 문제점이 있다. 따라서 urgent flag 대신 urgent location 을 추가하여 이를 해결하였다.

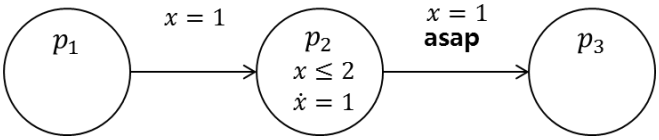


그림 6 urgent flag 를 이용해 표현한 HyTech 모델

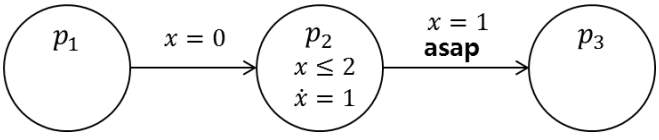


그림 7 control mode 의 전이가 일어나지 않는 예

Urgent location 은 앞서 언급한 문제를 해결하기 위해 기본적인 변환에서 사용되는 control mode 외에 추가적인 control mode 를 만드는 방법이다. **그림 8** 은 urgent location 의 예를 보여주고 있다. '*statndby_u*' control mode 는 urgent location 으로, 변환전의

ECML 에는 없던 변수인 'e'에 대한 flow 와 invariant 를 가지고 있다. 'e'는 urgent location 만을 위해 사용되는 변수로, 1 씩 증가하는데 ($\dot{e} = 1$) 반면 *inv* (*statndby_u*)는 0 이하이기 ($e \leq 0$) 때문에 jump condition 을 만족하면 바로 control switch 에 의해 전이가 발생한다. 본 논문에서는 urgent location 에 해당하는 control mode 는 이름의 끝에 '_u'를 붙여 구별하였다.

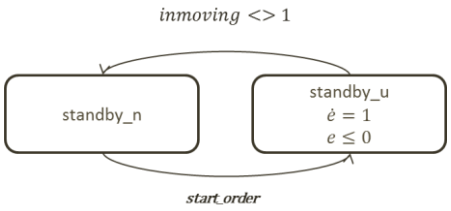


그림 8 urgent location 의 예

3.1.8 연속 변수만 갖는 전이

기본적으로 전이의 condition 단위 명제를 갖는 jump condition 과 그에 대한 negation 을 고려하여 변환한다. **그림 9** 와 같이 ECML 의 $Tran^S(q_1, x_C) \rightarrow q_2$ 에 대해 urgent location 인 q_1_u, negation 인 q_1_n, q_2 control mode 를 정의한다. 그리고 $Tran^S(q_1, x_C) \rightarrow q_2$ 를 만족하는 $jump(q_1\_u, q_2)$ 를 정의하고 negation 을 만족하는 jump condition 들도 새로 정의한다. 'e'는 S 에 포함되는 urgent location 을 위한 변수이기 때문에 $jump(q_1_u, q_2)$에서 초기화를 한다.

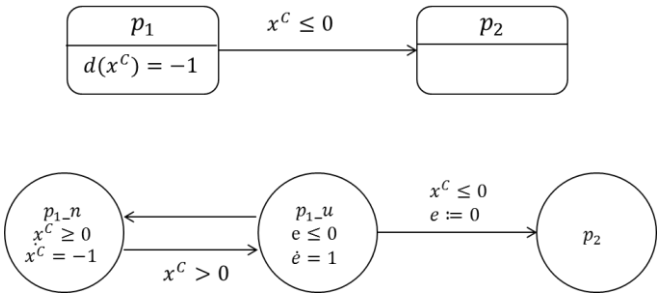


그림 9 연속 변수를 갖는 전이의 변환

연속 변수가 conjunction 으로 연결되어 있을 경우 ($\varphi = z \triangleright\triangleleft z'$)에는 전이의 조건 전체를 negation 을 하여 변환을 한다. **그림 10** 에서 $Trans^S(q_1, x_{C1}, x_{C2}) \rightarrow q_2$ 를 $e(q_1', q_2)$로 표현하였고 그 역은 $x_{C1} > 0 \vee x_{C2} > 2$ 로 두 개의 이산 변수를 갖는 전이를 변환하는 것과 같다. 따라서 $e(q_1', q_1''')$, $e(q_1', q_1'')$의 두 control switch 로 표현되었다. 그림 11 과 같은 경우에는 연속 변수가 disjunction 으로 연결되어 있는 경우라고 할 수 있는데, $TransS(q_1, x_{C1}) \rightarrow q_2$, $TransS(q_1, x_{C1}) \rightarrow q_3$ 의 두 전이를 $e(q_1', q_2)$, $e(q_1', q_3)$로 정의하고 역을 취하면 disjunction 의 역인 conjunction 을 갖는 하나의 control switch, $e(q_1', q_1'')$ 으로 정의 할 수 있다.

