Information and Software Technology 92 (2017) 121–144

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Formal verification of ECML hybrid models with spaceex

Sanghyun Yoon, Junbeom Yoo*

Konkuk University, 1 Hwayang-dong, Gwangjin-gu, Seoul 143-701, Republic of Korea

ARTICLE INFO

Article history:
Received 2 August 2016
Revised 29 July 2017
Accepted 31 July 2017
Available online 1 August 2017

Keywords:
Formal verification
ECML
Linear hybrid automata
Automatic translation
Spaceex

ABSTRACT

Context: ECML is a modeling language for hybrid systems, proposed by ETRI in Korea. ECML extended the basic formalism, DEV&DESS, with uses in modeling and simulation, whereas algorithmic verification on the ECML models continues to be an on-going research task.
Objective: This paper proposes a verification technique to verify ECML models with SpaceEx, a verification platform for hybrid systems. It includes translation rules from ECML into the SpaceEx model.
Method: As SpaceEx reads linear hybrid automata, we developed translation rules from ECML models to linear hybrid automata and implemented an automatic translator *ECMLtoSpaceEx*. We also developed a rule checker *ECML Checker* to check whether an ECML model complies with assumptions and restrictions to overcome the semantic gap between the two formal languages. We performed a case study with an extension of the widely used example 'barrel-filler system' to demonstrate the effectiveness of our verification technique.
Results: The verification result shows that our verification technique can translate ECML models into SpaceEx models, and we also perform formal verification on ECML models with SpaceEx.
Conclusions: The proposed technique can verify ECML with support fromSpaceEx. We expect that the proposed translation rules can be used with minor modifications to translate ECML models into different notations, and thus allow for the use of verification tools other than SpaceEx.

1. Introduction

A hybrid system [1] is a dynamical system whose behavior is a combination of continuous and discrete dynamics. The discrete part usually models modes of system operations, whereas the continuous part models physical interactions with environments. Many approaches for modeling and analyzing hybrid systems, based on finite state machines (FSMs), have been proposed. Timed automata [2], (linear) hybrid automata [3,4], DEV&DESS [5], *CHARON* [6], and ECML [7] are examples of modeling methods. *UPPAAL* [8], *KRONOS* [9], *HyTech* [10], *PHAVer* [11], and *SpaceEx* [12] are algorithmic verification tools (*i.e.,* for model checking [13,14]) for hybrid systems. Rigorous quality demonstration [15–19] is required for hybrid system models, since they are used for the modeling and analysis of safety-critical systems such as automotive, avionics, and military defense systems.

ECML (ETRI Cyber-Physical System Modeling Language) is an extension of the basic formalism DEV&DESS (Discrete Event & Differential Equation System Specification) [5] with various uses in modeling and simulation, and was recently proposed by ETRI (Electronics and Telecommunications Research Institute) in Korea. It has been used for hybrid system modeling and the simulation of systems such as those of robot control [20], military weapons, and unmanned aerial vehicles [21]. ETRI also proposed '*EcoSuite* [22]' which is a dependable CPS (Cyber-Physical System) software development environment. *EcoSuite* includes the ECML modeling environment '*EcoPOD*' (ETRI CPS Open Platform Developer), '*EcoSIM*' (ETRI CPS Simulator) and it supports HILS (Hardware-in-the-Loop Simulation) [23]. These modeling and simulation environments are presented in [24], and the formal definition of the ECML formalism is proposed in [7].

It, however, still needs algorithmic methods for verifying safety and reachability. Our previous studies [25–29] attempted to verify the DEV&DESS and ECML models with *HyTech*. The tool generates an *error trajectory*, and it helps the user to analyze the verification result, but *HyTech* may not be able to analyze complex systems [11,30]. All of our previous studies focused only on the formal verification of small and simple dynamics, and they used the *urgent transition* of *HyTech* to overcome the semantic gap between the ECML and *HyTech* models. We, therefore, required a more powerful approach capable of accommodating the ECML models used in practice. Here, we present a solution for verifying ECML models via *PHAVer* and *SpaceEx*. In [31], we present a pilot study performed to confirm that our proposal can perform more powerful verifications on more complicated models.

* Corresponding author.
E-mail addresses: pctkdgus@konkuk.ac.kr (S. Yoon), jbyoo@konkuk.ac.kr (J. Yoo).

http://dx.doi.org/10.1016/j.infsof.2017.07.014
0950-5849/

This paper proposes a verification technique that can verify hybrid system models written in ECML using the *SpaceEx* verification platform. *SpaceEx* supports three verification scenarios (*i.e.,* algorithms), which are *PHAVer* (*Polyhedral Hybrid Automaton Verifier*) [11], *LGG* (*Le Guernic-Girard*) [12], and *STC* (*Space-Time with Clustering*) [32]. *PHAVer* was a stand-alone tool and it is integrated into *SpaceEx*. *STC* is an enhancement of *LGG* that produces fewer convex sets for a given accuracy and computes more precise images of discrete transitions [33]. *LGG and STC* use nonlinear hybrid automata as an input model, whereas *PHAVer* uses linear hybrid automata (hereafter LHA) [4]. We choose *PHAVer* scenario to verify ECML models, since ECML models use deterministic transitions and we use linear dynamics to describe the behaviors of the deterministic transitions in hybrid automata model (see Section 3.5). When we choose the *PHAVer* scenario [34,35], *SpaceEx* uses LHA as an input front-end, and we first have to translate the ECML models into LHA models in order to make the *SpaceEx* verification possible.

We first developed a set of rules for translating an ECML model into LHA models. It covers the translation of coupled models as well as single models, and also considered the semantic gap between these two formalisms carefully. For example, ECML uses deterministic transitions, but LHA use nondeterministic transitions [36]. ECML variables are updated in three ways (*i.e.,* continuously, discretely, and based on events), whereas LHA have only continuous variables. The communication approaches between models are also different. LHA are synchronized with a synchronization label, whereas ECML, which has no such mechanism, only uses I/O-based sequential coupling. The proposed translation rules considered most of the semantic gaps and implemented an automatic translator '*ECMLtoSpaceEx.*' This translator reads an ECML model and translates it into LHA models for *SpaceEx* mechanically.

This paper also proposes a small set of assumptions and restrictions for modeling with ECML in order to overcome the gap that our translation rules could not address. ECML supports various conveniences in modeling (*i.e.,* syntactic sugars), which cannot be transformed into equivalent LHA. For example, ECML supports user-defined data types and external functions, defined with functions of the *C++* programming language. We developed '*ECML Checker*' to check an ECML model against the assumptions and restrictions, and it then advises experts to avoid building the model in these ways.

In order to demonstrate the effectiveness of tools, '*ECMLtoSpaceEx*'[1] and '*ECML Checker*' can conduct a formal verification with *SpaceEx* on ECML models, we performed a case study with an example of a '*barrel production system*' [29]. It extends the widely used example '*barrel-filler system*' of [5] into a larger system (*i.e.,* the '*barrel-filler system*' is part of the '*barrel production system*').

We modeled the system with ECML and used "*ECMLtoSpaceEx*" to translate it into a LHA for *SpaceEx*. We then successfully performed the *SpaceEx* reachability analysis to verify the *reachability* and *safety* properties against the mechanically translated model. *SpaceEx* supports graphical representations of variables as a counter-example, however they are neither easy to understand nor analyze. The verification results of *SpaceEx* shows all reachable sets when the model satisfies the specific condition (*i.e., forbidden states*) (Section 4.3). Therefore, our translation from ECML into linear hybrid automata uses intentionally extra elements such as clock variables to allow for a more in-depth analysis of the verification results.

This paper is organized as follows. Section 2 reviews and compares the two formalisms—ECML and linear hybrid automata—to aid the understanding of the translation proposed in the next Section. Section 3 proposes the translation rules from ECML to LHA for the *SpaceEx* verification. Section 4 reports on a case study with an ECML example of a *barrel production system*, and Section 5 surveys modeling and verification techniques for hybrid systems and compares them with the proposed technique. We conclude the paper in Section 6.

[1] Available at "http://dslab.konkuk.ac.kr/Hybrid-System/ECMLtoSpaceEx/ECMLtoSpaceEx.zip".

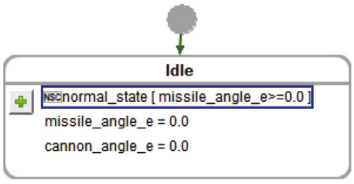


Fig. 1. Error modeling function of ECML.

2. Background

In the following sections we model a simple thermostat example [10] with ECML and LHA and propose formal definitions for the formalisms. Besides that, we review and compare the two formalisms—ECML and LHA.

2.1. ECML

ECML [7] is a modeling language for hybrid systems [37] that was recently proposed by ETRI in Korea. It extends the basic formalism DEV&DESS with various conveniences, such as hierarchies and error modeling. For example, error modeling is an optional function for ECML simulation. An expert set a normal state for a variable as described in Fig. 1. When the normal state is violated during simulation, the simulation environment informs the expert about the violation.

The basic component of ECML models is *BM* (Behavioral Model). Fig. 2 describes an ECML *BM* for a thermostat system which is a modified model of [10]. The system models states of a heater in order to control the temperature. It has three phases, *switch_off* (the power switch of the heater is turned off), *on* (the heater is turned on) and *off* (the heater is turned off). The initial phase is *switch_off*, thus the behavior of the model starts in *switch_off* phase. The input variable x_1^D represents a power switch. When the power switch is turned on (*i.e.,* $x_1^D = 1$), the model transits to *on* phase and a lamp is turned on (*i.e.,* $y_1^D = 1$) to indicate state of the switch. The value of the continuous state variable s_1^C represents temperature, and the value rises at the rate two degrees per time units ($d(s_1^C) = 2$) in *on* phase. The heater is turned off (*i.e., the model transits to off*) when the temperature reaches four degrees. The temperature falls according to $d(s_1^C) = -4$, therefore, it returns to *on* phase in order to turn on the heater after one time unit.

ECML models have three different *rate types* of variables: discrete events (E), discrete values (D), continuous values (C). The three types of variables have different behavior with respect to how their values evolve over time. When an event occurs, a discrete event type variable is assigned a value and is reset to zero. The value of discrete value type variable is held constant until the next assignment occurs. The value of a continuous value type variable changes continuously, and *rate* defines the rate of change of the value at a *phase*.

Variables in ECML also have *port types* and various *data types*. ECML has three *port types: input* (X), *output* (Y), and *state* (continuous state S^C, discrete state S^D). An ECML *BM* obtains data from a connected model via *input variables*, changes the internal states with *state variables*, and sends data with *output variables*. *Data types* determine the possible values for the variable. *Integer, Double, Boolean* and *String* are the types supported by ECML. The modeling

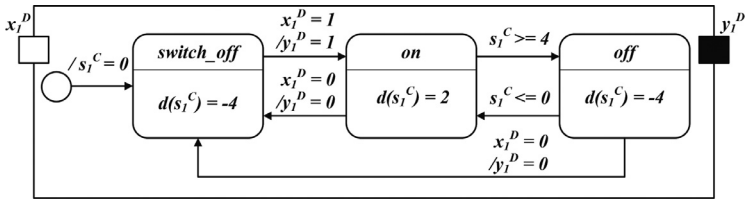


Fig. 2. ECML BM_{TH} for the thermostat system.

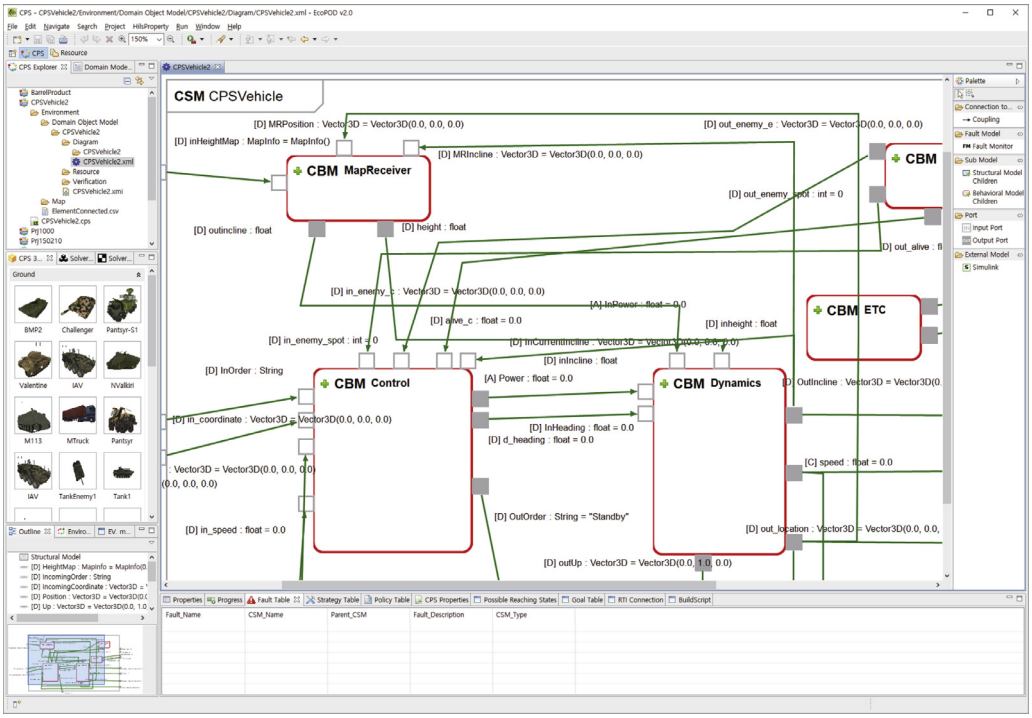


Fig. 3. Modeling and simulation tools for ECML ('*EcoPOD* and '*EcoSIM*'.

environment "*EcoPOD*" [38,39] and the visual simulator "*EcoSIM*" [40,41] support visual modeling and simulation [24], as illustrated in Fig. 3.

An ECML model is hierarchically structured into a set of basic components. A basic component is modeled as a *BM* and their composition is modeled as an *SM* (Structural Model). Fig. 3 shows an example model in EcoPOD. '*CBMs*' (CPS Behavioral Models) are *BMs* and rectangles tagged '*CSM*' (CPS Structural Model) are *SMs*. They are connected via port variables. We use [*A*] (Analog) to depict continuous values in *EcoPOD*, because [*C*] is used to depict constant variables. A *BM* is defined as follows:

$BM = \langle X, Y, S, Init, Cond^E, Trans^E, Out^E, Cond^S, Trans^S, Out^S, Out^C, Out^D, Rate \rangle$

- $X = X^C \times X^D \times X^E$ is the set of inputs,
  where,
  - $X^C = \{(x_1^C, x_2^C, \dots) \mid x_1^C \in X_1^C, x_2^C \in X_2^C, \dots\}$ is the structured set of continuous value inputs with input variables x_i^C
  - $X^D = \{(x_1^D, x_2^D, \dots) \mid x_1^D \in X_1^D, x_2^D \in X_2^D, \dots\}$ is the structured set of discrete value inputs with input variables x_i^D
  - X^E is the set of discrete event inputs
- $Y = Y^C \times Y^D \times Y^E$ is the set of outputs,
  where,
  - $Y^C = \{(y_1^C, y_2^C, \dots) \mid y_1^C \in Y_1^C, y_2^C \in Y_2^C, \dots\}$ is the structured set of continuous value outputs with output variables y_i^C
  - $Y^D = \{(y_1^D, y_2^D, \dots) \mid y_1^D \in Y_1^D, y_2^D \in Y_2^D, \dots\}$ is the structured set of discrete value outputs with output variables y_i^D
  - Y^E is the set of discrete event outputs
- $S = P \times S^D \times S^C$ is the set of states: the Cartesian product of phases P, discrete states S^D and continuous states S^C
- $Init = S_0 \times X_0 \times Y_0$ is the initial condition set to define the initial states and initial values of inputs and outputs
- $Cond^E$: $S \times X \nrightarrow Bool$ is the external event transition condition function for conditioning the execution of the external events
- $Trans^E$: $S \times X \nrightarrow S$ is the external event transition function
- Out^E: $S \times X \nrightarrow Y$ is the output function for external event transitions
- $Cond^S$: $S \times X^C \times X^D \nrightarrow Bool$ is the state transition condition function for conditioning the execution of the internal state events
- $Trans^S$: $S \times X^C \times X^D \nrightarrow S$ is the internal state transition function
- Out^S: $S \times X^C \times X^D \nrightarrow Y$ is the output function for internal state transitions
- Out^C: $S \times X^C \times X^D \nrightarrow Y^C$ is the continuous value output function
- Out^D: $P \times S^D \times X^D \nrightarrow Y^D$ is the discrete output function
- $Rate$: $S \times X^C \times X^D \nrightarrow S^C$ is the rate of change function

A *BM* corresponds to a DEV&DESS model. The semantics of an ECML *BM* is described as follows:

1. *Intervals* $\langle t_1, t_2 \rangle$ *with no events:* Only the continuous states S^C change. The continuous states at the end of the interval are computed from the state at the beginning plus the integral of the rate of change function $Rate(s(t), x^C(t), x^D(t))$ $(t = \langle t_1, t_2 \rangle)$ along the interval. The continuous behavior of the model is specified by $Rate(s(t), x^C(t), x^D(t))$ and the continuous value

$BM_{TH} = \langle X,\ Y,\ S,\ Init,\ Cond^E,\ Trans^E,\ Out^E,\ Cond^S,\ Trans^S,\ Out^S,\ Out^C,\ Out^D,\ Rate \rangle$

- $X = \{x_1^D \mid x_1^D \in \mathbb{R}\}$
- $Y = \{y_1^D \mid y_1^D \in \{0,1\}\}$
- $P = \{switch\_off, on, off\}$
- $S^C = \{s_1^C \mid s_1^C \in \mathbb{R}\}$
- $Init = \{P = switch\_off, s_1^C = 0\}$
- $Cond^E$: $(P = switch\_off, x_1^D = 1)$, $(P = on, x_1^D = 0)$, $(P = off, x_1^D = 0)$
- $Trans^E$: $(P = switch\_off, x_1^D = 1) \mapsto (P = on)$
  $(P = on, x_1^D = 0) \mapsto (P = switch\_off)$
  $(P = off, x_1^D = 0) \mapsto (P = switch\_off)$
- Out^E: $(P = switch\_off, x_1^D = 1) \mapsto (y_1^D = 1)$
  $(P = on, x_1^D = 0) \mapsto (y_1^D = 0)$
  $(P = off, x_1^D = 0) \mapsto (y_1^D = 0)$
- $Cond^S$: $(P = on, s_1^C \geq 4)$, $(P = off, s_1^C \leq 0)$
- $Trans^S$: $(P = on, s_1^C \geq 4) \mapsto (P = off)$, $(P = off, s_1^C \leq 0) \mapsto (P = on)$
- Out^S: $\emptyset$
- Out^C: $\emptyset$
- Out^D: $\emptyset$
- $Rate$: $(P = switch\_off) \mapsto (d(s_1^C) = -4)$, $(P = on) \mapsto (d(s_1^C) = 2)$, $(P = off) \mapsto (d(s_1^C) = -4)$

Fig. 4. The formal definition of BM_{TH} for the '*Thermostat System*'.

output function $Out^C(s(t), x^C(t), x^D(t))$, whereas the discrete value output is generated by the discrete value output function $Out^D(p(t), s^D(t), x^D(t))$.

