Dynamic Linear Hybrid Automata and Their Applications to Formal Verification of Dynamic Reconfigurable Embedded Systems

博　　士　　論　　文

Dynamic Linear Hybrid Automata and Their Application to Formal Verification of Dynamic Reconfigurable Embedded Systems

金沢大学大学院自然科学研究科
電子情報科学専攻

学籍番号: 1323112012
氏名: 柳瀬　龍
主任指導教員: 山根　智
提出年月: 平成 29 年 2 月

DYNAMIC LINEAR HYBRID AUTOMATA AND THEIR APPLICATIONS TO FORMAL VERIFICATION OF DYNAMIC RECONFIGURABLE EMBEDDED SYSTEMS

Graduate School of Natural Science and Technology,
Kanazawa University

by
Ryo Yanase

February 2016

Dynamic Linear Hybrid Automata and Their Applications to Formal Verification of Dynamic Reconfigurable Embedded Systems

Ryo Yanase
Kanazawa University

ABSTRACT

A dynamically reconfigurable system can change its configuration during operation, and studies of such systems are being carried out in many fields. In particular, medical technology and aerospace engineering must ensure system safety because any defect will have serious consequences. Model checking is a method for verifying system safety. In this paper, we propose the Dynamic Linear Hybrid Automaton (DLHA) specification language and show a method to analyze reachability for a system consisting of several DLHAs.

Contents

1 Introduction

1.1 Background

Dynamically reconfigurable systems can change their configuration during operation. Such systems are being used in a number of areas [1, 2, 3] of an apparatus that involves human lives or expensive manufactured goods (e.g., in medical or aerospace engineering). Here, it is very important to guarantee safety. The major methods of checking system safety include simulation and testing; however, it is difficult for them to ensure safety precisely, since large systems can have infinite state spaces. In such a case, model checking that performs exhaustive searches is a more effective method.

In this paper, we propose the *Dynamic Linear Hybrid Automaton* (DLHA) specification language for describing dynamically reconfigurable systems and provide a reachability analysis algorithm for verifying system safety.

1.2 Features of dynamically reconfigurable systems consisting of CPUs and DRPs

The target of our research is an embedded system in which a CPU and dynamically reconfigurable hardware, e.g., DRP or D-FPGA [4] operate cooperatively. The dynamically reconfigurable processor (DRP) is a coarse-grained programmable processor developed by NEC [3], and it manages both the power conservation and miniaturization. The DRP is used to accelerate the computations of a general purpose CPU through cooperative operations, and it has the following features:

- Dynamic creation/destruction of functions: when a process occurs, the DRP constitutes a private circuit for processing it. The circuit configuration is released after the process finishes.
- Hybrid property: the operation frequency changes whenever a context switch occurs.
- Parallel execution: the DRP executes several processes on the same board at the same time.
- Queue for communication: the DRP asynchronously receives processing requests from the CPU.

1.3 Our Proposal

1.3.1 Specification

We devised the following new specification techniques for dynamically reconfigurable systems consisting of CPUs and DRPs:

- We use linear hybrid automata [5] describing changes in the operating frequency.
- We use linear hybrid automata that have creation/destruction events describing dynamic creations and destructions of configuration components.

- We use FIFO queues describing asynchronous communication.

We developed a new specification language (DLHA) based on a linear hybrid automaton with both creation/destruction events and unbounded FIFO queues. DLHA is different from existing research in the following points:

- V. Varshavsky and J. Esparza proposed the GALA (Globally Asynchronous - Locally Arbitrary) modeling approach including timed guards [6]. This approach cannot describe hybrid systems since it is the specification language based on discrete systems. Thus, GALA cannot represent changes in operating frequency.

- S. Minami and others have specified a dynamically reconfigurable system using linear hybrid automata and have verified it by using a model checker, HyTech [7]. Since linear hybrid automata cannot describe changes to the configuration and asynchronous communications, the system has been specified as a static system. Therefore, the specification presented in their work is unsuitable for representing dynamically reconfigurable systems. Moreover, they verified only the *schedulability property* of the system, whereas we have verified several other properties in our work.

1.3.2 Verification Method

The originality of our work on the verification method is twofold:

- Our method targets systems that dynamically change their configurations, which is something the existing work, such as HyTech, has studied. We extend the syntax and semantics of linear hybrid automata with special actions called *creation actions* and *destruction actions*. We define a state in which an automaton does not exist and transitions for creation and destruction.

- Our method is a comprehensive symbolic verification for hybrid properties, FIFO queues and creation/destruction of tasks.

