Translation from ECML to Linear Hybrid Automata

Jaeyeon Jo[1], Junbeom Yoo[1], Han Choi[2], Sungdeok Cha[2],
Hae Young Lee[3], and Won-Tae Kim[3]

[1] Konkuk University, Seoul, Republic of Korea
{tm77,jbyoo}@konkuk.ac.kr
[2] Korea University, Seoul, Republic of Korea
{issubi,scha}@korea.ac.kr
[3] Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea
{haelee,wtkim}@etri.re.kr

Abstract. ECML (ETRI CPS Modeling Language) is a modeling formalism for hybrid systems, recently proposed by a research institute - ETRI in Korea. It extends a basic formalism DEV&DESS (Discrete EVent & Differential Equation System Specification) with various conveniences in modeling and simulation. The formal verification tool for ECML has not been provided yet. This paper proposes translation rules from ECML to linear hybrid automata which is an input front-end of HyTech. We can verify ECML models with the HyTech model checker.

Keywords: Hybrid System, Formal Verification, Translation, ECML, Linear Hybrid Automata.

1 Introduction

Hybrid system is a dynamical system whose behavior is a combination of continuous and discrete dynamics. The discrete parts naturally model modes of operation of system, while the continuous dynamic model physical interactions with themselves or environment, such as automotive controllers, avionic, defense modeling and simulation and medical equipment and controllers. Hybrid automata [1] has been proposed as a formal model for hybrid systems. Linear hybrid automata (LHA) [2] is restricted hybrid automata describing hybrid systems using linear dynamics. The examples of verification tools for linear hybrid automata are HyTech [3], d/dt [4], PHAver [5] and SpaceEx [6].

ECML [7] is an extension of the basic formalism DEV&DESS [8] with various conveniences in modeling and simulation. It was recently proposed by ETRI in Korea. It still needs algorithmic method for verifying ECML models against important properties such as safety and reachability efficiently. In the previous studies [10, 11], DEVS or DEV&DESS model are translated into other models for performing formal verification. This paper proposes translation rules from ECML to LHA, since LHA is a common front-end for formal verification tools such as HyTech. We can perform the HyTech verification on the ECML models. It is worth noting that semantic gap between ECML and LHA requires restricting the ECML's expressive power. We only consider the ECML models which can be translated into LHA. The paper is organized as follows. Section 2 introduces ECML and LHA briefly. Section 3 explains the translation algorithm from ECML to LHA. Section 4 concludes the paper with our plan of future work.

2 Background

2.1 ECML

ECML (ETRI CPS Modeling Language) is a modeling language for hybrid systems, recently proposed by ETRI (Electronics and Telecommunications Research Institute) in Korea. It extends the basic formalism DEV&DESS with various conveniences such as hierarchies and error modeling. ECML models have three different types of inputs/outputs: discrete event, discrete value, continuous value. The basic model of ECML is defined as

$$BM = < X, Y, S, Trans^E, Trans^S, Cond^S, Rate, Out^C, Out^D, Out^E, Out^S > \qquad (1)$$

Fig.1 shows an example of a *Barrelfiller* basic model(BM) written in ECML. The *Barrelfiller* model is filling system that fills liquid to a barrel. Formal definition of BM (1) is belows:

$X = \{On, Inflow\}$: a set of input variable. $X{=}X^C{\times}X^D{\times}X^E$
- *On* : a discrete input variable in X^D, switch phase.
- *inflow* : a continuous input variable in X^C. flow of level.
- X^C is set of discrete value inputs.
- X^D is set of discrete value inputs.
- X^E is set of discrete event inputs.

$Y = \{Barrel\}$: a set of output variable. $Y{=}\ Y^C{\times}Y^D{\times}Y^E$
- *Barrel* : a discrete output variable in Y^E.
- Y^C is set of continuous value outputs.
- Y^D is set of discrete value outputs.
- Y^E is set of discrete event outputs.

S = {Level} : a set of states, $S{=}P{\times}S^C{\times}S^D$.
- *Level* : a state variable in S^C , observes a level of liquid.
- S^C is set of continuous states.
- S^D is set of discrete states.