2. *An internal state event occurs first at time t in interval* $\langle t_1, t_2 \rangle$: The continuous states at the time of the transition are computed from the state at the beginning plus the integral of the rate of change function $Rate(s(t'), x^C(t'), x^D(t'))$ $(t' = \langle t_1, t])$ along the interval until time *t*. Likewise, the hybrid outputs are generated until time *t*. At time *t*, the state transition condition function $Cond^S(s(t), x^C(t), x^D(t))$ evaluates to true. That is, an internal state event occurs. Here, the internal state transition function $Trans^S$ $(s(t), x^C(t), x^D(t))$ is executed to define a new state. The output function for internal state transitions Out^S $(s(t), x^C, x^D(t))$ is called to generate an output at time *t*.
3. *An external discrete event occurs first at time t in interval* $\langle t_1, t_2 \rangle$: The continuous states at the time of the transition are computed from the state at the beginning plus the integral of $Rate(s(t'), x^C(t'), x^D(t'))(t' = \langle t_1, t])$ along the interval until t. Likewise, the hybrid outputs are generated until time *t*. At time *t*, the external event transition condition function $Cond^E(s(t), x(t))$ evaluates to true. That is, the external event transition occurs. Here, the external event transition function $Trans^E(s(t), x(t))$ is executed to define a new state. The output function for external event transitions $Out^E(s(t), x(t))$ is called to generate an output at time *t*.

The formal definition for the BM_{TH} for the thermostat system is described in Fig. 4.

An *SM* corresponds to a coupled DEV&DESS model. A structural model contains basic behavioral models (*BMs*), coupled to each other with connecting ports describing the flow of rate typed as discrete, continuous, or event. '*CSM CPSVehicle*' in Fig. 3 is an example of an ECML *SM*. The formal definition of an *SM* is based on the formal definition of the coupled DEV&DESS model [5]. It uses *influencer* to represent I/O connection (*i.e., coupling*) of models. The *influencer* is a model (*i.e.,* BM or SM) of which the output variable influences the input variable of the other model. For example, '*CBM Control*' is an *influencer* of '*CBM Dynamics*,' and the output translation function $z_{i,r}$ defines I/O connection of the models. The SM formal definition includes formal definitions of BMs which are included corresponding SM. An *SM* is defined as follows:

$SM = \langle X_S, Y_S, R, BM, I, Z, Select \rangle$

- $X_S = X_S^C \times X_S^D \times X_S^E$ is the set of inputs of the structural model

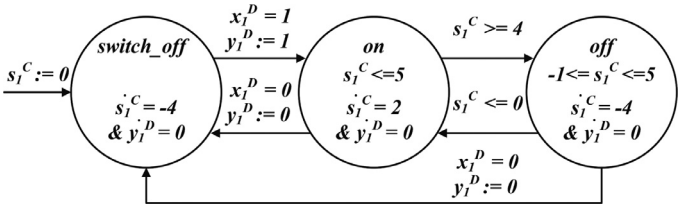


Fig. 5. Linear hybrid automaton LHA_{TH} for the thermostat system.

- $Y_S = Y_S^C \times Y_S^D \times Y_S^E$ is the set of outputs of the structural model
- $R = R^B \times R^S$ is the set with a single references to the current structural model, where,
  - R^B is the set of behavioral model references
  - $R^S = \{r^S\}$ is the set of the structural model reference
- For each $r^B \in R^B$,
  - $bm_{r^B} \in BM$ is a behavioral model,
    $bm_{r^B} = \langle\ X, Y, S, Init, Cond^E, Trans^E, Out^E, Cond^S, Trans^S, Out^S, Out^C, Out^D, Rate \rangle$
- For each $r \in R$,
  - I_r is the set of influencers. $I_r \subseteq R$, $r \notin I_r$
- For each $i \in I_r$, $z_{i,r} \in Z$ is the function mapping i to r as follows
  - $z_{i,r}$: $X_S \nrightarrow X_r$ and $Select = bm_r$, if $i = r^S$
  - $z_{i,r}$: $Y_i \nrightarrow Y_S$ if $r = r^S$
  - $z_{i,r}$: $Y_i \nrightarrow X_r$ and $Select = bm_r$, if $i \neq r^S$ and $r \neq r^S$
- *Select* is the function selecting a behavioral model to be executed

2.2. Linear hybrid automata

LHA [4] is a kind of hybrid automata [42]. At each location of LHA, the behavior of all variables is governed by linear constraints on the first derivatives. Fig. 5 describes a linear hybrid automaton for the thermostat system. The states of the heater are the *switch_off, on* and *off* locations. Initial location is *switch_off* and the model transits to *on* when the power switch is turned on $(x_1^D = 1)$. When the model remains in the *on* location, the value of temperature (s_1^C) rises at the rate two degrees per time unit $(\dot{s}_1^C = 2)$. The transitions of LHA are nondeterministic transitions; thus, the transition might not occur although the jump condition $s_1^C \geq 4$ is true. In order to remain in the *on* location, the variable should satisfy the invariant condition of *on* $(s_1^C \leq 5)$. Therefore, the model transits from *on* to *off* when the value of s_1^C is $4 \leq s_1^C \leq 5$.

Some elements have different characteristics between ECML models and linear hybrid automata. In LHA, variables can have continuous, real-valued types only. Transitions in ECML are deterministic, whereas the control switches of linear hybrid automaton are nondeterministic. LHA also have invariant conditions (*inv*). A linear hybrid automaton basically uses shared variables and synchronization labels (*syn*) to communicate with another linear hybrid automaton, whereas ECML uses a port connection (to output variables). There is no hierarchy in LHA (*i.e.,* it is a flat system). However, *SpaceEx* supports port connections whereas basic LHA [10] do not support this feature. Therefore, we use the formal definition of [11] in order to specify port connections of *SpaceEx* LHA model. The translation rules consider not only characteristics of basic LHA but also characteristics of *SpaceEx* LHA model. A linear hybrid automaton is defined as follows:

$LHA = \langle X, V, flow, inv, init, E, jump, \Sigma, syn \rangle$

- X is a set of continuous, real-valued variables. X is divided into *controlled* variables C and *input* variables $I = X \setminus C$, and a subset O of C is designated as the *output* variables [11].
- V is a set of control modes (locations)
- *flow* is a labeling function that assigns a flow condition to each control mode $v \in V$. The flow condition $flow(v)$ is a predicate over the variables in $X \cup \dot{X}$, and it refers to the first derivative of $x_i \in \dot{X}$. While the control of the hybrid automaton A is in mode v, the values of the variables and their first derivatives satisfy the flow condition $flow(v)$.
- *inv* is a labeling function that assigns an invariant condition to each control mode $v \in V$. While the control of the hybrid automaton A is in mode v, the variables in X must satisfy the invariant condition $inv(v)$.
- *init* is a labeling function that assigns an initial condition to each control mode $v \in V$.
- E is a finite multiset of control switches. Each control switch (v, v') is a directed edge between a source mode $v \in V$ and a target mode $v' \in V$.
- *jump* is a labeling function that assigns a jump condition to each control switch $e \in E$. The jump condition $jump(e)$ is a predicate over variables in $X \cup X'$, where $X' = \{x'_1, x'_2, \ldots, x'_n\}$. The unprimed symbol x_i, for $1 \le i \le n$, refers to the value of the variable x_i before the control switch, and the primed symbol x'_i refers to the value of x_i after the control switch.
- Σ is a finite set of events
- A labeling function *syn* that assigns an event in Σ to each control switch $e \in E$.

LHA satisfy the following two requirements [10], *Linearity* and *Flow independence*.

1. Linearity: For every control mode $v \in V$, the flow condition $flow(v)$, the invariant condition $inv(v)$, and the initial condition $init(v)$ are convex linear predicates. For every control switch $e \in E$, the jump condition $jump(e)$ is a convex linear predicate.
2. Flow independence: For every control mode $v \in V$, the flow condition $flow(v)$ is a predicate over the variables in $\dot{X}$ only (and does not contain any variables from X).

A *SpaceEx* model is structured into subsystems and they communicate with each other using input and output variables. *SpaceEx* classifies variables according to two criteria. One is *local* and the other is *controlled*. A *local* variable is only used in the corresponding subsystem. A *controlled* variable is assigned a value in the subsystem, thus a *local* variable is a subset of *controlled* variables. A *flow* for a *controlled* variable should be defined in the subsystem, if not, the variable can have a random value ($[-\infty, \infty]$). If a variable is not classified as a *controlled* variable (*i.e., input variable*), *flow* for the variable does not need to be defined in the subsystem. In Fig. 5, we assume that s_1^C and y_1^D are *controlled* variables and x_1^D is an input variable. Thus, LHA_{TH} has *flow conditions* for s_1^C and y_1^D in each location, while the model does not have *flow conditions* for the input variable x_1^D. The formal definition for LHA_{TH} is defined as described in Fig. 6.

2.3. The semantic gap between these two formalisms

This paper proposes to use *SpaceEx* as a means to verify ECML models formally. Therefore, we first need to translate ECML models into LHA models. This section compares the two formalisms as shown in Table 1, and briefly overviews the semantic gaps between them and how we could resolve some of them. All important semantic gaps to be resolve are labeled as *G.1* ∼ *G.7*. We left *G.1* and *G.7* to the modeler with support of our CASE tool ('*ECML Checker*'), and resolve the other gaps except for a specific case of *G.6*. Section 3 explains how the gaps are addressed in details.

- *(G.1 Dynamics)* ECML supports nonlinear dynamics, but LHA does not. We advise the ECML modeler to use linear dynamics if formal verification via *SpaceEx* is required.
- *(G.2 Variables)* Variables in ECML consist of *port types, rate types*, and *data types*, whereas LHA has only one type – the *continuous real-value type*. All types of ECML variables should be translated appropriately.
- *(G.3 Transitions)* Transitions of ECML are enabled deterministically, whereas *control switches* of LHA transit nondeterministically. We need to make the translated LHA work deterministically as ECML.
- *(G.4 Invariants)* ECML does not support *invariant* as LHA. The semantic gap is not a problem we need to resolve because we translate ECML models into LHA models. We use *invariants* to resolve the **G.3**.
- *(G.5 Communication)* An ECML BM shares data with another BM through I/O port connections, whereas LHA uses shared variables and broadcasting events with synchronization labels. I/O ports are translated into shared variables, and we also use the synchronization labels to reset *discrete event* ports.
- *(G.6 Variable Assignments)* ECML assigns a value at *phase* and transition, whereas LHA does this at transitions only. Furthermore, when a transition has more than one variable assignment, ECML processes it sequentially, whereas LHA does it in parallel. We translate an ECML model into an equivalent LHA, which has a particular structure to guarantee the assignment order (see Sections 3.1.6 and 3.1.7).
- *(G.7 User-Defined Functions)* User-defined functions of ECML cannot be translated naïvely, since they are defined using the *C++* programming language.

3. The ECMLtoLHA translation

This section proposes translation rules from an ECML model into LHA models. The rules map each element of ECML into a corresponding element of LHA. Whereas most of the rules are 1:1 translations, some rules are not straightforward due to the semantic gap between these two formalisms. Fig. 7 overviews the structure of translation from ECML models to LHA models.

Section 3.1 describes the translation of behavioral models, whereas Section 3.2 presents the translation of structural models. Section 3.3 explains the restrictions and assumptions of the proposed translation rules. The translation process from ECML models to LHA models and the supporting tools are introduced in Section 3.4. We discuss further considerations of the proposed translation rules in Section 3.5.

$LHA_{TH} = \langle\ X,\ V,\ flow,\ inv,\ init,\ E,\ jump,\ \Sigma,\ syn\ \rangle$

- $I = \{x_1^D\}$
- $C = \{s_1^C, y_1^D\}$
- $O = \{y_1^D\}$
- $V = \{switch\_off, on, off\}$
- $flow(V)$: $(V = switch\_off) \mapsto (\dot{s_1^C} = -4 \wedge \dot{y_1^D} = 0)$
  $(V = on) \mapsto (\dot{s_1^C} = 2 \wedge \dot{y_1^D} = 0)$
  $(V = off) \mapsto (\dot{s_1^C} = -4 \wedge \dot{y_1^D} = 0)$
- $inv(V)$: $(V = on) \mapsto (s_1^C \leq 5)$, $(V = off) \mapsto (-1 \leq s_1^C \leq 5)$
- $init(V)$: $(V = switch\_off) \mapsto (s_1^C := 0)$, $(V = \{on, off\} \mapsto false)$
- $E = \{(switch\_off, on), (on, switch\_off), (on, off), (off, on), (off, switch\_off)\}$
- $jump(V, V)$: $(switch\_off, on) \mapsto (x_1^D = 1 \wedge y_1^D := 1)$
  $(on, switch\_off) \mapsto (x_1^D = 0 \wedge y_1^D := 0)$
  $(on, off) \mapsto (s_1^C \geq 4)$
  $(off, on) \mapsto (s_1^C \leq 0)$
  $(off, switch\_off) \mapsto (x_1^D = 0 \wedge y_1^D := 0)$
- $\Sigma = \emptyset$
- $syn(e, jump(e))$:$\emptyset$

Fig. 6. The formal definition of LHA_{TH} for the '*Thermostat System*'.

Table 1
Comparison of ECML and LHA.

Category	ECML	LHA
G.1 Dynamics	Nonlinear dynamics	Linear dynamics
G.2 Variables		
G.2.1. Port Type	State/Input/Output	Variable *State/Input/Output
G.2.2 Rate Type	Continuous/Discrete value/Discrete event	Continuous
G.2.3 Data Type	Integer, Double, String, User-defined	Real
G.3 Transitions	Deterministic	Non-deterministic
G.4 Invariants	Unsupported	Supported
G.5 Communication	I/O port connection	Shared variables & Synchronized labels *I/O port connection & Synchronized labels
G.6 Variable Assignments	Phase & Transition/Sequential	Jump Condition/Parallel
G.7 User-Defined Functions	Supported	Unsupported

*in SpaceEx

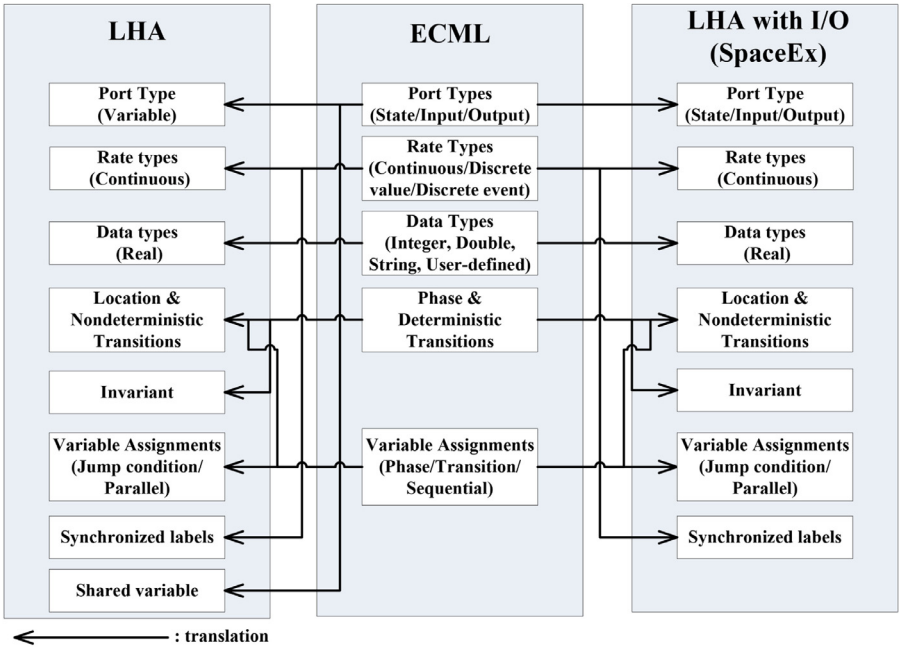


Fig. 7. Overview of translation structure from ECML models to LHA models.

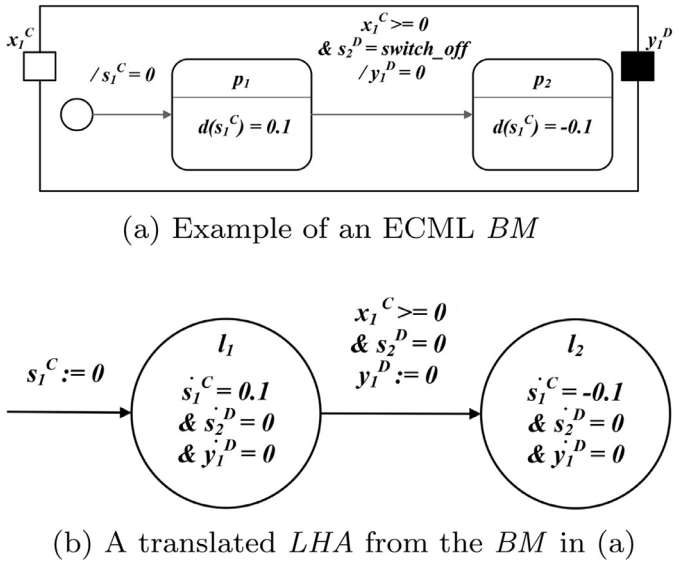


Fig. 8. Example of a basic translation from ECML *BM* into LHA.