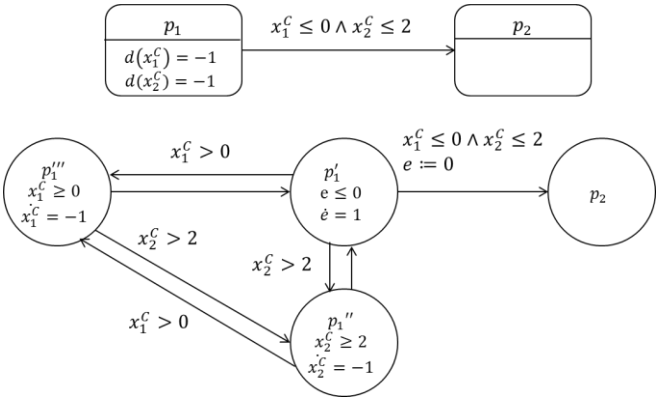


그림 10 연속 변수가 conjunction 으로 연결된 경우

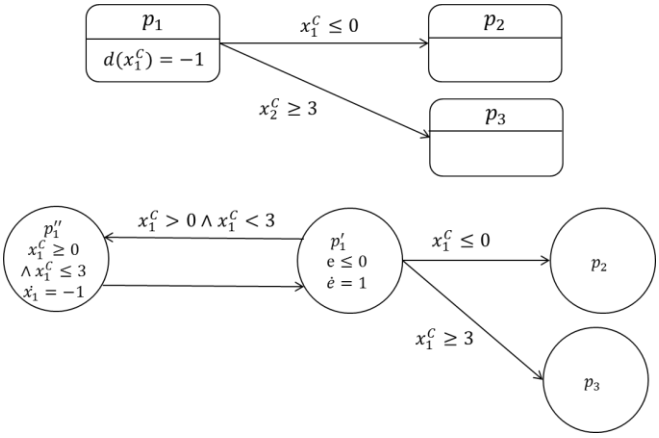


그림 11 연속 변수가 disjunction 으로 연결된 경우

3.1.9 연속, 이산이 혼용된 전이

이산 변수를 갖는 전이의 변환은 연속 변수를 갖는 전이와 크게 다르지 않지만 negation 을 한 control mode 에서 *flow* 를 정의하지 않는다는 점과 이산 변수에 대한 조건과 연속 변수에 대한 조건이 독립적으로 만족될 수 있다는 점을 고려한 변환이 필요하다. **그림 12** 에서 $e(q_1', q_1'')$, $e(q_1', q_1''')$가 개별적으로 조건을 만족하지 못하였을 때를, $e(q1''', q1')$, $e(q1'', q1')$이 조건을 만족하였을 때를 나타내고 있다.

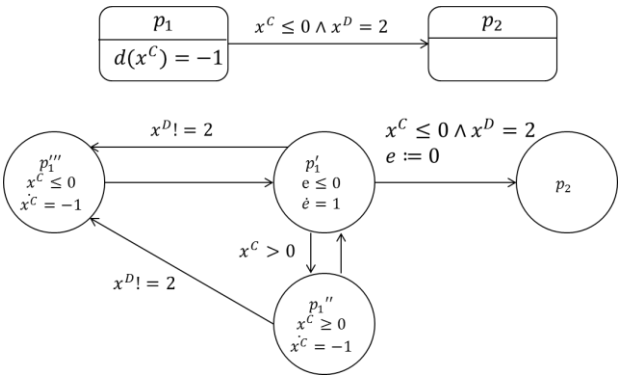
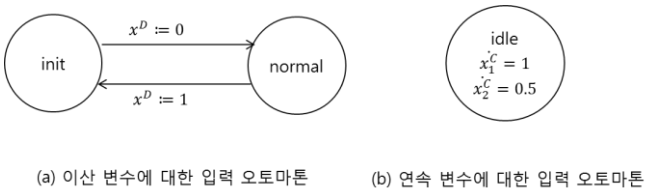