$P \in \{Ready, Filling\}$: the set of phases
- *Ready* : a phase in *P, Level* is not changing.
- *Filling* : a phase in *P, Level* is changing at rate of *Inflow*.

$Trans^E(p, Inflow, Level, On)$: external event transition function
- (*p=Ready*, On=true) : *p:=Filling*
- (*p=Filling*, On=false) : *p:=Ready*

$Trans^S(p, Inflow, Level, On)$: state transition function
- (*p=Filling*, $Level \geq 10$) : *p:=Filing*, *Level*:=0

Out^S(*p*, *Inflow, On*) : discrete event output function for internal state transitions

- (*p*=*Filling*, *Level*≥ 10) : *Barrel*:=1 : while value of *level* is changing 10 to 0, output value of barrel is assigned 1.

$Cond^S$(*p*, *Inflow, On*)=(*p*=*Filling*∧*Level*≥10) : state transition condition function

Rate(*p*, *Inflow, On*) : rate of change function

- (*p*=*Ready*) : *dLevel*/*dt*=0 : *Level* is not changing.
- (*p*=*Filling*) : *dLevel*/*dt*=*Inflow*; *Level* is changing at rate of *Inflow*.

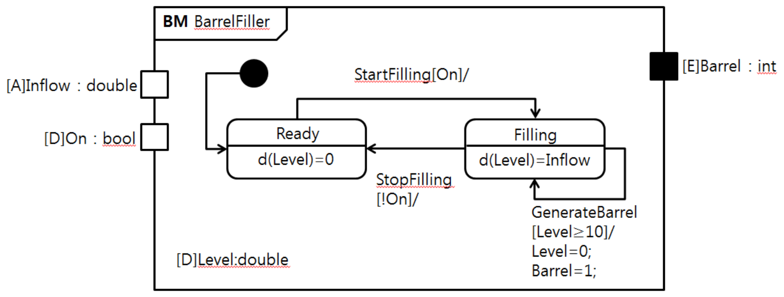


Fig. 1. A graphical representation of the *Barrelfiller* system

2.2 Linear Hybrid Automata

The linear hybrid automata [3] integrate discrete behavior of digital computer systems with continuous behavior of environment and hardware systems. Informally, it annotates finite state automaton with conditions on real-valued variables. The linear hybrid automata formalism is defined as follows [3]:

$$HA= <X, V, \text{flow}, \text{inv}, \text{init}, E, \text{jump}, \sum, \text{syn}> \quad (2)$$

Fig.2 is a LHA model of the *Barrelfiller*, whose behavior is semantically similar to Fig.1. Formal definition of linear hybrid automata (2) is belows:

X = {*On*, *Level*} : a variable set.

- *On* : a variable to switching locations.
- *Level* : a variable that show liquid level of barrel.

V = {*Ready*, *Filling*} : a control mode set.

- *Ready* : a control mode.
- *Filling* : a control mode to filling barrel.

flow(*Ready*) = *dLevel*/*dt*=*0* : flow condition.

flow(*Filling*) = *dLevel*/*dt*=1 : flow condition.

inv(*Filling*) = (*Level*≤10) : invariant condition, while control is in mode Filling, *Level* must be below 10.

init(*Ready*) = (*level*=0) : initial condition, control of HA may start in the control mode *Ready* when the *level*=0 is true.

init(*Filling*) = *false* : initial condition.

E={*(Ready, Filling), (Filling, Ready), (Filling, Filling)*} : a finite multi-set of control switches.

Jump(*Ready*, *Filling*)={*On*=1} : a jump condition.
Jump(*Filling*, *Ready*)={*On*=0} : a jump condition.
Jump(*Filling*, *Filling*)={*Level*=10 ∧ *Level'*=0} : a jump condition.