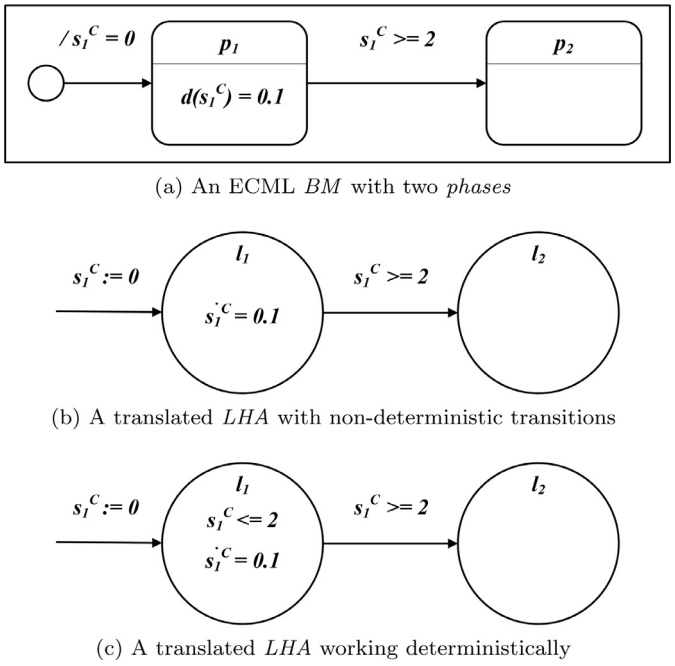


Fig. 9. Examples of translating deterministic/non-deterministic transitions.

3.1. The translation of behavioral models

A Behavioral Model (*BM*) comprises variables with initial conditions, *rate*, *phases*, and transitions. Each element of a *BM* is translated into a corresponding element of an *LHA* as explained in the following sections. On the other hand, some elements of ECML cannot be translated into LHA directly and involve auxiliary definitions. The translation of sequential assignments of ECML models is an example, and we proposed appropriate solutions for each case.

3.1.1. Variables

Every *input/output variable* (*i.e.*, *X* and *Y*) and *state variable* (*i.e.*, S^D and S^C) of *BM* is translated into a corresponding variable of LHA. A variable of ECML has a *rate type* and *data type*, whereas LHA support continuous real-valued variables only (*G.2*). Therefore, the ECML variables of various *rate types* and *data types* are translated into continuous real-valued variables. Fig. 8(a) describes example of an ECML *BM*. The model has two phases p_1 and p_2. When the external input $x_1^C \geq 0$ is true and the value of discrete state s_2^D is '*switch_off*,' the model transits to p_2 besides the discrete output variable y_1^D is assigned 0 value.

The *rate* types of ECML (*continuous, discrete*, and *event*) are translated into the *continuous* type using the *flow* element of LHA. The *continuous* type is translated straightforwardly, but a zero flow (*e.g.*, $\dot{y_1^D} = 0$ as depicted in Fig. 8) is added to every location for the variables of *discrete*/*event* types. The *discrete*/*event* type variables are only defined in output functions (*Out*) and the values of the variables are not changed continuously in ECML models. It is noteworthy that input variables of ECML can be translated in two ways. In case of those connected to a top-level *SM* (*i.e.*, it obtains input values from the external environment), it is translated into a variable with random values. This is achieved by not defining a *flow* condition to this variable. If the input obtains values from other connected *BM*, we consider this connection in the translation.

Integer, Double, and *Boolean* data types of ECML are translated into the *Real* type of LHA without losing information. *String* data type variables are used to describe simulation user commands such as '*switch_on*' and '*switch_off*' and these are translated into specific $\mathbb{R}$ values in LHA. In Fig. 8(a), s_2^D is a *String* data type variable, used in the transition condition between p_1 and p_2. Considering that '*switch_off*' is mapped to the number 0, the condition $s_2^D =$ *switch_off* is translated into $s_2^D = 0$.

3.1.2. Initial condition

A *BM* has an initial condition *Init* consisting of an initial *phase* and a set of initial values of all variables. An LHA also has an initial *control mode* and a set of initial values of all variables. All corresponding elements of initial conditions should coincide with each other.

3.1.3. Rate

The *rate* of each *phase* in a *BM* corresponds to the *flow* of each *control mode* in LHA. We write "$\dot{contents} = a$" or "$contents' == a$", (a is a real value constant) to describe the *flow* condition of each *control mode* in LHA, while *BM* uses the form of "$d(contents) = a$." As described in Section 3.1.1, discrete and event type variables are translated into continuous variables in LHA. If a *flow* condition is not used to define the variables, an LHA recognizes that the variable can have $[-\infty, \infty]$ ranged values (*i.e.*, random values). This is the case for input variables that receive values from the external environment, as previously explained. The *rate* of discrete and event type variables is defined using zero-valued *flow* conditions.

3.1.4. Phase & transition

Phase and *transition* in *ECML* basically correspond to *control mode* (*i.e.*, *location*) and (*source control mode, target control mode*) $\in E$ in LHA, respectively. Each *condition* and *action* statement on a transition in a *BM* are equivalent to the *jump* of an *LHA*. The straightforward translation of *phases* and *transitions*, however, results in different behaviors of LHA, since the deterministic transitions of ECML will be translated into non-deterministic ones of LHA *(G.3)*. When a transition condition is *true*, then a deterministic transition of ECML will be executed without delay. On the other hand, even if a transition condition is satisfied, a non-deterministic transition of LHA may not be enabled [43].

For example, Fig. 9(a) is an ECML *BM* consisting of two phases (p_1 and p_2) and one transition (p_1, p_2). When the transition condition "$s_1^C >= 2$" with a continuous state variable s_1^C is true in p_1, the transition is executed and the *phase* changes into p_2 without time advance (*i.e.*, immediately). The *BM* can be translated into the LHA model in Fig. 9(b) straightforwardly. However, the *control* switch between l_1 and l_2 might not happen even if the *jump*(l_1,

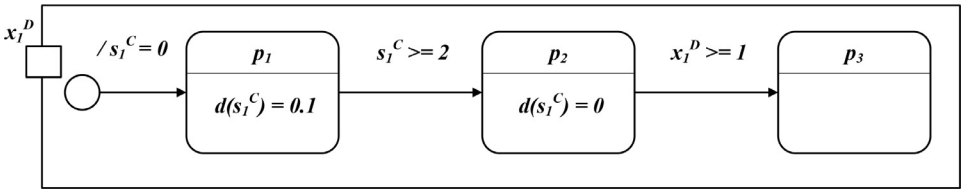


Fig. 10. An ECML *BM* with three *phases*.

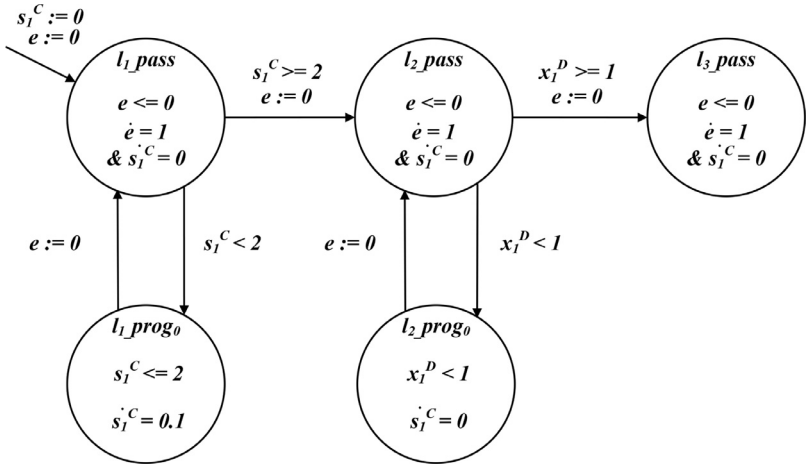


Fig. 11. Translated LHA from the ECML *BM* presented in Fig. 10.

l_2) condition is satisfied, and the translated LHA model in Fig. 9(b) now shows a different behavior with respect to the original ECML model in Fig. 9(a). On the other hand, the translated LHA in Fig. 9(c) works deterministically. If the invariant condition "$s_1^C <= 2$" is not satisfied, it cannot remain at l_1. So it has to transit to l_2 immediately, and the value of s_1^C is 2 at that time. Finally, the translated Fig. 9(c) now works equivalently with its origin Fig. 9(a).

It appears that the translation with an invariant as presented in Fig. 9(c) could simply solve the problem (*G.3*). It, however, cannot solve many common cases such as "a *phase* has ingoing transitions and outgoing transitions simultaneously," as shown in Fig. 10. As the transition condition of (p_2, p_3) is "$x_1^D >= 1$," the solution of Fig. 9(c) suggests to use "$x_1^D <= 1$ " as an *invariant* at p_2. However, if x_1^D has a larger value than 1 at p_1 (*i.e.,* if it does not satisfy the *inv* of p_2), transition from p_1 to p_2 never happens, and the translation will result in an obviously different behavior. We, therefore, propose a more elaborate solution, '*passing location*' and '*progress location*' as explained in Section 3.1.5.

3.1.5. Passing and progress locations

We propose '*passing location*' and '*progress location*' in order to accommodate the deterministic behavior of ECML. A *phase* of an EMCL *BM* is translated into one *passing location* and several *progress locations*. For translation of the whole LHA from an ECML *BM*, we also use an additional variable '*e*' to indicate elapsed time at *passing locations*. A *passing location* is a window for transiting to other locations immediately. Time does not advance due to the *inv* and *flow* at the *passing location*, and the outgoing transitions can transit to specific locations immediately when *jump* conditions are satisfied. At a *progress location*, time advances and values of variables change according to their *flow*.

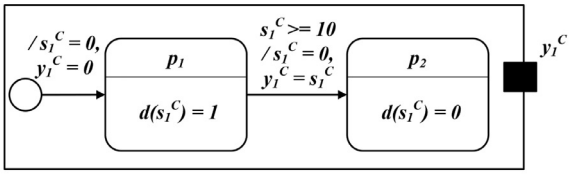
Fig. 11 shows the translated LHA from the ECML *BM* in Fig. 10. l_1_pass, l_2_pass, and l_3_pass are *passing locations* and $l_1\_prog_0$ and $l_2\_prog_0$ are *progress locations*. Henceforth, we use '*_pass*' and '*_prog*' to indicate *passing location* and *progress location*, respectively. Every *passing location* has a *flow* "$\dot{e} = 1$" and an *inv* "$e <= 0$." The initial value of *e* is zero (0) and all ingoing/outgoing *control switches* of *passing locations* reset it as "$e := 0$." We consider the complement of the *jump* condition from a *passing location* to create an *inv* condition of a *progress location*.

The translated LHA in Fig. 11 starts at l_1_pass, but it transits to $l_1\_prog_0$ immediately due to the *inv* and *flow* "$e <= 0 \wedge \dot{e} = 1$". At $l_1\_prog_0$, the time advances and s_1^C increases continuously up to 2. When the *inv* "$s_1^C <= 2$" does not hold, it transits back to l_1_pass immediately, besides resetting *e*. Then, it also immediately transits to l_2_pass because of the *jump* condition "$s_1^C >= 2$".

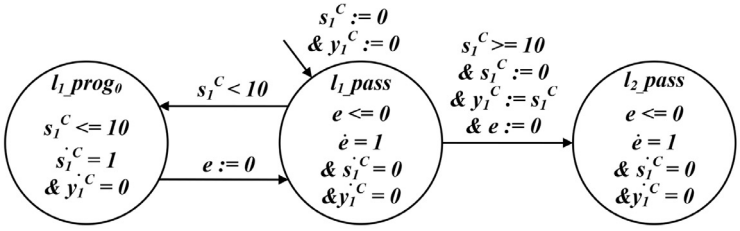
Details of translation using *passing* and *progress locations* depend on the *rate types* of variables in transition conditions. In Fig. 10, the transition condition of (p_2, p_3) has a discrete value variable "x_1^D." It is translated into a continuous variable, but the translated variable could retain a value for a while, since every *flow* for the variable is $\dot{x_1^D} = 0$ (Section 3.1.1). In this case, *inv* of a *progress location* should be the negation of the *guard*. When the LHA in Fig. 11 remains at $l_2\_prog_0$, and the value of x_1^D is then changed to 1, it transits to l_3_pass through l_2_pass ($\{x_1^D | x_1^D \in inv(l_2\_prog_0) \wedge x_1^D \in jump(l_2\_prog_0, l_2\_pass)\} \neq \varnothing$). If $inv(l_2\_prog_0)$ were "$x_1^D <= 1$," it could remain in $l_2\_prog_0$ because $inv(l_2\_prog_0)$ is satisfied when $x_1^D == 1$. On the other hand, concerning continuous variables, if $inv(l_1\_porg_0)$ were "$s_1^C < 2$," $jump(l_1\_pass, l_2\_pass)$ would not be satisfied (the value of s_1^C increases up to 1.999...). The proposed translation using *passing location* and *progress location* can accommodate the deterministic behavior of ECML successfully. The translation of transitions that have *event* type variable is described in Section 3.2.2.

3.1.6. Sequential variable assignment

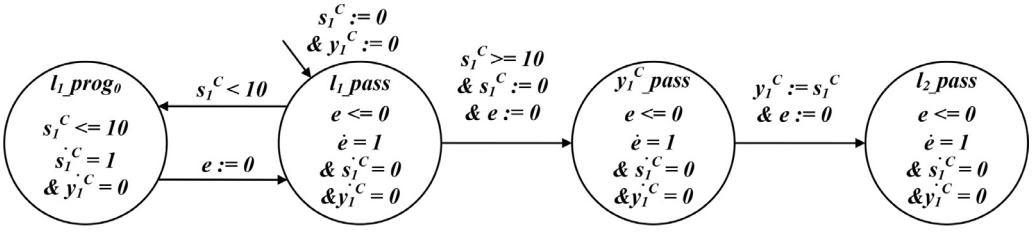
The transition (p_1, p_2) has an assignment $s_1^C := 0 \wedge y_1^C := s_1^C$ in Fig. 12(a), and it could be translated into Fig. 12(b) without consideration for *sequential assignments* of ECML. The assigned values of y_1^C are different after ($p1$, $p2$) and (l_1_pass, l_2_pass) are performed, since ECML assigns values sequentially, whereas LHA does it in parallel (*G.6*). When $s_1^C >= 10$, ECML assigns "$s_1^C := 0$" first, after which it assigns "$y_1^C := s_1^C$"; thus, the values of s_1^C and y_1^C are both '0.' LHA assigns "$s_1^C := 0$" and "$y_1^C := s_1^C$" in parallel, hence the value



(a) Example of sequential assignment



(b) Example of parallel assignment



(c) *LHA* translated from (a)

Fig. 12. Example of sequential variable assignment.

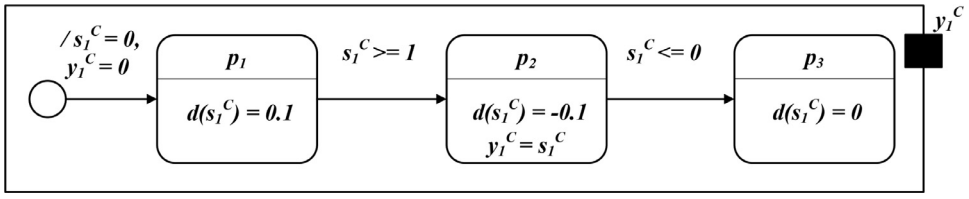


Fig. 13. ECML *BM* with an output function in a *phase*.

of s_1^C is "0" and that of y_1^C is "10" in this case. We use one more *passing location* to resolve the semantic gap as shown in Fig. 12(c), and it is a modified LHA from Fig. 12(b). It has the *passing location* y_1^C*_pass* and the *jump*(l_1*_pass*, l_2*_pass*) in Fig. 12(b) is divided into *jump*(l_1*_pass*, y_1^C*_pass*) and *jump*(y_1^C*_pass*, l_2*_pass*), therefore the values of s_1^C and y_1^C are both "0" at l_2*_pass*.

3.1.7. Variable assignment at phase

Although an LHA assigns a value to a variable at labeling functions (*jumps*) on transitions, ECML can do it at both transitions and *phases (G.6)*. We, therefore, should provide a rule to translate variable assignments on *phases* of ECML into ones on *transitions* of LHA, without distorting behaviors. For example, in *ECML*, the value of an output variable can change without transiting between *phases*, as shown in Fig. 13. s_1^C is a continuous state variable and y_1^C is a continuous output variable. p_2 has an output function "$y_1^C = s_1^C$," assigning the current value of s_1^C to y_1^C continuously when the *BM* remains at p_2. The value of y_1^C becomes 1 when the *BM* enters p_2, and decreases until 0 along with s_1^C before transiting to p_3.

Fig. 14 is the LHA translated from the ECML *BM* presented in Fig. 13. The *control switches* (l_1*_pass*, l_2*_pass*) and (l_2*_pass*, l_3*_pass*) have the same assignment ($y_1^C := s_1^C$), thus y_1^C has the same value of s_1^C whenever it enters and exits l_2*_pass*. When it remains at l_2*_prog*$_0$, the *flow* "$\dot{s_1} = -0.1$ & $\dot{y_1} = \dot{s_1}$" maintains the equivalence of the values of y_1 and s_1.

Our translation rules cover all cases except one case such as that depicted in Fig. 15. ECML supports assignments containing input variables at *phases* (*e.g.,* $y_1^C = x_1^C$ at p_1 in Fig. 15). In this case, the ECML model cannot be translated into LHA. The connected model could transit its own *phases* many times and change the value of the input variable, whereas the *BM* model shown in Fig. 15 remains at p_1. However, there is no element to reflect the changes without a *jump* in LHA. Our rule checker "*ECML Checker*" forbids creating models in such a situation.

3.2. The translation of structural models

An ECML model is defined with the aid of an *SM* as described in Fig. 16. An *SM* contains behavioral models *BMs* that are connected with external input and output ports. *BMs* are also coupled with each other through connecting input and output ports to describe the data-flow between them. ECML does not support the hierarchical composition between *BMs*, unlike *Charon* [6] and *Statecharts* [44], because this complicates mechanical verification. All *BMs* are at the same level and composed in an *SM*. In order to translate an *SM*, the translation rules translate *BMs* of the *SM* to *LHA* models and their port connections. A *SpaceEx* model is structured as a top-

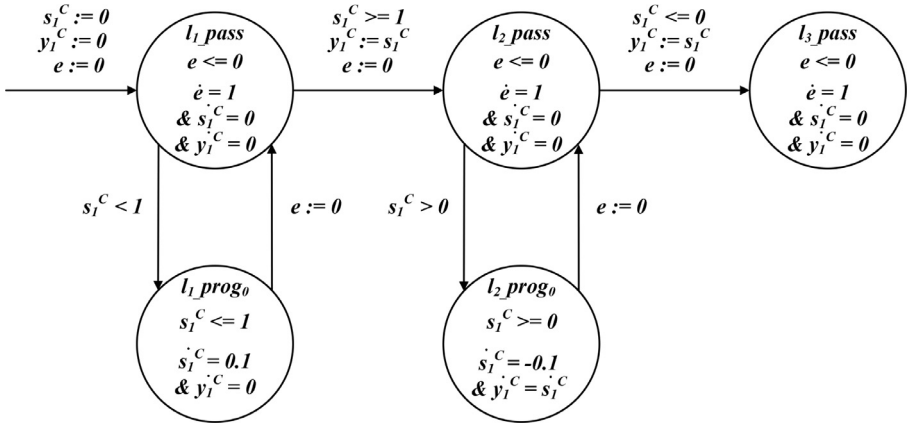


Fig. 14. *LHA* translated from the *BM* in Fig. 13.