그림 12 이산 변수와 연속 변수가 혼용된 전이변환

3.1.10 입력 오토마타

선형 하이브리드 오토마타에는 모델 외부로의 입출력 기능이 없고, 변수 공유 및 control switch 에 할당된 event (synchronization label)을 통한 병렬 실행을 지원한다. 따라서 ECML 모델의 외부 입출력 기능을 구현하기 위해서는 공유 변수의 값을 할당해주는 오토마타가 추가로 필요하게 된다. **그림 13** (a)에서, 이산 변수에 0 과 1 을 무작위로 번갈아 가며 할당하도록 하였으며, 해당 변수를 다른 오토마톤에서 사용하면 외부 입력과 같이 사용할 수 있다.

외부 입력으로 들어오는 연속 변수에 대한 입력 오토마타는 *jump* 에 일정 값을 정의 하는 것이 아니라 control mode 에 *flow* 에 변화율을 정의하는 방식으로 생성한다는 점에서 이산 입력 오토마타 생성과 다르다. **그림 13** (b)에서 두 연속 변수는 특정한 조건 없이 각각 1 과 0.5 씩 증가하도록 되어 있다. 입력 오토마타를 생성하는 것은 검증 및 시뮬레이션을 하는 사람의 의도에 따라 갱신되는 시기 등을 모델링 해야 한다.



(a) 이산 변수에 대한 입력 오토마톤 (b) 연속 변수에 대한 입력 오토마톤

그림 13 입력 오토마타의 예

3.1.11 Execution order

선형 하이브리드 오토마타의 경우 한 jump condition 에서 여러 변수의 값을 할당해주면 각 할당이 병렬적으로 취급되지만, ECML 에서는 **그림 14** 와 같이 정의된 순서대로 순차적으로 실행되게 된다. 이 부분은 이러한 차이를 고려하여 ECML 모델을 모델링 할 때 제약사항으로 고려해야 할 부분이다.

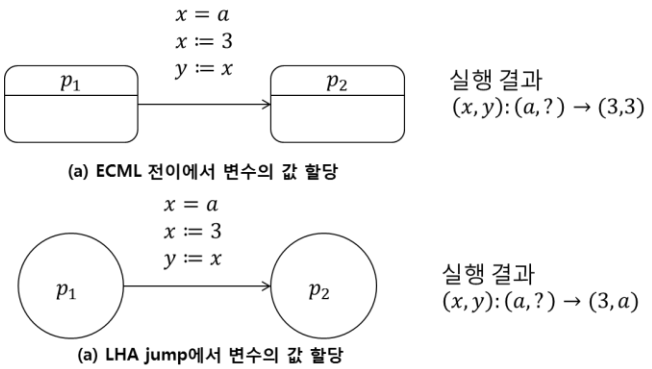


그림 14 Execution order 의 차이점

3.2 ECML 결합모델

ECML 로 명세 된 결합 모델은 포트간에 값을 할당해주는 형태로 되어 있다. 이를 선형 하이브리드 오토마타에서 표현하려면 입력 오토마타에서 했던 것과 유사하게 포트에 해당하는 공유 변수를 정의하는 것이 필요하다. 연속 변수의 경우에는 단순히 공유 변수를 지정하여 두 오토마타에서 공유 변수의 값을 할당 및 조건으로 사용 하면 되지만, 이산 변수의 경우 synchronization label 을 사용해야 한다. **그림 15** 에서 BM1 은 이산 값을 할당해주고, BM2 는 이산 값을 조건으로 사용하고 있다. BM1 에서 '$y_D\_x_D$'에 값이 할당되자 마자 BM2 의 '$y_D\_x_D$'에서도 값이 할당되어야 하고, 조건식이 사용되기 전에 일치해야 하므로 '*assign*'라는 synchronization label 을 사용하여 두 jump 가 동시에 발생 할 수 있도록 하였다.