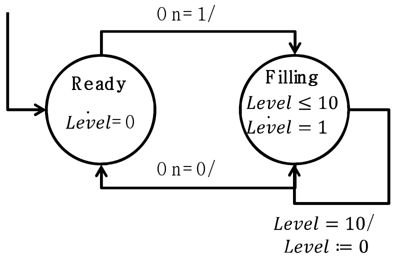


Fig. 2. A graphical representation of the *Barrelfiller* using linear hybrid automata

3 Translation from ECML to LHA

This section introduces translation rules from a single ECML model to linear hybrid automata. Coupled models are also considered briefly.

3.1 Single ECML Model

We explain the translation rules one-by-one, mapping one element in ECML (1) to one of LHA (2). They include variables, phases, initial translation, rate, transition, etc. We also had to solve several problems due to semantic gap between ECML and LHA, such as I/O structure and transition execution. This section includes our solution for the semantic gap, too.

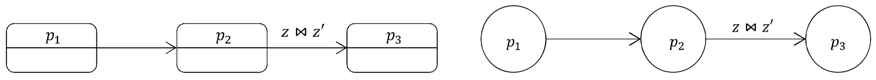
Variables. Every input/output variable and state variable in an ECML model is translated into a corresponding variable of linear hybrid automata. They both should have the same data type.

Phases. Each phase in ECML corresponds to control node in LHA. If $P \in \{p_1, p_2, \ldots, p_i\}$ is a set of phases, all control nodes v in modes $V \in \{v_1, v_2, \ldots, v_i\}$ has values such as $v_1 := p_1, v_2 := p_2, \ldots, v_i := p_i$.

Initial Condition. An ECML model has an initial condition consisting of a phase and a set of states of all variables, while LHA has a control node a set of states of all variables. All elements should coincide with each other.

Rates. The *rate* of each phase in ECML decides the *flow* of each control mode in LHA. We use the form of derivation of continuous variables in order to describe the *flow* of continuous state variables using continuous input variables. For a *rate* function $rate(S, X^C, X^D) = \{s \mid ds/dt=a(dx/dt)+b, s \in S^C, x \in X^C, \{a, b\} \subset R\}$, the *flow* function is $flow(p) = \{s \mid ds/dt=a(dx/dt)+b, s \in X_{HA}, \{a, b\} \subset R\}$. X_{HA} is equals to X in (2).

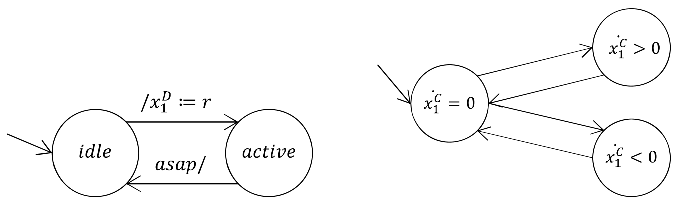
Transitions without Type Distinction. Translating transitions needs to analyze conditions which are conjunctions of atomic propositions. Atomic proposition is defined as follows. $\rhd\lhd = \{=, \leq, <\}$ are comparison operators. $z := x \mid n \mid z+n \mid z\times n$ belong to linear expressions if x is a variable and n is a constant. $\varphi = z\rhd\lhd z'$ is an atomic proposition too. For $Trans^E(S, X)$=p$_2$, $P=p_1$, $cond1=\varphi_1\wedge\varphi_2\wedge\ldots\wedge\varphi_m$, $z\in\{X^C\cup X^D\cup S^C\cup S^D\}$ and discrete event output function $Out^E(S, X^C, X^D)=\{(n_1, n_2, \ldots, n_m)\mid n_k\in\{X^C\cup X^D\cup S^C\cup S^D\}, \{i,k\}\in R\}$ in ECML, the translated *jump* condition in LHA is as follows. $(p_1, p_2)\in E$, $jump(p_1, p_2)=\{cond_1\wedge y_1'=n_1\wedge y_2'=n_2\wedge\ldots\wedge y_i'=n_i\}$. Fig.3 shows an example translating a transition in ECML to that on LHA.