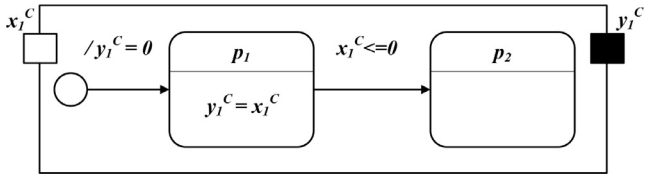


Fig. 15. Example of assignment containing input variable.

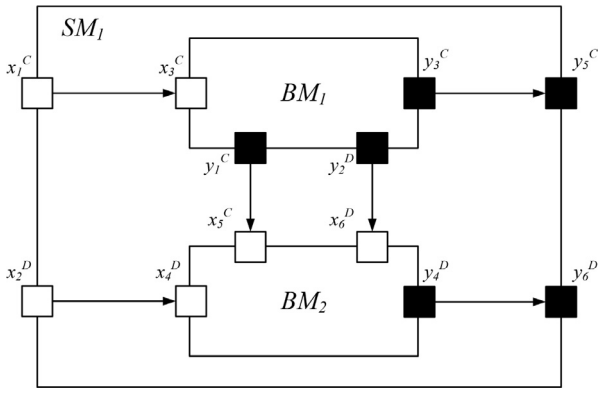


Fig. 16. Example of a structural model.

level *network component*, which can have *base components* and *network components* as children components. The relation of *network component* and *base components* is similar to the relation between an *SM* and its *BMs*. The *SM* is translated into *network component* and its *BMs* are translated into *base component* (Section 4.2.3).

3.2.1. Port

ECML uses an I/O port connection, whereas LHA uses shared variables and synchronization labels for communication (*G.5*). The translation rule translates I/O variables into variables in LHA, then changes the connected names into equivalent names. For example, y_2^D and x_6^D are connected in Fig. 16, and their names are both changed to "$y_2^D\_\_x_6^D$." Every port name used in *jump, flow, invariant*, and *init* is also changed accordingly. In our case study, we translated the ECML models into *SpaceEx* models, and it supports a port connection; thus, the names of port variables are not converted.

3.2.2. Discrete event type port

A discrete event type variable is used for sending an event, and implicitly resets to 0 after it is sent to the connected *BM*. Fig. 17(a) shows an example of a discrete event port connection. y_1^E is a discrete event output variable of BM_1, and BM_2 obtains the value through the port connection ($y_1^E \rightarrow x_1^E$). BM_1 assigns 2 to y_1^E at the (p_1, p_2) transition, and BM_2 checks the value of x_1^E at (p_3, p_4). The value of y_1^E is reset to 0 after it is sent to BM_2 as shown in Fig. 17(b). An LHA does not have discrete event variables *(G.2.2)*, therefore translation of the connection needs additional processing to reset the value.

Figs. 17(c) and (d) are the translated LHA from BM_1 and BM_2, respectively. Fig. 17(d) has two *progress locations* ($l_3\_prog_0$ and $l_3\_prog_1$) derived from the BM_2 transition. As *SpaceEx* does not support the *not equal symbol* (*e.g.*, x_1^E != 2, !(x_1^E == 2)) in modeling, we need two *progress locations* to translate a transition with an *equal symbol* (*e.g.*, x_1^E == 2 in BM_2). The translated LHA have a common *syn* $y_1^E\_\_x_1^E\_\_reset$, and Fig. 17(c) has an additional *passing location* $y_1^E\_\_x_1^E\_pass$ with an outgoing transition which resets the value of $y_1^E\_\_x_1^E$. When Fig. 17(c) remains at $y_1^E\_\_x_1^E\_pass$ and the value of $y_1^E\_\_x_1^E$ is 2, ($y_1^E\_\_x_1^E\_pass, l_2\_pass$) and ($l_3\_pass, l_4\_pass$) are executed at the same time by *syn* $y_1^E\_\_x_1^E\_\_reset$. Therefore, the event is sent to the corresponding LHA, and is reset after use.

3.3. Restrictions and assumptions

The translation rules discussed above have some assumptions and restrictions. Table 2 shows a checklist for these rules. The syntax of ECML is basically checked by *EcoPOD*, and the checklist focuses on checking assumptions and restrictions for LHA translation. The checklist is explained in previous sections; here, we summarize them. (*I.1*) notifies the *String* data type variable translation to the modeler. The process is depicted in Section 3.1.1. ECML supports user-defined data type variables and functions. However, they cannot be translated, since they are defined with the aid of the *C++* programming language (*I.2* and *I.3*). The translation covers variable assignments at *phases* except when an input variable is used in the assignment (*I.4*). The ECML model should be linear for the translation of LHA (*I.5*). Our translation strategy supports recursive variable assignments such as "$x = x + 1$" In this case, we define an additional variable to retain the previous value of x, and then perform the assignment.

We implemented this checklist in "*ECML Chekcer*" to support checking the assumptions and restrictions. '*ECML Checker*' automatically checks the checklist for an ECML model, and generates a checking report as described in Fig. 18. This report has error messages and guides for unsatisfied assumptions, and also shows which variables or phases are causing problems along with the corresponding ECML model. In this way, a modeler can modify the ECML model with support from the generated report in order to enable translation to LHA.

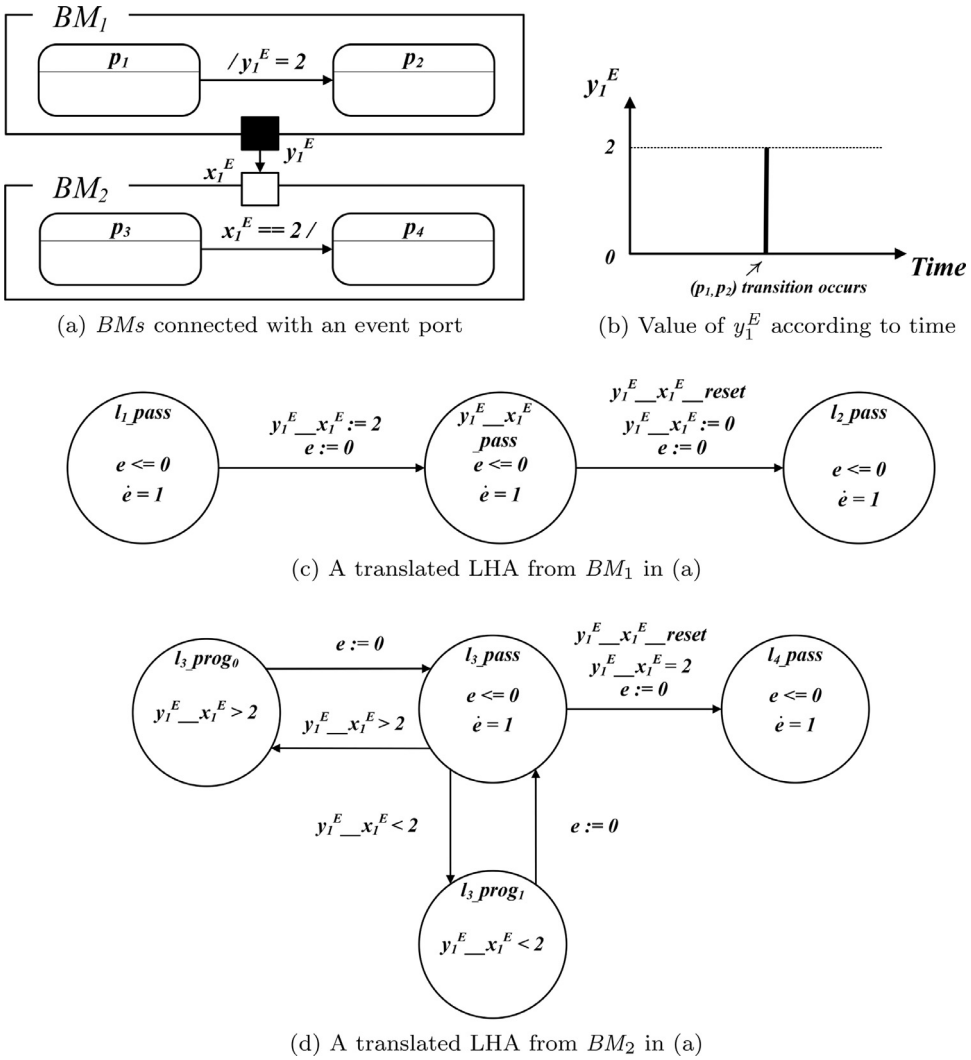


Fig. 17. An example of event port translation.

Table 2
Simplified checklist for ECML translation.

No.	Error message	Guide message	Level
I.1	String type variables are changed into *real* type.	–	Warning
I.2	Linear hybrid automata do not support user-defined data types.	Use other data types.	Error
I.3	Linear hybrid automata do not support user-defined functions.	Remove user-defined functions, and modify model.	Error
I.4	Assignment should not include an input variable at *phase*.	Use transition assignment.	Error
I.5	*Rate* function, transition function, and output function should be linear.	Use linear dynamics.	Error

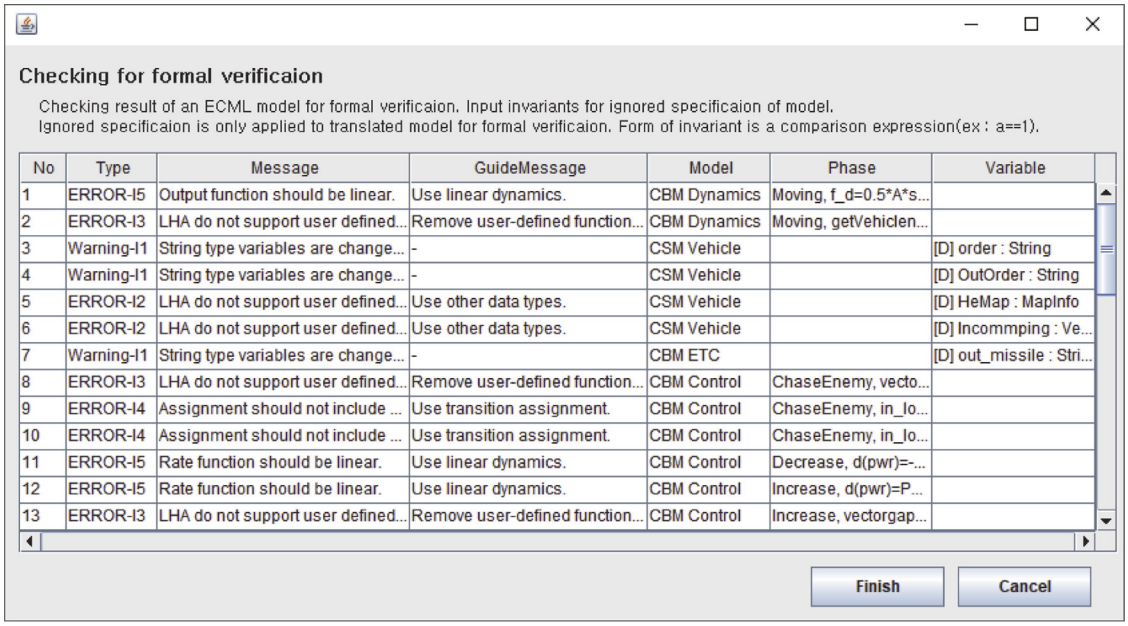
3.4. Implementing a CASE tool: ECMLtospaceex

We developed '*ECMLtoSpaceEx*' and '*ECML Checker*' using JAVA programming language, and they contain approximately 15,000 and 11,000 lines of code, respectively. We mechanized the translation rules with the support of *ANTLR* [45] and integrated it into an automatic translator, '*ECMLtoSpaceEx*.' Fig. 19 shows the process for ECML model verification. The ECML model should satisfy the assumptions and restrictions checked using '*ECML Checker*.' As previously mentioned, based on the checking report (see Fig. 18), the modeler can modify the original model to satisfy the underlying assumptions and restrictions.

'*ECMLtoSpaceEx*' reads an ECML model of a specific format and produces a LHA model that *SpaceEx* can read. The translated *SpaceEx* model is structured into two parts: the part containing the LHA models and that containing the *SpaceEx* configuration file. The LHA part includes a set of automata translated from ECML *SM* and *BM*. An expert can add invariants or an input automaton [27,29] to specify an input scenario, since ECML models obtain their input scenario from an external environment, which is part of the simulation program.

The *SpaceEx* configuration file part, on the other hand, provides a template for verification such as reachability. It consists of a verification scenario (*e.g., PHAVer, STC, LGG*), initial locations, initial states, and *forbidden states*. *SpaceEx* provides *forbidden states* to specify verification properties. If *forbidden states* are specified, *SpaceEx* shows a result that describes the intersection ($\cap$) of reachable states of the model and the *forbidden states*. This information should be manually specified by an expert.

We used a *SpaceEx* virtual server for verification. It produces a verification result with graphical and textual formats in order to support analyses by domain experts. Currently, '*ECMLtoSpaceEx*' has



No	Type	Message	GuideMessage	Model	Phase	Variable
1	ERROR-I5	Output function should be linear.	Use linear dynamics.	CBM Dynamics	Moving, f_d=0.5*A*s...	
2	ERROR-I3	LHA do not support user defined...	Remove user-defined function...	CBM Dynamics	Moving, getVehiclen...	
3	Warning-I1	String type variables are change...	-	CSM Vehicle		[D] order : String
4	Warning-I1	String type variables are change...	-	CSM Vehicle		[D] OutOrder : String
5	ERROR-I2	LHA do not support user defined...	Use other data types.	CSM Vehicle		[D] HeMap : MapInfo
6	ERROR-I2	LHA do not support user defined...	Use other data types.	CSM Vehicle		[D] Incommping : Ve...
7	Warning-I1	String type variables are change...	-	CBM ETC		[D] out_missile : Stri...
8	ERROR-I3	LHA do not support user defined...	Remove user-defined function...	CBM Control	ChaseEnemy, vecto...	
9	ERROR-I4	Assignment should not include ...	Use transition assignment.	CBM Control	ChaseEnemy, in_lo...	
10	ERROR-I4	Assignment should not include ...	Use transition assignment.	CBM Control	ChaseEnemy, in_lo...	
11	ERROR-I5	Rate function should be linear.	Use linear dynamics.	CBM Control	Decrease, d(pwr)=-...	
12	ERROR-I5	Rate function should be linear.	Use linear dynamics.	CBM Control	Increase, d(pwr)=P...	
13	ERROR-I3	LHA do not support user defined...	Remove user-defined function...	CBM Control	Increase, vectorgap...	

Fig. 18. Screen dump of the checking report generated by *ECML Checker*.

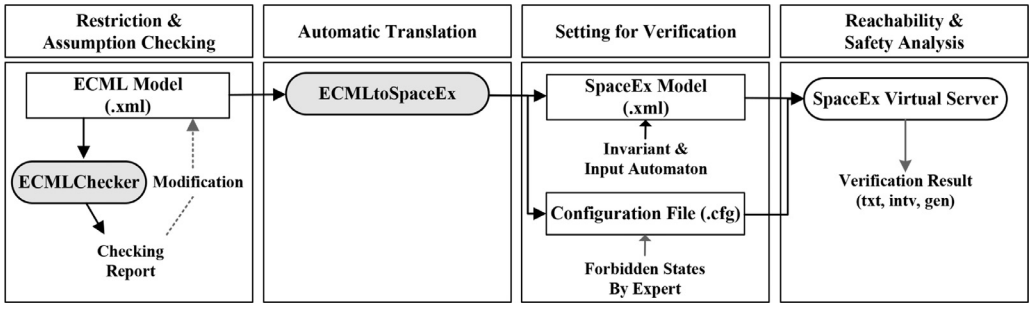


Fig. 19. ECML verification process with SpaceEx.

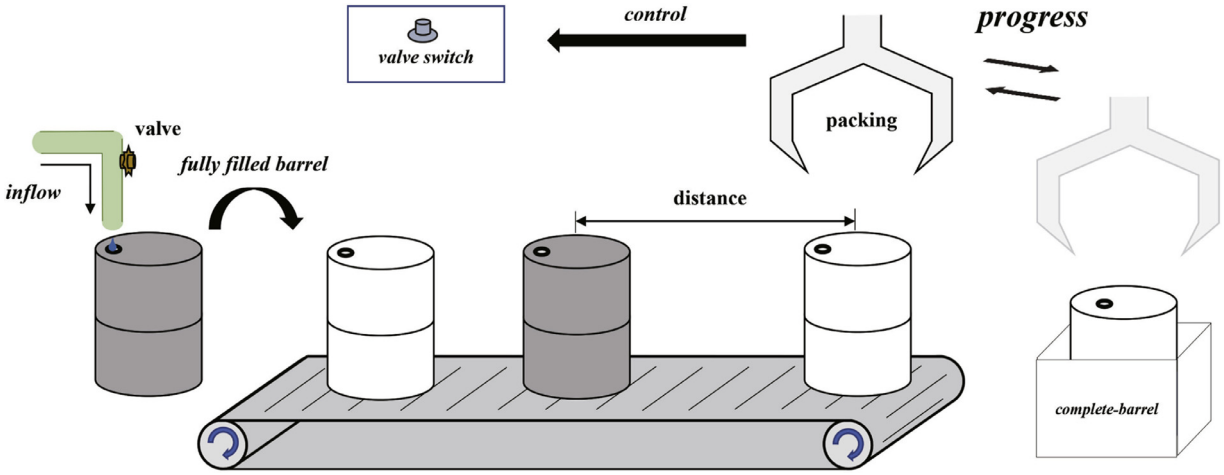


Fig. 20. Sketch of the *Barrel Production System*.

no graphical interface, since it was embedded in the ECML modeling tool, *EcoPOD*.