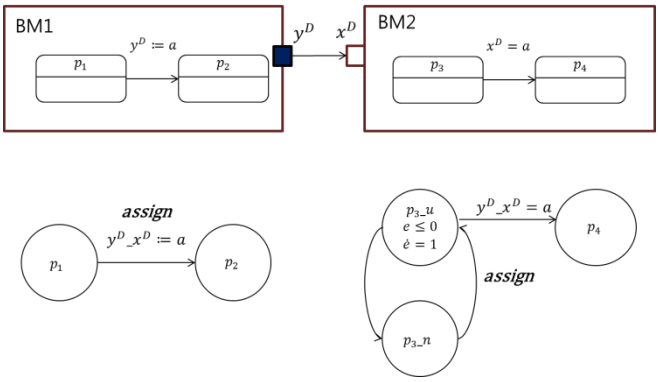


그림 15 이산 변수에 대한 결합 모델 변환

4. 사례연구

4.1 자동차 예제 변환

자동차 예제를 표현한 ECML 모델을 3 장의 변환 규칙에 따라 변환하고 이를 HyTech 를 이용하여 검증하였다. 자동차 모델은 현재 위치와 목표 위치를 모델의 외부에서 입력으로 받고 그에 따라 자동차가 진행하는 방향과 엔진의 출력을 외부 출력으로 내보내는 모델이며, 선형 하이브리드 오토마타로 변환할 수 있는 수준으로 변경하기 위해 모델을 간단하게 만들었다. ECML 모델은 **그림 16** 과 같이 '*control, thrustcontroller, Engine*'의 세 가지 행위 모델로 구성되어 있으며 각각의 내부는 그림 18~20 과 같다.

control 은 자동차 모델의 외부 입력을 받고 다른 행위 모델에게 명령을 내리는 모델이다. '*standby, moving*'이라는 두 phase 를 가지고 있으며, *moving* phase 에서 현재 좌표와 목표 좌표의 차이 벡터와 엔진 명령을 출력으로 보내준다. *Engine* 은 *control* 에서 들어온 '*ismoving*'의 값에 따라 엔진의 출력에 해당하는 '*Momentum*'의 값을 증가시키다 일정 수치에 다다르면 유지하거나 줄이는 행위를 하고, *thrustcontroller* 는 *control* 에서 들어온 좌표 벡터를 가지고 자동차가

진행할 방향(*heading*)을 결정하여 출력으로 내보낸다. 현재 모델에서 자동차는 네 방향, 즉 45°, 135°, 225°, 315°로만 방향을 바꾼다. 각 행위 모델을 변환한 선형 하이브리드 오토마타는 그림 21~23 과 같다.

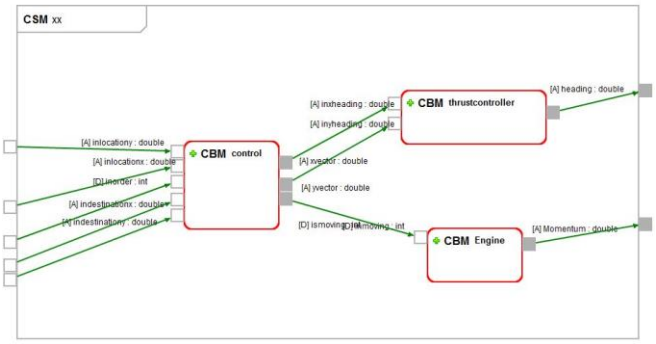


그림 16 ECML 구조모델로 표현한 자동차 모델

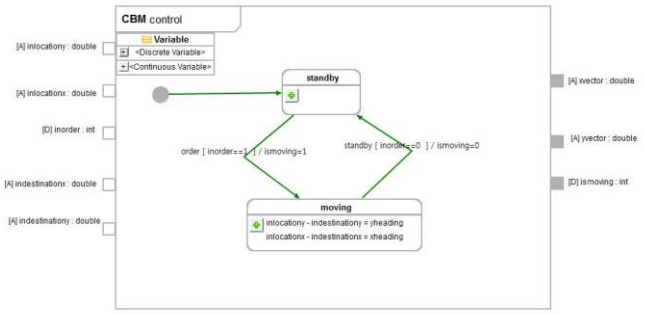


그림 17 ECML 로 표현한 자동차 모델(control)

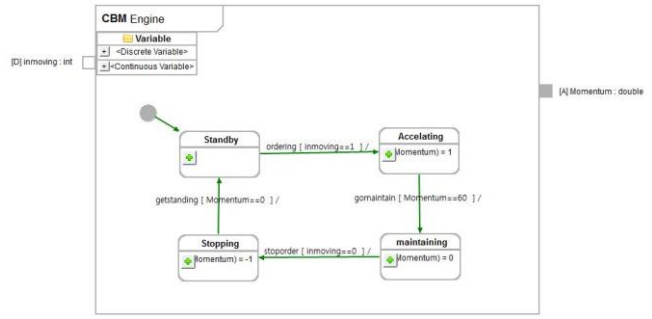


그림 18 ECML 로 표현한 자동차 모델(*Engine*)