1.3.3 Experiments on Verifying Dynamically Reconfigurable Systems

For the experiments, we specified a dynamically reconfigurable embedded system consisting of a CPU and DRP, and verified some of its important features. This is the first time that specification and verification of dynamic changes have been tried in a practical case.

1.4 Related Work

Here, we describe related work and how it differs from our work.

- P. C. Attie and N. A. Lynch specified systems whose components are dynamically created/destroyed by using I/O automata [8]. I/O automata cannot describe changes in variables, for example, changes in the clock and operating frequency.

- H. Yamada and others proposed hierarchical linear hybrid automata for specifying dynamically reconfigurable systems [9]. They introduced concepts such as class, object, etc., to the specification language. However, as the scale of the system to be specified increases, the representation and method of analysis in the verification stage tend to be complex.
- B. Boigelot and P. Godefroid specified a communication protocol in terms of finite-state machines and unbounded FIFO buffers (queues), and they verified it [10]. Since the finite-state machine also cannot describe changes in variables, it is unsuitable in our case.
- A. Bouajjani and others proposed a reachability analysis for pushdown automata and a symbolic reachability analysis for FIFO-channel systems [11, 12]. However, since their analysis don't provide for continuous changes in variables, in languages cannot be used for designing hybrid systems.

2 Dynamic Linear Hybrid Automaton

2.1 Preliminaries

Definition 1 (Constraint)**.** *Let V be a finite set of variables. A constraint ϕ on V is defined as*

$$\phi ::= true \mid x \sim e \mid x - y \sim e \mid \phi_1 \wedge \phi_2,$$

where $x, y \in V$, $e \in \mathbb{Q}$, ϕ_1 and ϕ_2 are constraints on V, and $\sim \in \{=, <, >, \leq, \geq\}$. $\Phi(V)$ denotes the set of all constraints on V.

Definition 2 (Flow condition)**.** *Let $V = \{x_1, \ldots, x_n\}$ be a finite set of (real-valued) variables. A flow condition f on V is defined as*

$$f ::= \dot{x}_1 = d_1 \wedge \ldots \wedge \dot{x}_n = d_1,$$

where $d_1, \ldots, d_n \in \mathbb{Q}$. $F(V)$ is the set of all flow conditions.

For each variable x, we use the dotted variable $\dot{x}$, to denote the first derivative of x.

Definition 3 (Update Expression)**.** *Let V be a finite set of variables. An update expression upd on V is defined as*

$$upd ::= x := c \mid x := x + c,$$

where $x \in V$ and $c \in \mathbb{Q}$. $UPD(V)$ is the set of all update expressions can be written.

2.2 Syntax

A dynamic linear hybrid automaton (DLHA) is a tuple $(L, V, Inv, Flow, Act, T, t_0, T_d)$, where

- L is a finite set of locations.

- V is a finite set of (real-valued) variables.
- $Inv : L \to \Phi(V)$ is a function that assigns a constraint to each location.
- $Flow : L \to F(V)$ is a function that assigns a flow condition to each location.
- $Act = Act_{in} \cup Act_{out} \cup Act_{\tau}$ is a finite set of *actions*.
  - Act_{in} is a finite set of *input actions*, and each input action has the form $a?$. An input action $m?$ denotes receiving the message m.
  - Act_{out} is a finite set of *output actions*, and each output action has the form $a!$. An output action $m!$ denotes sending the message m to each DLHA.
  - Act_{τ} is a finite set of *internal actions* that denote changing a state of a DLHA.

  Moreover, we formalize the following special actions:
  - A *creation action* that has the form $Crt\_\mathcal{A}'?$ or $Crt\_\mathcal{A}'!$ denotes a message for creation of DLHA $\mathcal{A}'$. $Crt\_\mathcal{A}'? \in Act_{in}$ is an input action, and it represents that $\mathcal{A}'$ has been created. $Crt\_\mathcal{A}'! \in Act_{out}$ is an output action, and represents a request for creating $\mathcal{A}'$.
  - A *destruction action* that has the form $Dst\_\mathcal{A}'?$ or $Crt\_\mathcal{A}'!$ denotes a message for a destruction of DLHA $\mathcal{A}'$. $Dst\_\mathcal{A}'? \in Act_{in}$ is an input action that indicates $\mathcal{A}'$ has been destroyed.
  - An *enqueue action* that has the form $q!m$ denotes enqueueing of message m into a queue q. This action is an internal one, that is, $q!m \in Act_{\tau}$.
  - A *dequeue action* that has the form $q?m$ denotes dequeueing of message m from the top of q.
- $T \subseteq L \times \Phi(V) \times Act \times 2^{UPD(V)} \times L$ is a finite set of *transitions*. A constraint $\phi \in \Phi(V)$ is called a *guard condition*.
- $t_0 \in L \times (Act_{in} \cup Act_{\tau}) \times 2^{UPD(V)}$ is an *initial transition*.
- $T_d \subseteq L \times \Phi(V) \times Act_{out}$ is a finite set of *destruction-transitions*.