3.5. Further considerations on the translation

ECML translation into nonlinear hybrid automata. Our translation rules use *passing locations* and *progress locations* to describe deterministic transition in hybrid automata. The translation has two preconditions for deterministic transition translation. The behavior of all variables should be linear, and *rate* should not be defined as a *range formula including zero* such as "$\dot{s^C} \in [-1, 1]$" in the original ECML model. If these preconditions are not satisfied, the obtained LHA could remain in a *progress location*, even when the corresponding guard is satisfied only once. A zero *rate* such as "$\dot{s^C} = 0$"

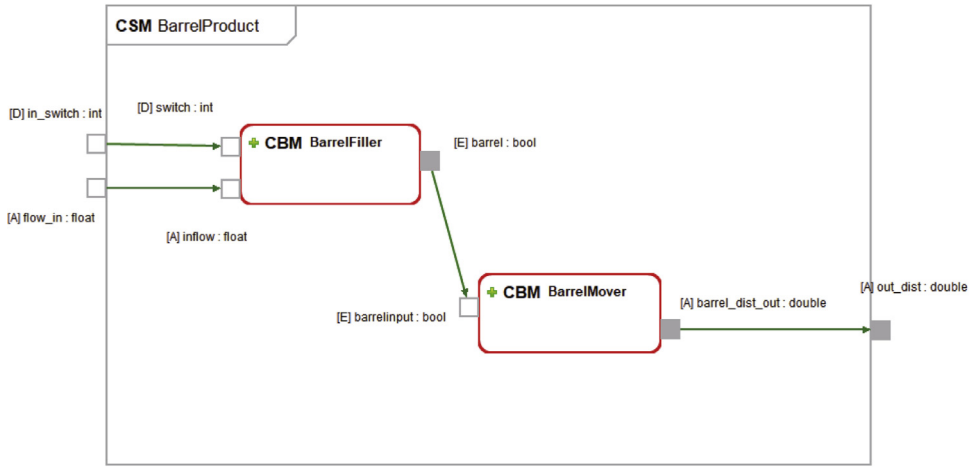


Fig. 21. The ECML *SM* for *Barrel Production System*.

is not an issue with respect to this precondition. If an ECML model has this *rate*, the translated model will behave as the modeler intended. Since the linearity of the original ECML model is verified and ensured by the '*ECML Checker*', and ECML does not support the definition of *rate* based on ranges, the two aforementioned preconditions are satisfied. We still need to refine the translation rules and find a suitable analysis algorithm & supporting tool for non-linear hybrid automata translation and verification.

Formal definition of the translation. Formal definition of the proposed translation rules is strongly required to demonstrate, verify or prove its functional correctness thoroughly. We are currently working on this topic, but it's challenging, as other literature reports [46–51], to provide formal, semi-formal, and graphical translation rules in different formats.

Functional correctness of the translation. Verifying the correctness of compilers, code generators and translators is one of the most difficult topics in computing research [52]. Research to verify the correctness through various techniques has been reported (see [53,54] for a survey). We are planning to apply compiler/translator verification techniques to assess the correctness of the proposed translation rules.

4. Case study: A Barrel production system

This section models a '*Barrel Production System*' with ECML, and translates it into LHA. Section 4.1 briefly explains the target system - '*Barrel Production System*,' structured with the subsystems '*Barrel-Filler System*,' and '*Barrel-Mover System*.' In Section 4.2, the considered subsystems are modeled as ECML *BMs* and translated into LHA. The whole system is modeled as an ECML *SM* and translated into a *network component* of *SpaceEx*. All LHA based models are generated mechanically from the ECML models of the '*Barrel Production System*' by the CASE tool '*ECMLtoSpaceEx*,' and read by SpaceEx to perform formal verifications such as safety verification and reachability analysis (Section 4.3). The translation with '*ECMLtoSpaceEx*' takes 0.495 s and 18.985 MB RAM (mean value of 100 runs), with the '*Barrel Production System*.' We performed the translation and analysis on a 2.5 GHz processor with 8 GB memory.

4.1. Overview of the model

Fig. 20 describes the '*Barrel Production System*.' It has three subsystems: '*Barrel-Filler System*,' '*Barrel-Mover System*' and '*Barrel-Packing System*.' The '*Barrel-Filler System*' fills a barrel with a specific *inflow* and places the fully filled barrel on the conveyor belt. The '*Barrel-Mover System*' then moves the fully filled barrel to a specific point to pack it out. A conveyor belt transmits the barrel at a certain velocity from the barrel-filler system to the packing point. The distance from the barrel to the packing point is calculated continuously. The fully filled barrel is finally packed. The case study originated in [5], but we modified and extended it with the barrel-mover system and the barrel-packing system.

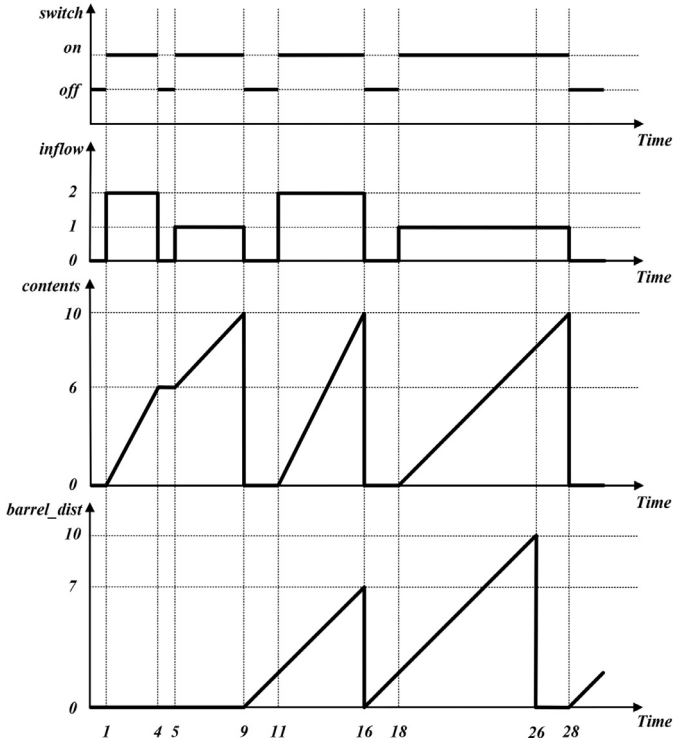


Fig. 22. Example behavior for the *Barrel Production System*.

The '*Barrel Production System*' is modeled as an ECML *SM* as shown in Fig. 21. It has two input variables and one output variable. As mentioned in Section 2.1, [*A*] (Analog) depicts continuous values in *EcoPOD*. It also has two *BMs* and they are connected via a discrete event variable *barrel*, which indicates production of a fully filled barrel. Fig. 22 shows an example behavior for the *Barrel Production System*. When *switch* is *on*, continuous input *inflow* increases the level of liquid in a barrel (*i.e., contents*). When time is 9, a fully filled barrel is produced and replace by a new empty barrel while *switch* is *off*, then the conveyor belt starts moving the barrel.

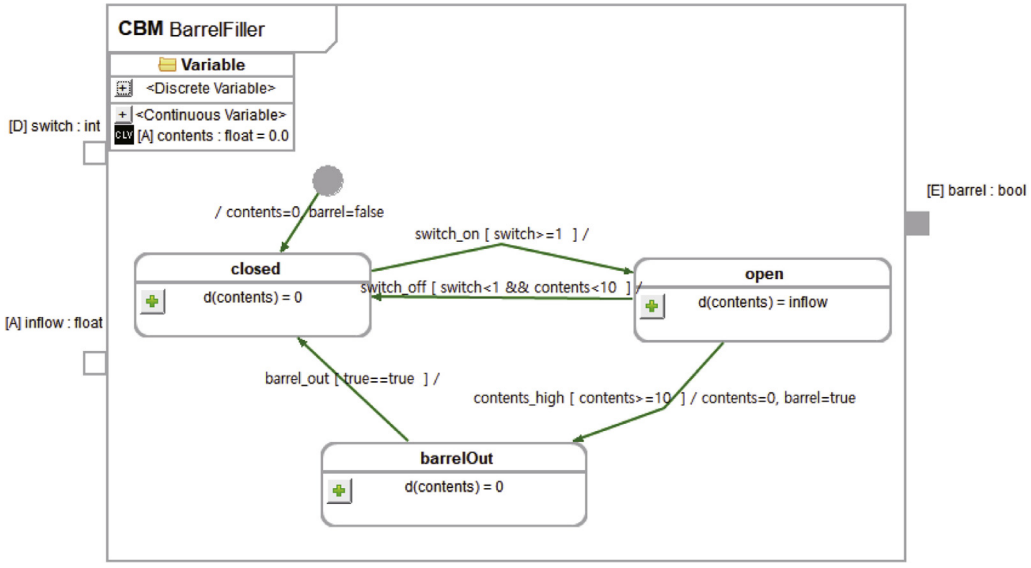


Fig. 23. An ECML *BM* for the *Barrel-Filler System* (BM_{BF}).

A continuous state, *barrel_dist* starts to increase time 9, and the conveyor belt stops when "*barrel_dist* == 10". If a new fully filled barrel is produced while another barrel is being moved, *barrel_dist* is reset to 0 to move the fully filled barrel that arrived last. A new fully filled barrel is produced at time 16, hence *barrel_dist* is reset at this time, then increases again.

4.2. Translation of ECML models

This section translates *BMs* for *Barrel-Production System* according to the translation rules presented in Section 3.

4.2.1. Barrel-Filler system

Fig. 23 is an ECML *BM* for the *Barrel-Filler System* (BM_{BF}). It has two input ports, one output port and one state variable. *contents* is a continuous state variable, describing the level of liquid in a barrel. *switch* is a discrete input, which can open and close the valve. When *switch* is set to open, a continuous input (*inflow*) increases *contents*. *barrel* is a discrete event output indicating that a barrel is completely filled.

The *Barrel-Filler System* fills a barrel and puts it out whenever the barrel is filled up to a specific water level. The ECML *BM* has three phases: *closed, open*, and *barrelOut*. The initial phase is *closed*, and the value of *contents* is not changed in this phase. The system fills a barrel in the *open* phase. If *contents* is greater than or equal to 10, the continuous state variable *contents* is reset to 0 and a barrel is produced (*i.e., barrel* = *true*) by the state transition labeled *contents_high*. The transition condition '*true*==*true*' is trivially true, hence the phase *barrelOut* changes automatically to *closed*. This model assumes that a new empty barrel starts filling as soon as a fully filled barrel is produced as in the original example in [5]. The formal definition of the basic ECML model *BM* for the '*Barrel-Filler System*' (BM_{BF}) is as follows in Fig. 24.

The state variables (S^C, S^D) in ECML are translated into the local variables of subsystem in a *SpaceEx* model. Output variables (Y) and state variables (S^D, S^C) are translated into *controlled* variables. A *flow* for a *controlled* variable should be defined in the subsystem. The translation by '*ECMLtoSpaceEx*' considers this property, thus *flows* are defined for local variables and output variables, except input variables ($I = X \setminus C$). In Fig. 25, every location in LHA_{BF} has a *flow* for *contents* and *barrel*, which are a local and output variables, respectively, but does not have a *flow* for *switch*, an input variable. A special variable *e* is used in *passing locations* (Section 3.1.5). This variable is a *local* variable, but *flow* is omitted for *e* at *progress locations*. *e* is assigned a value of '0' at ingoing and outgoing transitions of passing locations, thus it retains the '0' value at *passing locations*.

'*ECMLtoSpaceEx*' mechanically generated a LHA model in *xml* format from the single ECML model BM_{BF} in Fig. 23. Fig. 25 shows the LHA model, as it can be seen in the *SpaceEx* model editor. The translated *init* information is described in a configuration file (*cfg* format), since *SpaceEx* uses it to specify initial locations and initial values for all variables of each subsystem. The LHA_{BF} functions as follows: the initial location is *closed_pass*. If the switch is turned on (*i.e., switch* >= 1) at the *closed_pass* location, it transits to *open_pass*, if not, it transits to *closed_prog0* immediately. At *closed_prog0*, it waits until the switch is turned on. If the model transits to *open_pass*, it checks the values of the *switch* and *contents*. When the switch is turned off (*i.e., switch* < 1), it returns to *closed_pass*. Otherwise, if the switch is on and *contents* < 10, it transits to *open_prog*0, and then *contents* increases according to $contents' == inflow$. When *contents* increases to 10, the model transits to *barrel__barrelinput_pass0*. The source *BM* in Fig. 23 has a transition (*open, barrelOut*), however the LHA has one more location *barrel__barrelinput_pass0* for the discrete event variable translation (Section 3.2.2). It transits to *closed_pass* through *barrelOut_pass* after resetting the discrete event variable. The formal definition of the linear hybrid automaton LHA_{BF} for the *Barrel-Filler System* is as follows in Fig. 26.

4.2.2. Barrel-Mover system

Fig. 27 is an ECML *BM* for the *Barrel-Mover System*. It has one input port, one output port and one state variable. *barrel_dist* is a continuous state variable, describing the distance from the *Barrel-Filler System* to the current position of a barrel on the conveyor belt. *barrelinput* is a discrete event input, indicating an incoming fully filled barrel. *barrel_dist_out* is a continuous output variable for the *barrel_dist*. The *BM* moves a fully filled barrel to a specific location (*i.e.,* the point of packing, *barrel_dist* >= 10). As soon as the *barrel-filler system* produces a *barrel*, the *Barrel-Mover System* obtains it through the connected port *barrel_input* as defined in Fig. 21. The initial phase is *stopped*, and when a fully filled barrel arrives, the system transits to *working* and *barrel_dist* is increased. If a new fully filled barrel arrives while the previous barrel is mov-

$BM_{BF} = \langle X,\ Y,\ S,\ Init,\ Cond^E,\ Trans^E,\ Out^E,\ Cond^S,\ Trans^S,\ Out^S,\ Out^C,\ Out^D,\ Rate \rangle$

- $X^C = \{inflow \mid inflow = flow\_in\}$
- $X^D = \{switch \mid switch = in\_switch\}$
- $Y^E = \{barrel \mid barrel \in \{true, false\}\}$
- $P = \{closed, open, barrelOut\}$
- $S^C = \{contents \mid contents \in \mathbb{R}\}$
- $Init = \{P = closed, contents = 0, barrel = false\}$
- $Cond^E$:
  $(P = closed, switch \geq 1)$
  $(P = open, switch < 1 \wedge contents < 10)$
- $Trans^E$:
  $(P = closed, switch \geq 1) \mapsto (P = open)$
  $(P = open, switch < 1 \wedge contents < 10) \mapsto (P = closed)$
- Out^E: $\emptyset$
- $Cond^S$:
  $(P = open, contents \geq 10)$
  $(P = barrelOut)$
- $Trans^S$:
  $(P = open, contents \geq 10) \mapsto (P = barrelOut, contents = 0)$
  $(P = barrelOut) \mapsto (P = closed)$
- Out^S:
  $(P = open, contents \geq 10) \mapsto (barrel = true)$
- Out^C: $\emptyset$
- Out^D: $\emptyset$
- $Rate$:
  $(P = \{closed, barrelOut\}) \mapsto (d(contents) = 0)$
  $(P = open) \mapsto (d(contents) = inflow)$

Fig. 24. The formal definition of BM_{BF} for the '*Barrel-Filler System*'.

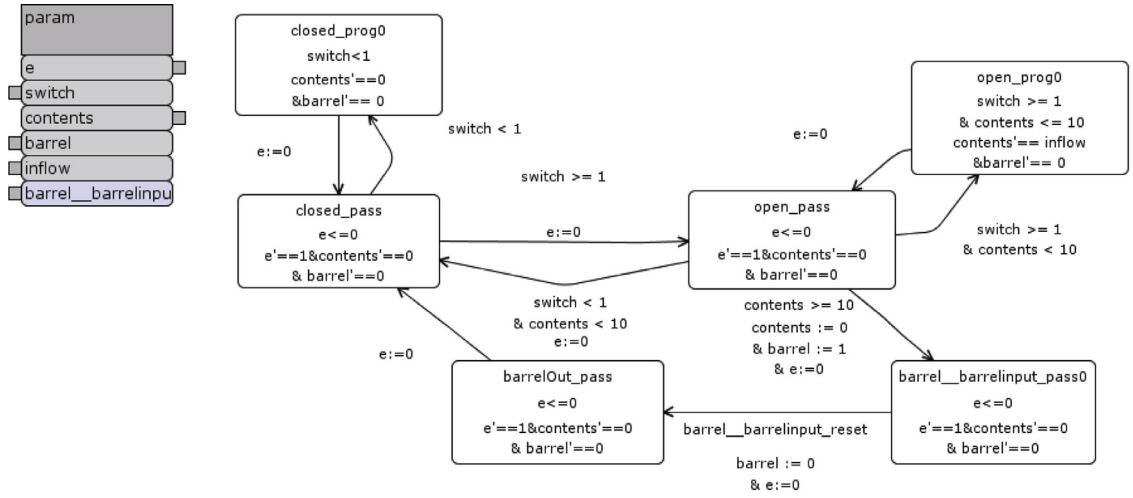


Fig. 25. *LHA* model translated from Fig. 23 (in the *SpaceEx* model editor).

ing, it proceeds to the *checkbarrel* phase and reset *barrel_dist* to move the last input barrel. It returns to the initial phase when all barrels have been moved to a specific point (*i.e., barrel_dist* >= 10). The formal definition of BM_{BM} for the *Barrel-Mover System* is as follow in Fig. 28.