변환된 선형 하이브리드 오토마타의 jump condition 에서 '*inorder*<>*1*'와 같이 사용된 단위 명제는 '*inoder*! =*1*'을 표현한 것이며 HyTech 의 문법적 특징을 반영한 것이다. **그림 20** 과 **그림 21** 에서 '*start_order*'와 '*stop_order*'가 event 로 같이 할당되어 각각 동시에 jump 가 발생하도록 되어 있다. 변환된 선형 하이브리드 오토마타를 HyTech 의 입력 코드로 변환한 것은 **그림 23** 와 같다.

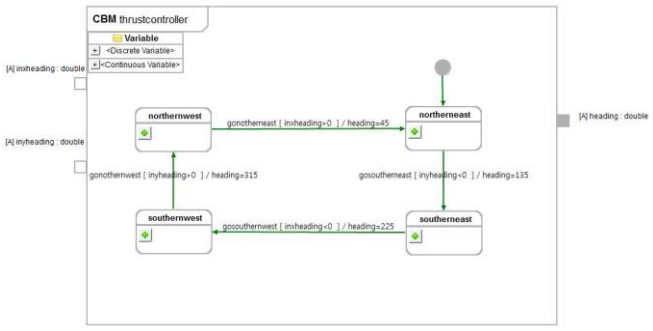


그림 19 ECML 로 표현한 자동차 모델(thrustcontroller)

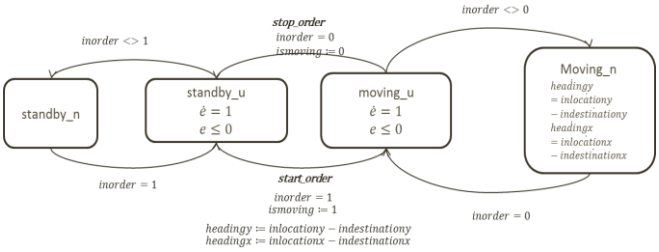


그림 20 LHA 로 변환한 자동차 모델(control)

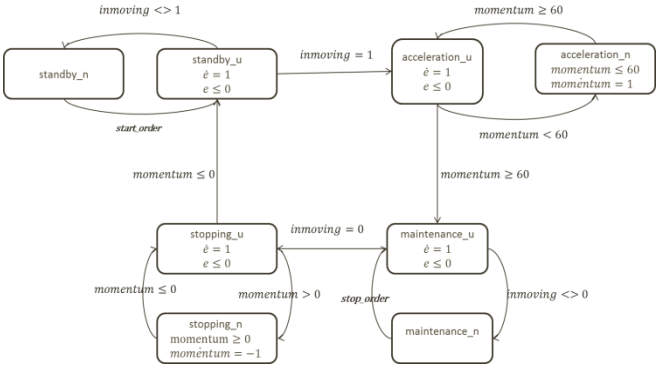


그림 21 LHA 로 변환한 자동차 모델(Engine)