2.3 Operational Semantics

A state σ of a DLHA $(L, V, Inv, Flow, T, t_0, T_d)$ is defined as

$$\sigma ::= \bot \mid (l, \nu),$$

where $l \in L$ is a location, $\nu : V \to \mathbb{R}$ is an assignment called *evaluation* of variables, and $\bot$ denotes an *undefined value*.

We define the semantics $\mathcal{M}$ of the DLHA by $(\Sigma, \Rightarrow, \sigma_0)$ where

- Σ is a set of states.

- $\Rightarrow$ is a set of *time transitions* and *discrete transitions*.
- σ_0 is the initial state.

The following rules define time and discrete transitions:

Definition 4 (Time transition of a DLHA)**.** *For any $\delta \in \mathbb{R}_{\geq 0}$,*

- $\bot \Rightarrow_\delta \bot$
- $(l, \nu) \Rightarrow_\delta (l, \nu')$ *if* $\nu' = \nu + \delta \cdot Flow(l) \in Inv(l)$
  where $\nu' = \nu + \delta \cdot Flow(l)$ denotes an evaluation such that $\forall x \in V. \nu'(x) = \nu(x) + \delta \cdot \dot{x} \wedge Flow(l)$, and $\nu' \in Inv(l)$ denotes that $\nu'(x)$ satisfies the constraint $Inv(l)$ for any $x \in V$.

Definition 5 (Discrete transition of a DLHA)**.** *For an evaluation ν and update expressions $\lambda \in 2^{UPD(V)}$, $\nu[\lambda]$ denotes an evaluation updated by λ, that is, for any $x \in V$,*

$$\nu[\lambda](x) = \begin{cases} c & (x := c \in \lambda) \\ \nu(x) + c & (x := c \notin \lambda, x := x + c \in \lambda) \\ \nu(x) & (otherwise) \end{cases}$$

- *For any transition $(l, \phi, a, \lambda, l') \in T$, $(l, \nu) \Rightarrow_a (l, \nu[\lambda])$ if $\nu \in \phi$ and $\nu[\lambda] \in Inv(l')$.*
- *(Creation of a DLHA) For the initial transition $t_0 = (l_0, a_0, \lambda_0)$, $\bot \Rightarrow_{a_0} (l_0, \vec{0}[\lambda_0])$, where $\vec{0}$ is an evaluation such that $\forall x \in V. \vec{0}(x) = 0$.*
- *(Destruction of a DLHA) For any destruction-transition $(l, \phi, a) \in T_d$, $(l, \nu) \Rightarrow_a \bot$ if $\nu \in \phi$*

For the initial transition (l_0, a_0, λ_0), the initial state σ_0 is defined as

$$\sigma_0 = \begin{cases} \bot & (a_0 \in Act_{in}) \\ (l_0, \vec{0}[\lambda_0]) & (otherwise). \end{cases}$$

3 Dynamically Reconfigurable Systems

To describe an asynchronous communication among DLHAs in a dynamically reconfigurable system (DRS), we use a queue (*unbounded* FIFO buffer) as a model of the communication channel. We assume that the system performs lossless transmission, so we can let the queue be unbounded.

3.1 Syntax of DRS

A dynamically reconfigurable system (DRS) $\mathcal{S}$ is defined by a tuple $(A, \mathcal{Q})$ consisting of a finite set $A = \{\mathcal{A}_1, \ldots, \mathcal{A}_{|A|}\}$ of DLHAs and a finite set $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$ of queues.

3.2 Semantics of DRS

A state s of a DRS $\mathcal{S} = (A, \mathcal{Q})$ is a tuple $\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle$, where

- $\vec{\sigma} \in \Sigma_1 \times \cdots \times \Sigma_{|A|}$ is a vector of the states of DLHAs.
- $\vec{w}_{\mathcal{Q}} \in M_1^* \times \cdots \times M_{|\mathcal{Q}|}^*$ is a vector of the content of the queues, where each M_i is the set of all messages that can be stored in queue q_i.