Fig. 29 shows a linear hybrid automaton model LHA_{BM} generated from Fig. 27. The initial state of LHA_{BM} is *stopped_pass*, and the model transits to *stopped_prog0* or *stopped_prog1*. $Cond^E(P = stopped, barrelinput = true)$ in Fig. 27 has equal symbol and its negation is not supported in *SpaceEx*, thus the translation uses two progress locations, *stopped_prog0* and *stopped_prog1* (Section 3.2.2). In each location, it waits for a fully filled barrel event. If a fully filled barrel is placed on the conveyor belt (*i.e., barrelinput* == 1), the model transits to *working_prog0* or *working_prog1* through *stopped_pass* and *working_pass*. In order to reset *berrelinput*, *jump(stopped_pass, working_pass)* has a *syn* label, which is the same as (*barrel__barrelinput_pass0, barrelOut_pass*) in Fig. 25. At *working_prog0* and *working_prog1*, *barrel_dist* increases according to *flow barrel_dist'*== 1 to describe the current position of the last produced full-filled barrel. The output variable *barrel_dist_out* outputs the value of *barrel_dist* simultaneously. If a new fully filled barrel arrives while the conveyor belt is moving, the model transits to *checkbarrel_pass*, and then to *barrel_dist_pass1*. *barrel_dist_out_pass0* and *barrel_dist_out_pass1* are *passing locations* for sequential variable assignment (Section 3.1.6). The transitions (*working, stopped*) and (*checkbarrel, working*) reset *barrel_dist* in Fig. 27; however, the translated model would have *barrel_dist* := 0 ∧ *barrel_dist_out* := *barrel_dist* in a *jump*, because of the assignment to *barrel_dist_out* at phase *working* (Section 3.1.7). In order to model the same behavior, the transition from *checkbarrel_pass* to *barrel_dist_out_pass*1 resets *barrel_dist*, and then *jump(barrel_dist_out_pass1, working_pass)* outputs the up-

$LHA_{BF} = \langle\ X,\ V,\ flow,\ inv,\ init,\ E,\ jump,\ \Sigma,\ syn\ \rangle$

- $I = \{switch, inflow\}$
- $C = \{e,\ contents, barrel\}$
- $O = \{barrel\}$
- $V = \{closed\_pass, closed\_prog0, open\_pass,\ open\_prog0, barrel\_\_barrelinput\_pass0,$ $barrelOut\_pass\}$
- $flow(V)$:
  $(V = \{closed\_pass, open\_pass, barrel\_\_barrelinput\_pass0, barrelOut\_pass\})$
  $\mapsto (\dot{e} = 1 \wedge \dot{contents} = 0 \wedge \dot{barrel} = 0)$
  $(V = closed\_prog0) \mapsto (\dot{contents} = 0 \wedge \dot{barrel} = 0)$
  $(V = open\_prog0) \mapsto (\dot{contents} = inflow \wedge \dot{barrel} = 0)$
- $inv(V)$:
  $(V = \{closed\_pass\ open\_pass, barrel\_\_barrelinput\_pass0, barrelOut\_pass\}) \mapsto (e \leq 0)$
  $(V = closed\_prog0) \mapsto (switch < 1)$
  $(V = open\_prog0) \mapsto (switch \geq 1 \wedge contents \leq 10)$
- $init(V)$:
  $(V = closed\_pass) \mapsto (e = 0 \wedge contents = 0 \wedge barrel = 0)$
  $(V = \{closed\_prog0, open\_pass, open\_prog0, barrel\_\_barrelinput\_pass0, barrelOut\_pass\})$
  $\mapsto false$
- $E = \{(closed\_pass, closed\_prog0), (closed\_prog0, closed\_pass), (closed\_pass, open\_pass)$
  $(open\_pass, closed\_pass), (open\_pass, open\_prog0), (open\_prog0, open\_pass),$
  $(open\_pass, barrel\_\_barrelinput\_pass0), (barrel\_\_barrelinput\_pass0, barrelOut\_pass),$
  $(barrelOut\_pass, closed\_pass)\}$
- $jump(V, V)$:
  $(closed\_pass, closed\_prog0) \mapsto (switch < 1)$
  $(closed\_prog0, closed\_pass) \mapsto (e := 0)$
  $(closed\_pass, open\_pass) \mapsto (switch \geq 1 \wedge e := 0)$
  $(open\_pass, closed\_pass) \mapsto (switch < 1 \wedge contents < 10 \wedge e := 0)$
  $(open\_pass, open\_prog0) \mapsto (switch \geq 1 \wedge contents < 10)$
  $(open\_prog0, open\_pass) \mapsto (e := 0)$
  $(open\_pass, barrel\_\_barrelinput\_pass0)$
  $\mapsto (contents \geq 10 \wedge contents := 0 \wedge barrel := 1 \wedge e := 0)$
  $(barrel\_\_barrelinput\_pass0, barrelOut\_pass) \mapsto (barrel := 0 \wedge e := 0)$
  $(barrelOut\_pass, closed\_pass) \mapsto (e := 0)$
- $\Sigma = \{barrel\_\_barrelinput\_reset\}$
- $syn(e, jump(e))$:
  $((barrel\_\_barrelinput\_pass0, barrelOut\_pass), true) \mapsto (barrel\_\_barrelinput\_reset)$

Fig. 26. The formal definition of LHA_{BF} for the '*Barrel-Filler System*'.

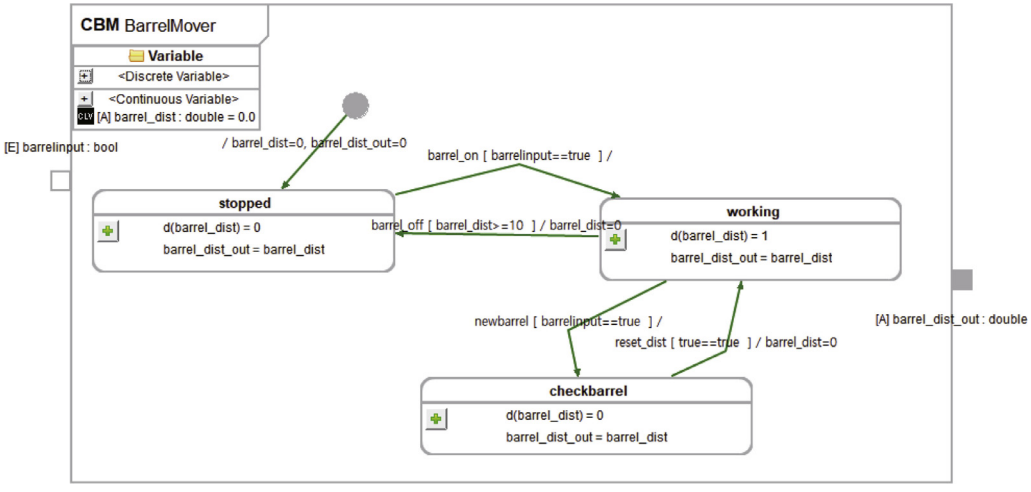


Fig. 27. ECML *BM* for *Barrel-Mover System* (BM_{BM}).

dated and correct value, besides returning to *working_pass*. The LHA transits to *stopped_pass* through *barrel_dist_out_pass0* when all barrels have been moved to the packing position (*i.e.*, *barrel_dist* >= 10). The formal definition of the linear hybrid automaton LHA_{BM} for the *barrel moving system* is as follows in Fig. 30.

4.2.3. Translation of the structural ECML model

Barrel Production System is originally modeled as an ECML *SM*, which is composed of two *BMs* as shown in Fig. 21. The *SM* is translated into a *network component*, and it has *base components*, which are translated from the *BMs*. Fig. 31 is a *network component*

$BM_{BM} = \langle X,\ Y,\ S,\ Init,\ Cond^E,\ Trans^E,\ Out^E,\ Cond^S,\ Trans^S,\ Out^S,\ Out^C,\ Out^D,\ Rate \rangle$

- $X^E = \{barrelinput \mid barrelinput = barrel\}$
- $Y^C = \{barrel\_dist\_out \mid barrel\_dist\_out \in \mathbb{R}\}$
- $P = \{stopped, working, checkbarrel\}$
- $S^C = \{barrel\_dist \mid barrel\_dist \in \mathbb{R}\}$
- $Init = \{P = stopped, barrel\_dist = 0, barrel\_dist\_out = 0\}$
- $Cond^E$:
  $(P = stopped, barrelinput = true)$
  $(P = working, barrelinput = true)$
- $Trans^E$:
  $(P = stopped, barrelinput = true) \mapsto (P = working)$
  $(P = working, barrelinput = true) \mapsto (P = checkbarrel)$
- Out^E: $\emptyset$
- $Cond^S$:
  $(P = working, barrel\_dist \geq 10)$
  $(P = checkbarrel)$
- $Trans^S$:
  $(P = working, barrel\_dist \geq 10) \mapsto (P = stopped, barrel\_dist = 0)$
  $(P = checkbarrel) \mapsto (P = working, barrel\_dist = 0)$
- Out^S: $\emptyset$
- Out^C:
  $(P = \{stopped, working, checkbarrel\}) \mapsto (barrel\_dist\_out = barrel\_dist)$
- Out^D: $\emptyset$
- $Rate$:
  $(P = \{stopped, checkbarrel\}) \mapsto (d(barrel\_dist) = 0)$
  $(P = working) \mapsto (d(barrel\_dist) = 1)$

Fig. 28. The formal definition of BM_{BM} for the '*Barrel-Mover System*'.

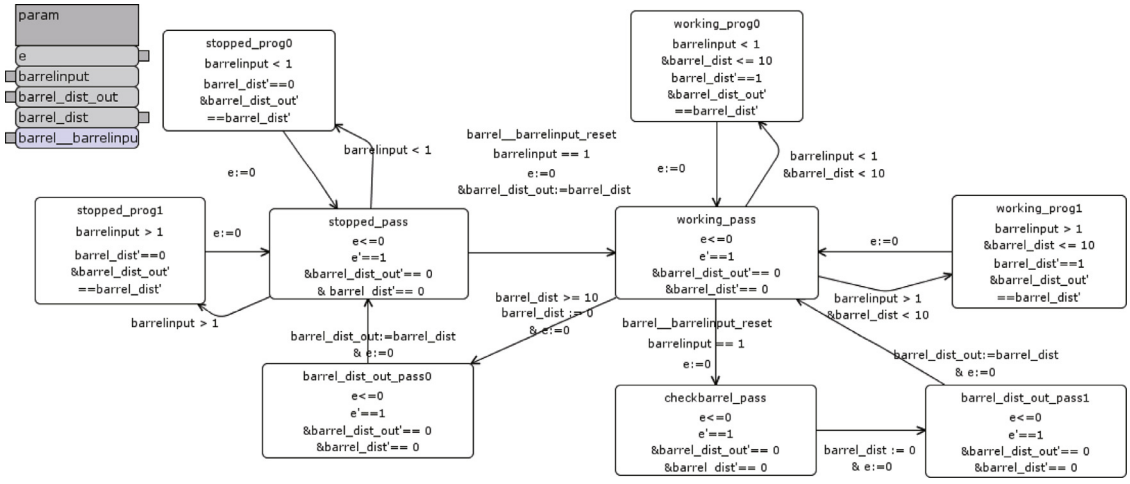


Fig. 29. A LHA model translated from Fig. 27 (in *SpaceEx* model editor).

for the *Barrel Production System*. It has four base components. *BarrelProduct_BarrelFiller* and *BarrelProduct_BarrelMover* are defined as shown in Fig. 25 and Fig. 29, respectively. *timer* is a base component to support the analysis of the verification result and details of *timer* are depicted in Section 4.3.

BarrelProduct_Input_1 is an input automaton for verification. We manually created the input automaton for verification based on Fig. 22, since ECML obtains the input scenario from an external environment, which is a part of the simulation program. Fig. 32 shows this input automaton. It generates an input scenario where *in_switch* and *flow_in* evolve as described in Fig. 22. The transitions of the input automaton works deterministically. It starts to generate *flow_in* and the value is sent to *Barrel-Filler System*. When time is 9, a fully filled barrel is produced in *Barrel-Filler System*, besides that the input automaton does not generate input. After 2 time units it generates input again.

4.3. Verification of the translated LHA

SpaceEx obtains a model file and a configuration file as inputs, and generates the verification result with textual and graphical formats [55]. A configuration file defines the initial states of the model, verification scenario, forbidden states, and other options for verification. This file can be modified in the *SpaceEx* web interface. We chose *PHAVer* as verification scenario; therefore, we need to manually set the *initial states, forbidden states, Max. iteration*, and *output format. Initial states* define the initial locations of subsystems and initial values of variables. *Max. iteration* refers to the maximum number of iterations for the reachability algorithm, which is the total number of discrete post computations on symbolic states [12]. An appropriate number should be set for *Max. iteration*, since it could induce the state explosion problem if it is too large, or an inaccurate verification result when it is too

$LHA_{BM} = \langle\ X,\ V,\ flow,\ inv,\ init,\ E,\ jump,\ \Sigma,\ syn\ \rangle$

- $I = \{barrelinput\}$
- $C = \{e,\ barrel\_dist, barrel\_dist\_out\}$
- $O = \{barrel\_dist\_out\}$
- $V = \{stopped\_pass, stopped\_prog0, stopped\_prog1, working\_pass, working\_prog0, working\_prog1, checkbarrel\_pass, barrel\_dist\_out\_pass0, barrel\_dist\_out\_pass1\}$
- $flow(V)$:
  $(V = \{stopped\_pass, working\_pass, checkbarrel\_pass, barrel\_dist\_out\_pass0, barrel\_dist\_out\_pass1\}) \mapsto (\dot{e} = 1 \wedge \dot{barrel\_dist} = 0 \wedge \dot{barrel\_dist\_out} = 0)$
  $(V = \{stopped\_prog0, stopped\_prog1\}) \mapsto (\dot{barrel\_dist} = 0 \wedge \dot{barrel\_dist\_out} = \dot{barrel\_dist})$
  $(V = \{working\_prog0, working\_prog1\}) \mapsto (\dot{barrel\_dist} = 1 \wedge \dot{barrel\_dist\_out} = \dot{barrel\_dist})$
- $inv(V)$:
  $(V = \{stopped\_pass, working\_pass, checkbarrel\_pass, barrel\_dist\_out\_pass0, barrel\_dist\_out_pass1\}) \mapsto (e \leq 0)$
  $(V = stopped\_prog0) \mapsto (barrelinput < 1)$
  $(V = stopped\_prog1) \mapsto (barrelinput > 1)$
  $(V = working\_prog0) \mapsto (barrelinput < 1 \wedge barrel\_dist \leq 10)$
  $(V = working\_prog1) \mapsto (barrelinput > 1 \wedge barrel\_dist \leq 10)$
- $init(V)$:
  $(V = stopped\_pass) \mapsto (e = 0, barrel\_dist = 0 \wedge barrel\_dist\_out = 0)$
  $(V = \{stopped\_prog0, stopped\_prog1, working\_pass, working\_prog0, working\_prog1, checkbarrel\_pass, barrel\_dist\_out\_pass0, barrel\_dist\_out\_pass1\}) \mapsto false$
- $E = \{(stopped\_pass, stopped\_prog0), (stopped\_prog0, stopped\_pass), (stopped\_pass, stopped\_prog1), (stopped\_prog1, stopped\_pass), (stopped\_pass, working\_pass), (working\_pass, working\_prog0), (working\_prog0, working\_pass), (working\_pass, working\_prog1), (working\_prog1, working\_pass), (working\_pass, barrel\_dist\_out\_pass0), (barrel\_dist\_out\_pass0, stopped\_pass), (working\_pass, checkbarrel\_pass), (checkbarrel\_pass, barrel\_dist\_out\_pass1), (barrel\_dist\_out\_pass1, working\_pass)\}$
- $jump(V, V)$:
  $(stopped\_pass, stopped\_prog0) \mapsto (barrelinput < 1)$
  $(stopped\_prog0, stopped\_pass) \mapsto (e := 0)$
  $(stopped\_pass, stopped\_prog1) \mapsto (barrelinput > 1)$
  $(stopped\_prog1, stopped\_pass) \mapsto (e := 0)$
  $(stopped\_pass, working\_pass) \mapsto (barrelinput = 1 \wedge barrel\_dist\_out := barrel\_dist \wedge e := 0)$
  $(working\_pass, working\_prog0) \mapsto (barrelinput < 1 \wedge barrel\_dist < 10)$
  $(working\_prog0, working\_pass) \mapsto (e := 0)$
  $(working\_pass, working\_prog1) \mapsto (barrelinput > 1 \wedge barrel\_dist < 10)$
  $(working\_prog1, working\_pass) \mapsto (e := 0)$
  $(working\_pass, barrel\_dist\_out\_pass0) \mapsto (barrel\_dist \geq 10 \wedge barrel\_dist := 0 \wedge e := 0)$
  $(barrel\_dist\_out\_pass0, stopped\_pass) \mapsto (barrel\_dist\_out := barrel\_dist \wedge e := 0)$
  $(working\_pass, checkbarrel\_pass) \mapsto (barrelinput = 1 \wedge e := 0)$
  $(checkbarrel\_pass, barrel\_dist\_out\_pass1) \mapsto (barrel\_dist := 0 \wedge e := 0)$
  $(barrel\_dist\_out\_pass1, working\_pass) \mapsto (barrel\_dist\_out := barrel\_dist \wedge e := 0)$
- $\Sigma = \{barrel\_\_barrelinput\_reset\}$
- $syn(e, jump(e))$:
  $((stopped\_pass, working\_pass), barrelinput = 1) \mapsto (barrel\_\_barrelinput\_reset)$
  $((working\_pass, checkbarrel\_pass), barrelinput = 1) \mapsto (barrel\_\_barrelinput\_reset)$

Fig. 30. The formal definition of LHA_{BM} for the '*Barrel-Mover System*'.

small. If *forbidden states* are provided, *SpaceEx* intersects the *forbidden states* with system states, to verify whether these states are reachable. However, *forbidden states* are not adequate to check the sequence of states such as a reachability analysis using temporal logics [56,57]. We carefully specified the *forbidden states* and conducted timed-based analysis with support of the *timer* component.

We performed a reachability analysis [58] for one safety requirement and two reachability requirements using *SpaceEx*.

Safety Requirement 1 "*The content of a barrel should not exceed 10 liters*".

Reachability Requirement 1 "*A fully filled barrel is moved to the end of the conveyor belt*".

Reachability Requirement 2 "*The conveyor belt starts to move after the first fully filled barrel arrives*".

Table 3 shows the configurations for the reachability analysis. *Scenario, Initial States*, and *Max. Iteration* are the common ele-

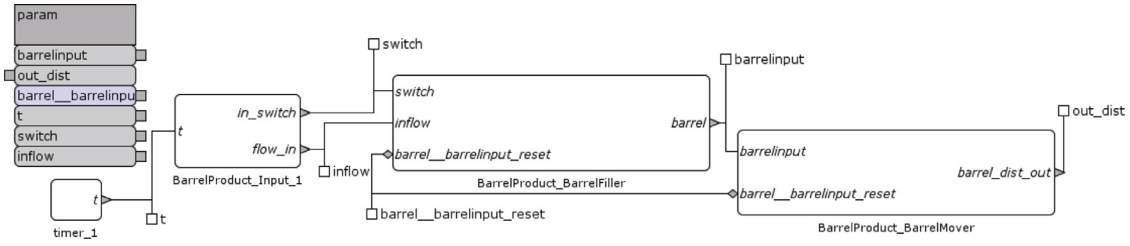


Fig. 31. A *network component* obtained from the *Barrel Production System* (in the *SpaceEx* model editor).