(a) A transition in ECML (b) A translated transition in LHA

Fig. 3. An example of translating transitions of ECML into LHA

Generating Discrete Input Automata. LHA has no input/output structure. Therefore, we develop a discrete input automaton additionally to control discrete input variables in ECML. The behavior of discrete input automata is as follows. It assigns random values to discrete variables when there is no specific occurring time of external event, and executes external transitions. However, if an external event occurs, a corresponding external transition should be executed immediately. We use '*synchronization labels*' to precisely simulate this behavior in LHA. For example, Fig.4a is a discrete input automaton. The transition from *idle* to *active* is executed randomly and assigns any value to the variable x_1^D. In *active* state, it immediately transits to *idle*. If a '*synchronization label*' is used in the transition, it synchronizes with other transitions those uses '*synchronization label*' in LHA.



(a) A discrete input Automaton (b) A continuous input automaton

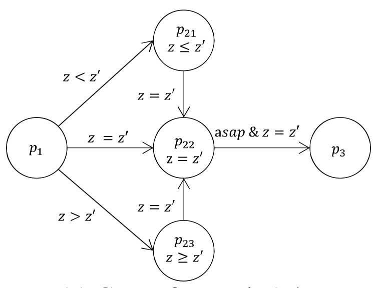
Fig. 4. Input control automata

Generating Continuous Input Automata. Continuous input automaton is related with X^C. Continuous input can be designed manually thorough considering trajectories, but we propose automatic generation of continuous inputs as shown in Fig.4b. The automaton shows that value of x^C changes while assigned with an unspecified value in unspecified time. A correspond hybrid automaton is generated from a continuous variable $x^C \in X^C$. One method to determine execution time is to use the *asap* condition. It is an urgent flag introduced in HyTech [9].

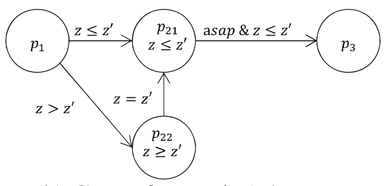
State Transitions. State transitions of ECML, which have no type distinction, are also translated into transitions of LHA. The condition for the state transitions are defined as $C=(\varphi_1^C \wedge \varphi_2^C \wedge \ldots \wedge \varphi_n^C \wedge \varphi_1^D \wedge \varphi_2^D \wedge \ldots \wedge \varphi_m^D)$, where φ^C is an atomic proposition for continuous terms while φ^D is for discrete terms. We also generated additional control modes using φ^C combinations as shown in Table 1. It shows translation rule about φ^C, an atomic proposition in state transition conditions. *Invariant* is an invariant condition for a control mode. *Cond*$_{\text{prev}}$ is a part of condition on a transition whose target control mode is the source of the state transition. *Cond$_{prev}$* is omitted if *Invariant* satisfies φ^C. *Cond$_{out}$* is a condition on the transition corresponding to the state transition. *Current* means elements changed after state transitions. *Negation* indicates generated elements for an alternative situation when invariant condition of source control mode from a state transition is *false* except boundary value.

Table 1. Translation rules for state transition

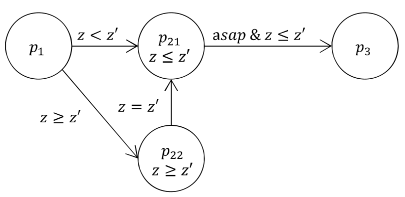
z⊳⊲z'	z=z'			z<z'			z≤z'		
	Cond$_{\text{prev}}$	*Invariant*	*Cond*$_{\text{out}}$	*Cond*$_{\text{prev}}$	*Invariant*	*Cond*$_{\text{out}}$	*Cond*$_{\text{prev}}$	*Invariant*	*Cond*$_{\text{out}}$
Current	z=z'	z=z'	z=z'	z<z'	z≤z'	z≤z'	z≤z'	z≤z'	z≤z'
Negation	z<z'	z≤z'	z=z'	z≥z'	z≥z'	z=z'	z>z'	z≥z'	z=z'
	z>z'	z≥z'	z=z'						



(a) Case of ⊳⊲ is '='