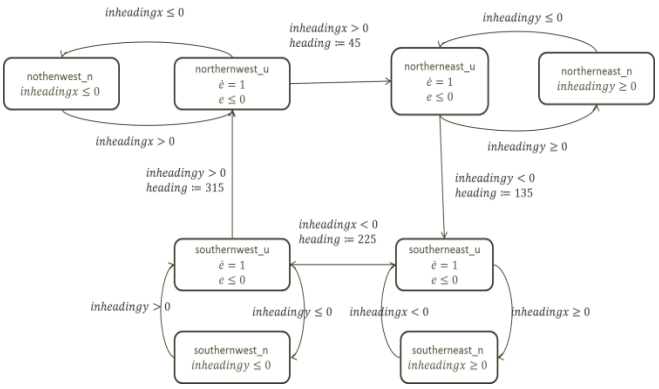


그림 22 LHA 로 변환한 자동차 모델(*thrustcontrol*)

```
--input analog variable : locationx, locationy, destinationx, destinationy
--local analog variable : u_control
--output analog variable : headingx, headingy
--initial condition : loc[control] = standby_u &
automaton control
synclabs:start_order, stop_order;
initially standby_u & u_control=0 &
    in_order=0 & heading_x=0 & heading_y=0 ;
loc standby_u :
    while u_control<=0
        wait{du_control=1, dheading_x = 0, dheading_y=0}
        when in_order>1 do {} goto standby_n;
        when in_order<1 do {} goto standby_n;
        when in_order=1 do
            {heading_x'=destination_x - location_x,
               heading_y'=destination_y - location_y}
            goto moving_u;
loc standby_n : while True wait{du_control=0, dheading_x = 0, dheading_y=0}
        when in_order=1 sync start_order do {} goto standby_u;
loc moving_u : while True wait{du_control=1, dheading_x = 0, dheading_y=0}
        when in_order>0 do {} goto moving_n;
        when in_order<0 do {} goto moving_n;
        when in_order=0 sync stop_order do {moving'=0} goto standby_u;
loc moving_n : while True wait
        {du_control = 0,
        dheading_x = ddestination_x - dlocation_x,
        dheading_y = ddestination_y - dlocation_y}
        when in_order=0 do{} goto moving_u;
end
```

그림 23 HyTech 입력 코드로 변환한 LHA(일부)

4.2 HyTech Analyzer 를 이용한 분석

HyTech 는 검증 속성에 대하여 analyze 와 trace 기능을 제공한다. Analyze 기능은 검증 속성을 만족할 때 모델에 속해있는 모든 변수가 어느 location 에 속속 있을 때 값을 가지는 공간, 즉 region 이 어떻게 되는지를 보여주는 것이고 trace 검증 속성을 만족할 때 initial 상태에서 검증 속성을 만족하는 여러 경로 중에 한 경로를 선택하여 어떤 경로를 거치는지 보여주는 기능이다. 자동차 모델을 사용하여 analyze 와 trace 기능을 대해 검증을 수행하였다.

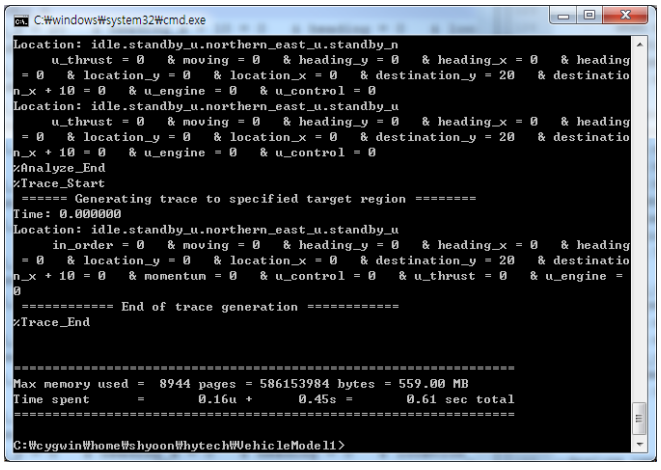


그림 24 HyTech 를 이용하여 검증한 결과

그림 24 는 검증된 결과를 보여 주고 있다. Analyze 기능으로 검증해 본 것은 initial 상태에서 도달할 수 있는 각 변수들(입력, 출력, 상태)의 모든 region 에 대해 확인한 것이고, trace 기능으로 확인해 본 것은 자동차의 진행 방향이 0° 를 향하는 경우에 대해 확인해 보았다. Trace 의 기능으로 확인해 본 것은 initial 상태에서만 만족되었기 때문에 검증 결과를 분석할 수 있지만, analyze 의 경우에는 복잡하여 결과를 한눈에 확인하기에 어렵다.

HyTechAnalyzer 는 eclipse plugin 으로 개발되었으며 HyTech 의 코드를 편집 및 HyTech 이용하여 검증하는 환경을 제공해 준다. 또한, trace 와 analye 기능을 이용한 검증 결과를 시각화하여 분석을 돕는 도구이다. 지원하는 기능에는 검증 결과를 표나 그래프 형태로 변환해주는 기능이 있다. **그림 25** 의 하단부는 analyze 기능을 이용한 검증 결과를 HyTechAnalyzer 가 RegionTreeView 기능을 이용하여 보여주고 있는데, tree 형태로 각 변수들이 어떤 조건을 가지고 어떤 region 을 갖는지를 볼 수 있다. Region 은 검증 속성을 만족하는 state (location, value set)의 연속으로 볼 수 있는데, 결과를 확인하고 싶은 location 을 선택하면 해당 location 에서 갖는 각 변수들의 값을 볼 수 있도록 계층적으로 구성되어 있다.