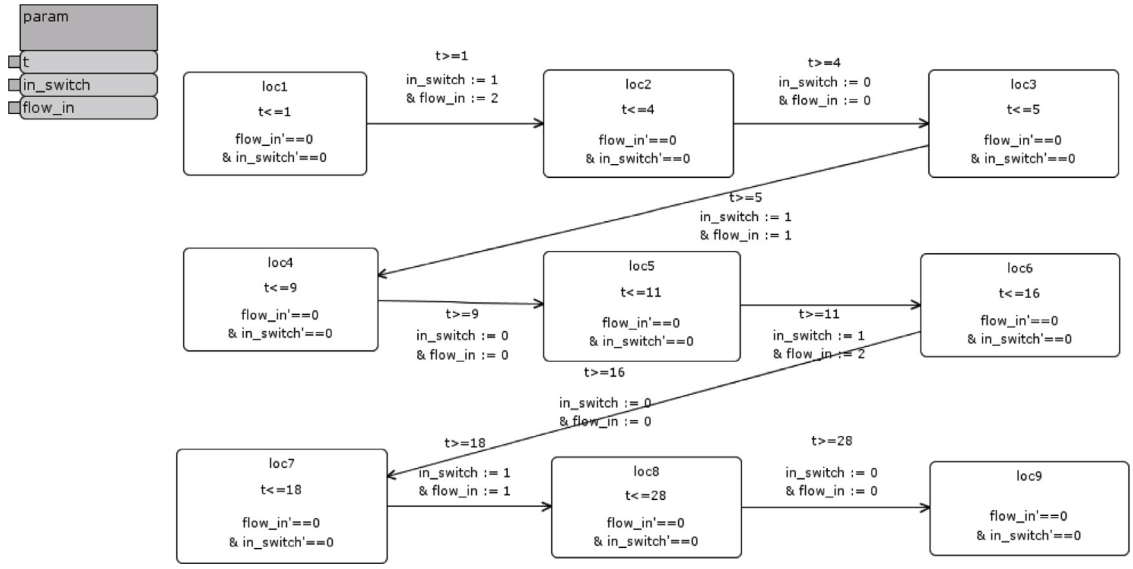


Fig. 32. Input automaton for verification (in the *SpaceEx* model editor).

Table 3
SpaceEx configurations for case study.

Scenario	PHAVer	
Initial States	*BarrelProduct_BarrelMover.e* == 0&*barrelinput* == 0&*out_dist* == 0 &*barrel_dist* == 0&*t* == 0&*BarrelProduct_BarrelFiller.e* == 0 &*switch* == 0&*contents* == 0&*inflow* == 0 &*loc*(*BarrelProduct_BarrelMover*) == *stopped_pass* &*loc*(*BarrelProduct_BarrelFiller*) == *closed_pass* &*loc*(*BarrelProduct_Input_*1) == *loc*1	
Max. Iteration	−1	
Requirement	Forbidden states	Output
Safety1	*contents* > 10	Text(.txt)
Reachability1	(*loc*(*BarrelProduct_BarrelMover*) == *working_prog*0 |*loc*(*BarrelProduct_BarrelMover*) == *working_prog*1)&*barrel_dist* == 10	Text(.txt)
Reachability2		2D graph(.gen)/t, contents, barrelinput,*barrel_dist*

Table 4
Runtime and memory usage of *SpaceEx* reachability analyses.

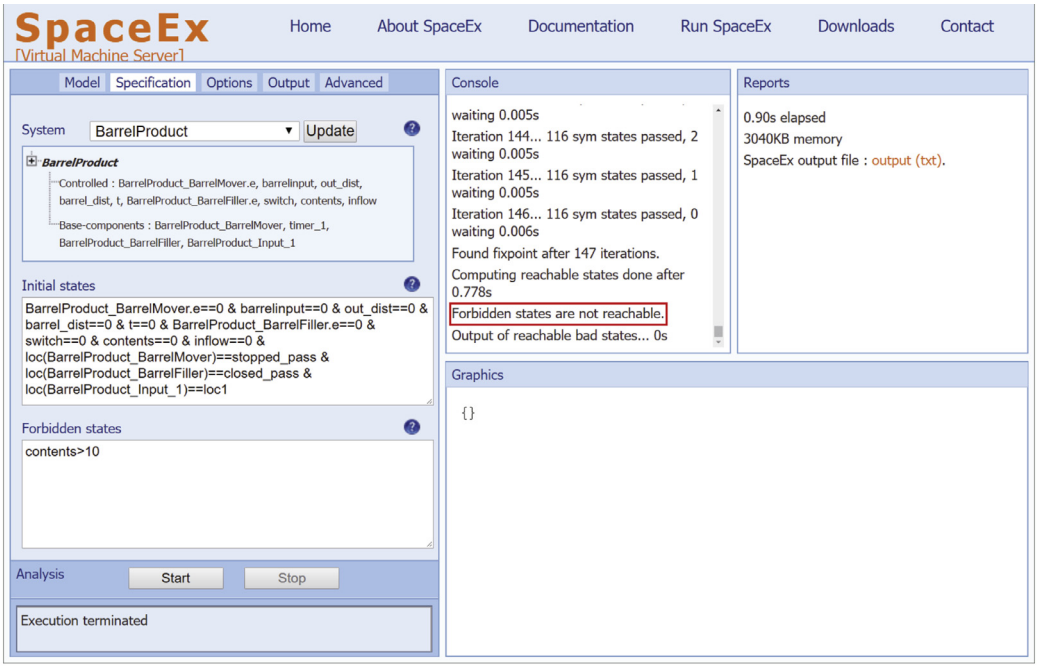
Requirement	Memory		Time	
	Mean (kb)	Deviation	Mean (s)	Deviation
Safety1	3040	0	0.868	0.0024
Reachability1	3040	0	0.850	0.0030
Reachability2	3072	0	1.255	0.0025

ments of the three reachability analyses and we set *forbidden states* for each requirement. Initial states are mechanically generated by '*ECMLtoSpaceEx*' from the source ECML model, and *Max. iteration* is set as '−1,' which means that '*the reachability analysis only terminates if a fixed point is found.*' The runtime and memory usage of the reachability analyses are described in Table 4. We performed each reachability analysis 100 times on a *SpaceEx* virtual server.

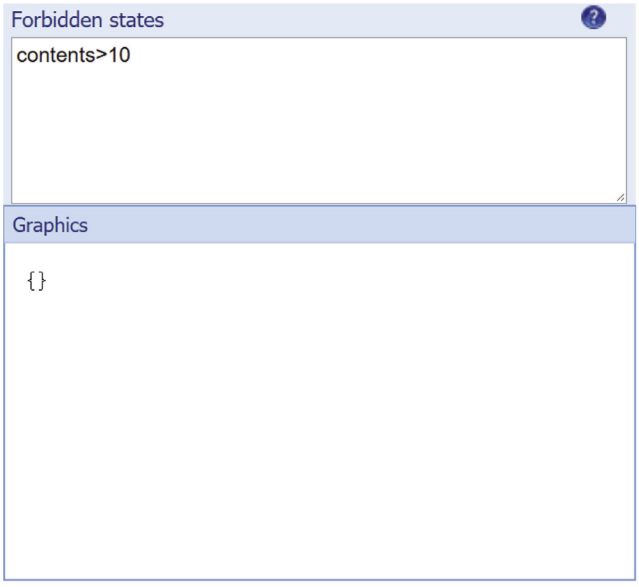
We set an error state as "*content of a barrel is over 10 liters*" to check the safety requirement, and conduct a reachability analysis for the error state. If the reachability analysis result shows that the error state is unreachable in all cases, we can conclude that the model satisfies the safety requirement. *Forbidden states* set as '*contents* > 10' to define the error state. Fig. 33(a) shows the result of this reachability analysis in *SpaceEx Web* interface. The verification result shows that "*Forbidden states are not reachable*" as a console message, and the textual output is an empty set in Fig. 33(b). It means that the error state is not reachable, thus we conclude that the safety property is satisfied.

We define two *forbidden states* to verify *Reachability Requirement 1*. First, "*A fully filled barrel is moving*" and second, "*The barrel is moved to the end of the conveyor belt*". The *Barrel-Mover System* conveys fully filled barrels at *working_prog*0 or *working_prog*1 in Fig. 29. The fully filled barrel event '*barrelinput* == 1' in Fig. 29 could not be used to specify the first *forbidden states*, since *barrel_dist* is reset whenever the event occurs. The second state can be simply specified as *barrel_dist* == 10. The verification result is "*Forbidden states are reachable*" and the textual output describes that the *forbidden states* are satisfied at '*t* == 26' as shown in Fig. 33(c). We can conclude that the model satisfies *Reachability Requirement 1*.

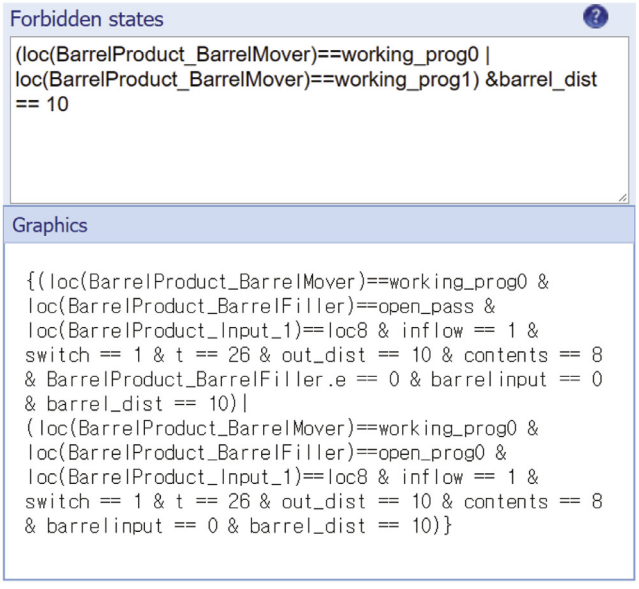
We analyzed the sequential behaviors of the model in order to check whether the model satisfies *Reachability Requirement 2*. It is composed of two conditions: "*The first fully filled barrel arrives*", and then "*The conveyor belt starts to move*". A fully filled barrel event can occur a few times, thus we need to find out when the event occurs for the first time, and analyze the response behavior of the model for the event. *SpaceEx* generates a graphical format verification result, and it needs a time variable in order to



(a) A screen dump of *SpaceEx* web interface



(b) Verification result of *Safety Requirement1*



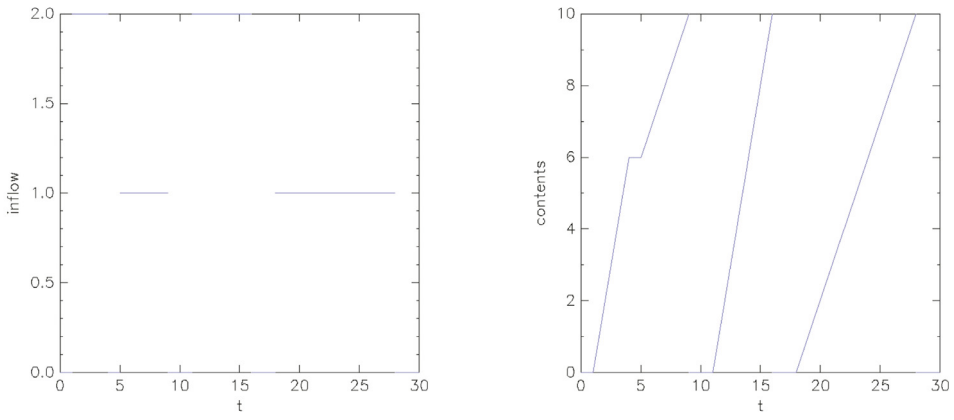
(c) Verification result of *Reachability Requirement1*

Fig. 33. Verification results for Safety Requirement1 & Reachability Requirement1.

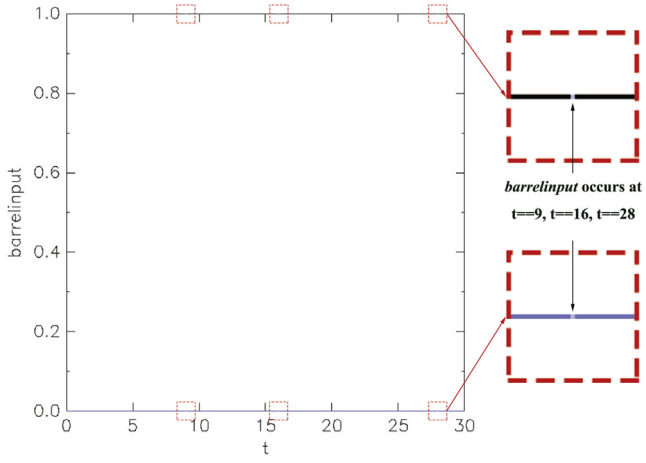
generate the sequential change of the variable. Therefore, '*ECML-toSpaceEx*' generates an additional base component (*timer*) to support the analysis of the sequential change of variables. The base component *timer* has only one variable *t* and one location. The value of *t* increases according to the *flow* '$t'==1$', without being interrupted by any other components. We used the *t* variable as a global clock variable, which is useful when analyzing the verification results of *SpaceEx*. As indicated in Table 3, we also used empty *forbidden states*, since *SpaceEx* shows the states of a model when the model satisfies the *forbidden states*. *SpaceEx* shows all the states of a model when the *forbidden states* are empty.

Figs. 34(b)–(d) show 2D graph outputs of *SpaceEx* for *Reachability Requirement 2*. Fig. 34(b) is a graph describing the sequential change of *contents*, and it shows that *contents* needs 9 time units to attain a value equal to 10. At 9 time units, fully filled barrel event *barrelinput* occurs for the first time as shown in Fig. 34(c). *barrel_dist* is equal to 0 until this time, and then increases when the *Barrel-Mover System* obtains the event as shown in Fig. 34(d). Hence, we conclude that *Reachability Requirement 2* is satisfied.

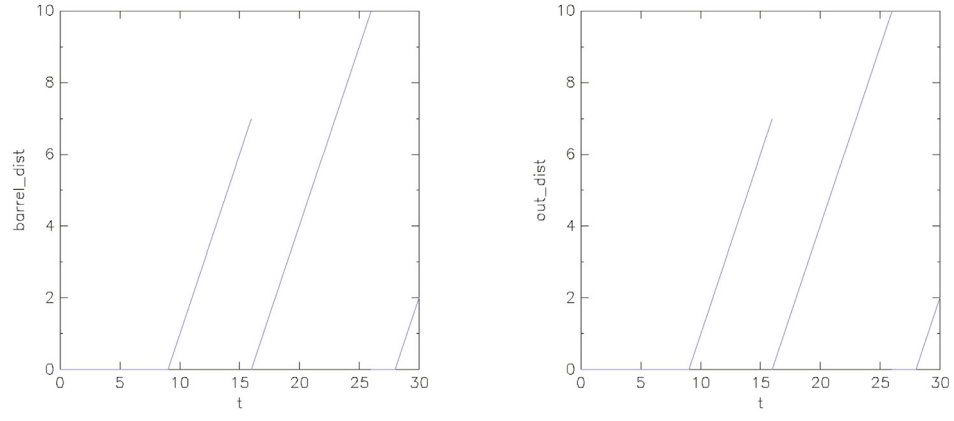
It is possible to determine whether the behavior of the translated model is the same as that in Fig. 22. Fig. 34(a) shows the sequential change of *inflow*, and *contents* increases according to the behavior of *inflow*. *barrelinput* occurs at 9, 16, and 28 time units,



(a) Sequential changes of *inflow* (b) Sequential changes of *contents*



(c) Sequential changes of *barrelinput*



(d) Sequential changes of *barrel_dist* (e) Sequential changes of *out_dist*

Fig. 34. Sequential changes of variables.

thus *barrel_dist* is reset and increases every time the event occurs. *barrel_dist* is also reset at 26 time units, because the last fully filled barrel is moved to the end of the conveyor belt. The output variable *out_dist* outputs the current position of the fully filled barrel on the conveyor belt (*i.e., barrel_dist*), and its behavior is shown in Fig. 34(e).

5. Related work

We surveyed modeling and verification techniques for hybrid systems to find efficient ways for verifying the ECML model, as listed in Table 5. Important criteria pertinent to our discussion are: (1) should be non-commercial, and (2) should perform formal verification automatically, *i.e.,* model checking, not a theorem-proving technique.

Table 5
Tools for modeling, analyzing and verifying hybrid systems.

Name	Objective	Input front-end	Verification method	Commercial
CHARON[6]	Modeling, simulation	CHARON language	None	No
CheckMate[59][a]	Verification	Autonomous linear Hybrid automata	Rectangular polytopes Automation	Yes
d/dt[60]	Verification	Linear hybrid automata	Over-approximation	No
Ellipsoidal ToolBox[61][a]	Verification	Controlled linear hybrid system	Pararellotope method[62]	Yes
GBT[63][a]	Computation	Polytope, ellipsoid	Convex hull determination	Yes
HSIF[64]	Modeling, simulation	Network (collection of hybrid automata)	None	No
HSolver[65]	Verification	Input hybrid system	Constraint propagation[b]	No
HyTech[10]	Verification	Linear hybrid automata	Quantifier elimination, validity checking	No
HyVisual[66]	Modeling	Embedded systems	None	No
KeYmaera[67]	Verification	Differential dynamic logic	Symbolic decomposition[b]	No
Level Set ToolBox[68][a]	Verification	Partial differential equation	Hamilton-Jacobi equation solutions[69]	Yes
MATISSE[70][a]	Verification	Transition system	Bisimulation	Yes
MultiParametric ToolBox[71][a]	Simulation, verification	Piecewise affine systems	Linear/quadratic programming solver	Yes
PHAVer[11]	Verification	Linear I/O hybrid automata	On-the-fly over-approximation	No
Ptolemy II[72]	Modeling, Simulating	Embedded system (contains hybrid system)	Non-hybrid system Verifier	No
SHIFT[73]	Modeling, translation	SHIFT language	None	No
SpaceEx[12]	Verification	Hybrid automata	Time-step flowpipe computation	No
STeP[74]	Verification	Real-time system	Invariant generation[b]	No

[a] Requiring Matlab, commercial.
[b] Theorem proving.