(b) Case of ⊳⊲ is '≤'



(c) Case of ⊳⊲ is '<'

Fig. 5. Translation of State transitions

Fig. 5 shows a translation result from the state transition in Fig.3a using the Table 1. Fig. 5a,b,c is the cases that $\triangleright\triangleleft$ is '=', '≤' and '<', respectively. In Fig.5a, The invariant of p_{22} is *Current Invariant*. $Cond_{\mathrm{out}}$ is the condition of the transition from p_{22} to p_3. $Cond_{\mathrm{prev}}$ is control modes of *Negation* which are p_{21} and p_{23}. When control is in p_1, then control changes by a result of comparison of z with z'. If control changes p_{22}, p_{23} control modes do not change until $z=z'$. If a value of z reaches z' while control mode is p_{21} or p_{23}, control changes to p_{22}. The jump condition of a transition (p_{21}, p_{23}) or (p_{21}, p_{23}) is from *Negation* $Cond_{\mathrm{out}}$. As soon as the control changes to p_{22}, it changes to p_3 by the urgent flag *asap*.

External Transitions. In case of external transitions, we first generate a transition in discrete input automaton in order to determine the time of executing external transition. And then we add a synchronization label to the transition and the external transition. Transition condition for the discrete input automaton has to use *asap*. A source control mode is *active* and the target control mode is *idle*. Execution scenario is as follows: When a discrete change occurs, the control changes from *idle* to *active*. As soon as the control has changed to *active*, the transition executes synchronously with the source transition through the synchronization label.

3.2 Coupled ECML Model

In the coupled models of ECML, we use the concept of structured model. A structured model contains basic models which connect with each other. Basic model are coupled with each other using connecting ports which describe data flow. Input coupling connects an input port of a structured model to an input port of a basic model. Output coupling connects an output port of a basic model to an output port of a structured model. Internal coupling connects an output port of basic model to input port of a basic model. Translating coupled model generates additional internal coupling automata from each internal coupling. Internal coupling is translated into automaton which is similar to discrete automaton or continuous automaton showed in Fig.4 with additional passing of values of connected ports. Translation of output coupling doesn't need to generate LHA.

4 Conclusion

This paper proposes translation rules from ECML to linear hybrid automata. The translation makes possible analysis and formal verification of ECML models using the HyTech model checker. We are now developing an automatic translator from ECML to LHA.

Acknowledgments. This work was supported by the IT R&D Program of MKE/KEIT [12ND-1310, "The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software"].

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138, 3–34 (1995)
2. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering 22, 181–201 (1996)
3. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: a model checker for hybrid systems. Software Tools for Technology Transfer 1, 110–122 (1997)
4. Asarin, E., Dang, T., Maler, O.: The **d/dt** Tool for Verification of Hybrid Systems. In: Brinksma, D., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 365–770. Springer, Heidelberg (2002)
5. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Hybrid Systems: Computation and Control, pp. 258–273 (2005)
6. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable Verification of Hybrid Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
7. Lee, D.A., Lee, J.H., Yoo, J., Kim, D.H.: Systematic verification of operational flight program through reverse engineering. In: International Conference on Advanced Software Engineering & Its Applications (2011) (submitted)
8. Praehofer, H., Auernig, F., Reisinger, G.: An environment for devs-based multiformalism simulation in common lisp/CLOS. Discrete Event Dynamic Systems: Theory and Application 3, 119–149 (1993)
9. Henzinger, T., Ho, P., Wong-Toi, H.: A user guide to hytech. Tools and Algorithms for the Construction and Analysis of Systems, 41–71 (1995)
10. Han, S., Huang, K.: Equivalent Semantic Translation from Parallel DEVS Models to Time Automata. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 1246–1253. Springer, Heidelberg (2007)
11. Choi, H., Cha, S., Jo, J.Y., Yoo, J., Lee, H.Y., Kim, W.T.: Formal Verification of DEV&DESS Formalism Using Symbolic Model Checker HyTech Control and Automation, and Energy System Engineering, pp. 112–121. Springer (2011)