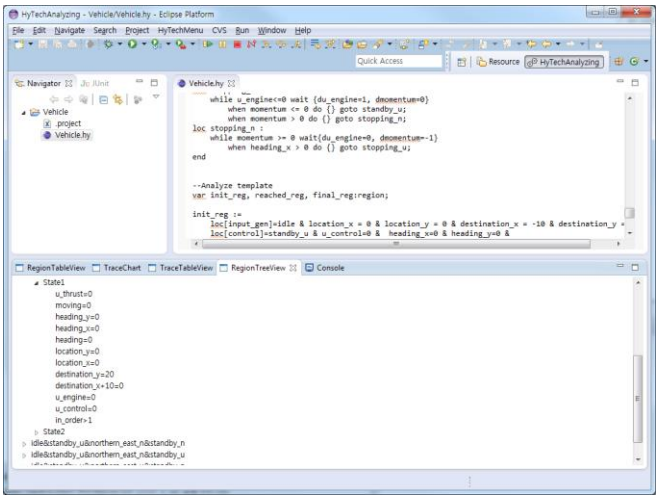


그림 25 HyTechAnalyzer 를 이용한 검증 결과 분석

5. 결 론

본 논문에서는 하이브리드 시스템 모델링 언어인 ECML 을 선형 하이브리드 오토마타로 변환하고 모델체커인 HyTech 를 이용하여 검증한 과정을 보였다. HyTech 를 이용한 ECML 모델 검증과정에서 선형 오토마타로 변환하는 규칙을 제안하였으며, HyTech 의 검증 결과를 시각적으로 분석할 수 있도록 도와주는 도구인 HyTechAnalyzer 를 소개하였다. 본 연구를 활용하기 위해서는 ECML 모델이 선형 오토마타로 변환할 수 있어야 한다는 제약사항이 있다. 이를 보완하기 위해 선형 오토마타를 입력으로 받는 모델체커인 HyTech 외의 다른 하이브리드 시스템 검증 모델체커를 이용하는 연구를 진행하고 있으며, 본 논문의 변환 과정을 자동화 하는 도구를 개발할 계

획이다.

Acknowledgement

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천기술개발사업의 일환으로 수행하였음. [10035708, 고신뢰자율제어 SW 를 위한 CPS(Cyber-Physical Systems) 핵심 기술 개발]

참고문헌

[1] R. Alur, C. Courcobetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, "The algorithmic analysis of hybrid systems", Theoretical computer science, Vol.138, No.1, pp.3-34, 1995

[2] R. Alur, T. A. Henzinger, P. H. Ho, "Automatic symbolic verification of embedded systems", IEEE Transctions on Software Engineering, Vol.22, pp.181-201, 1996

[3] T. A. Henzinger, P. H. Ho, H Wong-Toi, "A user guide to HyTech", Tools and Algorithms for the Construction and Analysis of Systems, LNCS, Vol.1019, pp.41-71,1995

[4] E. Asarin, T. Dang, O. Maler, "The d/dt Tool for verification of hybrid systems", Computer Aided Verification 2002, LNCS, Vol.2404, pp.365-770, 2002

[5] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech", Fifth International Workshop on Hybrid Systems: Computation and Control (HSCC), LNCS, Vol.3414, pp. 258-273, 2005

[6] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, "SpaceEX: Scalable verification of hybrid systems", CAV 2011, LNCS, Vol.6806, pp.379-395, 2011

[7] Bernard P. Zeigler, Herbert Praehofer, Tag-Gon Kim, "Theory of Modeling and Simulation, Second Edition", Academic Press, 2000

[8] E. M. Clarke, O. Grumberg, D. A. Peled, "Model Checking", MIT press, 1999

[9] T. A. Henzinger, P. H. Ho, H. Wong-Toi, "HyTech: A model checker for hybrid systems", Software Tools for Technology Transfer, Vol.1, pp.110-122, 1997

[10] J. Jo, J. Yoo, H. Choi, H. Y. Lee, W. Kim, "Translation from ECML to linear hybrid automata", Embedded and Multimedia Computing Technology and Service, LNCS, Vol.181, pp.293-300, 2012