We selected five tools appropriate for our criteria: *d/dt, PHAVer, SpaceEx*, and *Ptolemy II. HyTech* [10] focuses on simple continuous dynamics in each discrete state. We translated and tried to verify a simple vehicle model [28,75] with an input scenario automaton. The translated model is structured with 4 sub-systems, 19 variables and 38 locations, however we could not conduct reachability analysis because of scalability of *HyTech* [11,30]. *HyTech* used up to 4 GB memories and aborted the translated model verification. *d/dt* [60] can verify hybrid systems with linear continuous dynamics under uncertain bounded input using an over-approximation method. *PHAVer* [11] supplements the weakness of *HyTech* [76] and supports an I/O structure. It can compute non-convex polyhedra [77], and has an enhanced fixed-point computation. *SpaceEx* [12] can verify linear hybrid systems using various forms of a polyhedron. It also uses a time-step extension of a scalable time-elapse algorithm for verification. *Ptolemy II* [72] can verify a hybrid system using one of three model checkers: NuSMV [78], REDLIB [79], and Real Time Maude [80]. NuSMV is a model checker for finite state systems, and REDLIB is a model checker for timed automata. Real Time Maude can analyze a hybrid system, but it is not a verification tool for hybrid systems. In addition, *Ptolemy II* includes HyBisual which can model a hybrid system visually.

We have challenging work in order to conduct model checking for ECML models. *SpaceEx* supports large scale (*i.e.,* number) of variables, but it could not be sufficient for large scale models. We tried to apply our translation rules to conduct reachability analysis on an industrial vehicle example (see Fig. 3). We conducted reachability analysis for the model, however, the analysis needed many modifications of the model because the model system has many nonlinear dynamics and user-defined functions. In order to conduct liveness and fairness analyses other than reachability, we need temporal logic based verification environment [81], and we are considering other verification tools to analyze various dynamics [82].

6. Conclusion and future work

This paper describes translation rules for verifying ECML hybrid simulation models that are mechanized by supporting tools. Based on the rules, an ECML model is translated into linear hybrid automata and then a formal verification, such as safety verification and reachability analysis, is supported by *SpaceEx*. An ECML model is described as a structural model, which comprises structured behavioral models, and these models are translated into a network component and base components of *SpaceEx*, respectively. SpaceEx then performed formal verification on the translated models successfully. We implemented the proposed translation rules into an automatic translator, '*ECMLtoSpaceEx*,' and required the ECML model to satisfy the assumptions and restrictions checked by '*ECML Checker*'. We tried to verify an industrial vehicle ECML model with our verification technique; however, the approach is not directly suitable because the model has too many user-defined functions and nonlinear dynamics.

We expect it to be possible to use our translation rules to support verification of ECML models with tools other than *SpaceEx* with minor modifications, since they translate ECML models into linear hybrid automata based models, which is a notation commonly accepted by other verification tools. Currently, we are developing formal translation rules, and are also trying to translate ECML models into nonlinear hybrid automata, besides proposing verification strategies for these automata.

Acknowledgments

This paper was supported by Konkuk University in 2015.

References

[1] P.J. Antsaklis, J.A. Stiver, M.D. Lemmon, Interface and controller design for hybrid control systems, in: Hybrid Systems II, LNCS 999, Springer, 1995, pp. 462–492.
[2] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (2) (1994) 183–235.
[3] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, Theor. Comput. Sci. 138 (1) (1995) 3–34.
[4] R. Alur, T.A. Henzinger, P.-H. Ho, Automatic symbolic verification of embedded systems, IEEE Trans. Softw. Eng. 22 (3) (1996) 181–201.
[5] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation, Academic Press, 2000.
[6] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivančić, I.L. Vijay Kumar, P. Mishra, G.J. Pappas, O. Sokolsky, Hierarchical modeling and analysis of embedded systems, Proc. IEEE 91 (1) (2003) 11–28.
[7] S. Yoon, An ETRI CPS modeling language for hybrid system simulation, Technical Report TR-DS-2015-01, 2015. " http://dslab.konkuk.ac.kr/Publication/Publication.htm".
[8] K.G. Larsen, P. Pettersson, W. Yi, Uppaal in a nutshell, Int. J. Softw. Tools Technol. Transf. (STTT) 1 (1) (1997) 134–152.
[9] C. Daws, A. Olivero, S. Trypakis, S. Yovine, The tool KRONOS, in: Hybrid Systems III, LNCS 1066, Springer, 1996, pp. 208–219.
[10] T.A. Henzinger, P.-H. Ho, H. Wong-Toi, Hytech: a model checker for hybrid systems, Softw. Tools Technol. Transf. 1 (1–2) (1997) 110–122.
[11] G. Frehse, PHAVEr: algorithmic verification of hybrid systems past hytech, Hybrid Syst.: Comput. Control (2005) 258–273.

[12] 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, in: Computer Aided Verification, Springer, 2011, pp. 379–395.

[13] E.M. Clarke, E.A. Emerson, A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Program. Lang. Syst. 8 (2) (1986) 244–263.

[14] W. Visser, M.B. Dwyer, M. Whalen, The hidden models of model checking, Softw. Syst. Model. 11 (4) (2012) 541–555.

[15] S. Nair, J.L. de la Vara, M. Sabetzadeh, D. Falessi, Evidence management for compliance of critical systems with safety standards: a survey on the state of practice, Inf. Softw. Technol. 60 (2015) 1–15.

[16] D. Aceituna, G. Walia, H. Do, S.-W. Lee, Model-based requirements verification method: conclusions from two controlled experiments, Inf. Softw. Technol. 56 (3) (2014) 321–334.

[17] J.L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R.K. Panesar-Walawege, Á. López, I. del Río, T. Kelly, Model-based specification of safety compliance needs for critical systems: a holistic generic metamodel, Inf. Softw. Technol. 72 (2016) 16–30.

[18] R.K. Panesar-Walawege, M. Sabetzadeh, L. Briand, Supporting the verification of compliance to safety standards via model-driven engineering: approach, tool-support and empirical validation, Inf. Softw. Technol. 55 (5) (2013) 836–864.

[19] A. De Roo, H. Sözer, M. Akşit, Verification and analysis of domain-specific models of physical characteristics in embedded control software, Inf. Softw. Technol. 54 (12) (2012) 1432–1453.

[20] I. Chun, J. Kim, H. Lee, W. Kim, S. Park, E. Lee, Faults and adaptation policy modeling method for self-adaptive robots, in: Proceedings of International Conference on Ubiquitous Computing and Multimedia Applications, Springer, 2011, pp. 156–164.

[21] S. Kang, M. Kim, J. Park, I. Chun, W. Kim, LVC-interoperation development framework for acquiring high reliable cyber-physical weapon systems, J. Korean Inst. Commun. Inf. Sci. 38 (12) (2013) 1228–1236.

[22] W.-T. Kim, I.-G. Chun, S.-H. Lee, J.-M. Park, A large-scale autonomous CPS software platform (in Korean), Commun. Korean Inst. Inf. Sci. Eng. 12 (31) (2013) 16–28.

[23] M.-J. Kim, S. Kang, W.-T. Kim, I.-G. Chun, Human-interactive hardware-in-the-loop simulation framework for cyber-physical systems, in: Proceedings of Second International Conference on Informatics and Applications (ICIA), IEEE, 2013, pp. 198–202.

[24] H.Y. Lee, I. Chun, W.-T. Kim, DEV&DESS-based cyber-physical systems modeling language with uncertainty consideration, in: Proceedings of the 2013 Spring Simulation Multiconference Poster Session, Society for Computer Simulation International, 2013, p. 1.

[25] H. Choi, S. Cha, J.Y. Jo, J. Yoo, H.Y. Lee, W.-T. Kim, Formal verification of DEV&DESS formalism using symbolic model checker hytech, in: Control and Automation, and Energy System Engineering, Springer, 2011, pp. 112–121.

[26] H. Choi, S. Cha, J.Y. Jo, J. Yoo, H.Y. Lee, W.-T. Kim, Formal verification of basic DEV&DESS formalism using hytech, Inf.-Inter. Interdiscip. J. 16 (1 B) (2013) 821–826.

[27] J. Jo, J. Yoo, H. Choi, S. Cha, H.Y. Lee, W.-T. Kim, Translation from ECML to linear hybrid automata, in: Embedded and Multimedia Computing Technology and Service, Springer, 2012, pp. 293–300.

[28] S. Yoon, J. Jo, I.-g. Chun, J. Yoo, Verification and analysis of ECML models using HyTech (in Korean), in: Korea Computer Software Engineering 2014(KCSE2014), 2014, pp. 2–10.

[29] J. Jo, A systematic verification of ECML model using hytech, (Master's thesis), Department of Computer & Information Communication Engineering, Konkuk University, Korea, 2013.

[30] L.P. Carloni, R. Passerone, A. Pinto, A.L. Sangiovanni-Vincentelli, et al., Languages and tools for hybrid systems design, Found. Trends® Electron. Des. Autom. 1 (1–2) (2006) 1–193.

[31] J. Jo, S. Yoon, J. Yoo, H. Lee, W. Kim, Case study: verification of ECML model using SpaceEx, in: Proceedings of Korea-Japan Joint Workshop on ICT, Pohang, Korea, 2012, pp. 1–4.

[32] G. Frehse, R. Kateja, C. Le Guernic, Flowpipe approximation and clustering in space-time, in: Proceedings of Hybrid Systems: Computation and Control (HSCC'13), ACM, 2013, pp. 203–212.

[33] G. Frehse, A brief experimental comparison of the STC and LGG analysis algorithms in SpaceEx, 2012. " http://spaceex.imag.fr/documentation/user-documentation".

[34] M.F. Karoui, H. Alla, A. Chatti, Monitoring of dynamic processes by rectangular hybrid automata, Nonlinear Anal. Hybrid Syst. 4 (4) (2010) 766–774.

[35] A. Allahham, H. Alla, Post and pre-initialized stopwatch petri nets: formal semantics and state space computation, Nonlinear Anal. Hybrid Syst. 2 (4) (2008) 1175–1186.

[36] M. Sipser, Introduction to the Theory of Computation, Cengage Learning, 2012.

[37] K. Bae, J. Krisiloff, J. Meseguer, P.C. Ölveczky, Designing and verifying distributed cyber-physical systems using multirate PALS: an airplane turning control system case study, Sci. Comput. Program. 103 (2015) 13–50.

[38] J.H. Jeon, I. Chun, P.S.M. Kim Won-Tae, Design and method in modeling of cyber-physical systems, in: Proceedings of JCICT & The first Yellow Sea International Conference on Ubiquitous Computing (YES-ICUC), 2011.

[39] J. Jeon, I. Chun, K. Won-Tae, Metamodel-based CPS modeling tool, Embed. Multimed. Comput. Technol. Serv. (LNCS) 181 (2012) 285–291.

[40] E.I. Kim, M.J. Park, I.-g. Chun, W.-T. Kim, Reliability support framework for cyber physical systems, in: Proceedings of 2011 International Symposium on Embedded Technology (ISET 2011), 2011, pp. 1–5.

[41] J.Y. Kim, D.N. Choi, H.J. Kim, J.M. Kim, W.-T. Kim, Abstracted CPS model: a model for interworking between physical system and simulator for CPS simulation, in: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 2012), 2012, pp. 26–29.

[42] T.A. Henzinger, The theory of hybrid automata, in: Proceedings of the eleventh Annual Symposium on Logic in Computer Science (LICS'96), IEEE Comp. Soc. Press, 1996, pp. 278–292.

[43] R. Alur, A. Thomas, Real-time system= discrete system+ clock variables, Int. J. Softw. Tools Technol. Trans. 1 (1–2) (1997) 86–109.

[44] D. Harel, On visual formalism, Commun. ACM 31 (5) (1986) 514–530.

[45] ANTLR v3, (http://www.antlr3.org/).

[46] P. Schrammel, B. Jeannet, From hybrid data-flow languages to hybrid automata: a complete translation, Technical Report, 2012. Research Report n7859.

[47] C. van Beek, N.G. Jansen, K.E. Eooda, R.R. Schiffelers, K.L. Man, M.A. Reniers, Relating Chi to hybrid automata, in: Proceedings of the 2003 Winter Simulation Conference, 2003, pp. 632–640.

[48] A. Agrawal, G. Simon, G. Karsai, Semantic translation of simulink/stateflow models to hybrid automata using graph transformations, Electron. Notes Theor. Comput. Sci. 109 (2004) 43–56.

[49] S. Han, K. Huang, Equivalent semantic translation from parallel DEVS models to time automata, in: ICCS 2007, in: Part I, LNCS 4487, 2007, pp. 1246–1253.

[50] S. Borland, Transforming statechart models to DEVS, (Ph.D. thesis), McGill University, 2003.

[51] T. Baar, Correctly defined concrete syntax, Softw. Syst. Model. 7 (4) (2008) 383–398.

[52] M.A. Dave, Compiler verification: a bibliography, ACM SIGSOFT Softw. Eng. Notes 28 (6) (2003) 2.

[53] T. Hoare, The verifying compiler: a grand challenge for computing research, J. ACM 50 (1) (2003) 63–69.

[54] L.A. Rahim, J. Whittle, A survey of approaches for verifying model transformations, Softw. Syst. Model. 14 (2) (2013) 1003–1028.

[55] M. Konečnỳ, W. Taha, F.A. Bartha, J. Duracz, A. Duracz, A.D. Ames, Enclosing the behavior of a hybrid automaton up to and beyond a zeno point, Nonlinear Anal. Hybrid Syst 20 (2016) 1–20.

[56] D. Lepri, E. Ábrahám, P.C. Ölveczky, Sound and complete timed CTL model checking of timed Kripke structures and real-time rewrite theories, Sci. Comput. Program 99 (2015) 128–192.

[57] K. Bae, J. Meseguer, Model checking linear temporal logic of rewriting formulas under localized fairness, Sci. Comput. Program 99 (2015) 193–234.

[58] H.A. Hansen, G. Schneider, M. Steffen, Reachability analysis of complex planar hybrid systems, Sci. Comput. Program 78 (12) (2013) 2511–2536.

[59] A. Chutinan, B.H. Krogh, Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations, in: Proceedings of Hybrid Systems: Computation and Control, LNCS 1569, 1999, pp. 76–90.

[60] E. Asarin, T. Dang, O. Maler, The d/dt tool for verification of hybrid systems, in: Proceedings of Computer Aided Verification, LNCS 2404, 2002, pp. 746–770.

[61] A. Kurzhanski, P. Varaiya, Ellipsoidal Toolbox manual. EECS Department, 2008.

[62] E. Kostousovat, Control synthesis via parallelotopes: optimzation and parallel computations, Optim. Methods Softw. 14 (4) (2001) 267–310.

[63] C. Graves, S. Veres, Using MATLAB toolbox LDQUO;GBT RDQUO; in identification and control, in: Proceedings of IEE Colloquium on Identification of Uncertain Systems, 1994, pp. 11/1–11/6.

[64] A. Pinto, L. Carloni, R. Passerone, A. Sangiovanni-Vincentelli, Interchange format for hybrid systems: abstract semantics, Hybrid Syst.: Comput. Control 3927 (2006) 491–506.

[65] S. Ratschan, Z. She, HSolver: verification of hybrid systems based on the constraint solver RSolver, (Online) "http://hsolver.sourceforge.net/".

[66] C. Brooks, A. Cataldo, E.A. Lee, J. Liu, X. Liu, S. Neuendorffer, H. Zheng, Hyvisual: a hybrid system visual modeler, Technical Memorandum UCB/ERL M05/24, University of California, Berkeley, 2005.

[67] A. Platzer, J. Quesel, Keymaera: a hybrid theorem prover for hybrid systems (system description), Autom. Reason. 5195 (2008) 171–178.

[68] I. Mitchell, A toolbox of level set methods version 1.0, Department of Computer Science, University of British Columbia, UBC CS TR-2004-09, 2004.

[69] S. Osher, A level set formulation for the solution of the Dirichlet problem for Hamilton–Jacobi equations, SIAM J. Math. Anal. 24 (1993) 1145.

[70] A. Girard, G.J. Pappas, Approximation metrics for discrete and continuous systems, IEEE Trans Autom. Control 52 (5) (2005) 782–798.

[71] M. Kvasnica, P. Grieder, M. Baotić, M. Morari, Multi-parametric toolbox (MPT), Hybrid Syst.: Comput. Control 2993 (2004) 121–124.

[72] C. Brooks, E. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng, Heterogeneous concurrent modeling and design in java (volume 1: introduction to ptolemy ii), Tech. Rep. UCB/EECS-2008-28, Apr, 2008.

[73] M. Antoniotti, A. Göllü, SHIFT and SMART-AHS: a language for hybrid system engineering modeling and simulation, in: Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997, USENIX Association, 1997. 14–14

[74] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H. Sipma, T. Uribe, STeP: deductive-algorithmic verification of reactive and real-time systems, in: Computer Aided Verification, Springer, 1996, pp. 415–418.

[75] S. Yoon, I.-g. Chun, W.-T. Kim, J. Jo, J. Yoo, An ETRI CPS modeling language for specifying hybrid systems (in korean), J. KIISE 42 (7) (2015) 823–833.
[76] T. Henzinger, J. Preussig, H. Wong-Toi, Some lessons from the HyTech experience, in: Proceedings of the Fortieth IEEE Conference on Decision and Control, 2001., 3, IEEE, 2001, pp. 2887–2892.
[77] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, J. Jouannaud, Software Engineering with OBJ: Algebraic Specification in Action, Kluwer Academic Publishers, pp. 3–167.
[78] A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, NuSMV: a new symbolic model verifier, Computer Aided Verification, Springer, 1999. 682–682
[79] F. Wang, REDLIB for the formal verification of embedded systems, in: Proceedings of Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, 2006, pp. 341–346.
[80] P. CsabaOlveczky, J. Meseguer, Real-Time Maude: a tool for simulating and analyzing real-time and hybrid systems, in: Proceedings of Third International Workshop on Rewriting Logic and its Applications, WRLA, 2000, pp. 18–20.
[81] Y. Annpureddy, C. Liu, G. Fainekos, S. Sankaranarayanan, S-taliro: a tool for temporal logic falsification for hybrid systems, in: Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2011, pp. 254–257.
[82] M. Althoff, An introduction to cora 2015., in: Proceedings of ARCH@ CPSWeek, 2015, pp. 120–151.