Definition 6 (Time Transition of a DRS)**.** *For an arbitrary $\delta \in \mathbb{R}_{\geq 0}$, the time transition is defined as*

$$\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle \rightarrow_{\delta} \langle \vec{\sigma}', \vec{w}_{\mathcal{Q}} \rangle \iff \forall i. \sigma_i \Rightarrow_{\delta} \sigma_i.$$

Definition 7 (Discrete Transition of a DRS)**.** *Let $\vec{\sigma}, \vec{\sigma}', \vec{w}_{\mathcal{Q}}$ and $\vec{w}'_{\mathcal{Q}}$ be $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{|A|})$, $\vec{\sigma}' = (\sigma'_1, \ldots, \sigma'_{|A|})$, $\vec{w}_{\mathcal{Q}} = (w_1, \ldots, w_{|\mathcal{Q}|})$, and $\vec{w}'_{\mathcal{Q}} = (w'_1, \ldots, w'_{|\mathcal{Q}|})$.*

- *For any output action $a!$, $\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle \rightarrow_a \langle \vec{\sigma}', \vec{w}_{\mathcal{Q}} \rangle$*

  $$\begin{aligned} \textit{if } \exists i. \sigma_i \Rightarrow_{a!} \sigma'_i \wedge (\forall j \neq i. \sigma_j \Rightarrow_{a?} \sigma_j \\ \vee ((\neg \exists \sigma'_j. \sigma_j \Rightarrow_{a?} \sigma'_j) \wedge \sigma_j = \sigma'_j)). \end{aligned}$$

  An output action is broadcasted to all DLHAs, and a DLHA receiving the action moves by synchronization if the guard condition holds in the state.

- *For an internal action a_τ,*
  - *in the case of $a_\tau = q_k!w$, $\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle \rightarrow_{q_k!w} \langle \vec{\sigma}', \vec{w}'_{\mathcal{Q}} \rangle$,*

    $$\textit{if } (\exists i. \sigma_i \Rightarrow_{q_k!w} \sigma'_i \wedge \forall j \neq i. \sigma_j = \sigma'_j) \wedge w'_k = w_k w \wedge \forall l \neq k. w_k = w'_k,$$

  - *while in the case of $a_\tau = q_k?w$, $\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle \rightarrow_{q_k?w} \langle \vec{\sigma}', \vec{w}'_{\mathcal{Q}} \rangle$,*

    $$\textit{if } (\exists i. \sigma_i \Rightarrow_{q_k?w} \sigma'_i \wedge \forall j \neq i. \sigma_j = \sigma'_j) \wedge w_k = w w'_k \wedge \forall l \neq k. w_l = w'_l,$$

  - *otherwise, $\langle \vec{\sigma}, \vec{w}_{\mathcal{Q}} \rangle \rightarrow_{a_\tau} \langle \vec{\sigma}', \vec{w}_{\mathcal{Q}} \rangle$ if $\exists i. \sigma_i \Rightarrow_{a_\tau} \sigma'_i \wedge \forall j \neq i. \sigma_j = \sigma'_j$.*

A *run (or path)* ρ of the system $\mathcal{S}$ is the following finite (or infinite) sequence of states.

$$\rho : s_0 \rightarrow_{a_0}^{\delta_0} s_1 \rightarrow_{a_1}^{\delta_1} \cdots \rightarrow_{a_{i-1}}^{\delta_{i-1}} s_i \rightarrow_{a_i}^{\delta_i} \cdots$$

where $\rightarrow_{a_i}^{\delta_i}$ between s_i and s_{i+1} is defined as

$$s_i \rightarrow_{a_i}^{\delta_i} s_{i+1} \iff \exists s'_i. s_i \rightarrow_{\delta_i} s'_i \wedge s'_i \rightarrow_{a_i} s_{i+1}.$$

The initial state s_0 of a dynamically reconfigurable system is $\langle (\sigma_{01} \ldots, \sigma_{0|A|}), (w_{01}, \ldots, w_{0|\mathcal{Q}|}) \rangle$ where each σ_{0i} is the initial state of DLHA $\mathcal{A}_i$ and each w_{0j} is empty; that is, $\forall j. w_{0j} = \varepsilon$.

Example 1 (DLHA and DRS)**.** *A DLHA is represented by a directed graph, where each node represents a location and each edge represents a transition. Figure 1 shows a dynamically reconfigurable system $\mathcal{S}$ consisting of three DLHAs and one queue.*

$$\begin{aligned}
\mathcal{A}_1 &= (L_1, V_1, Inv_1, Flow_1, Act_1, T_1, t_{01}, T_{d1}),\\
\mathcal{A}_2 &= (L_2, V_2, Inv_2, Flow_2, Act_2, T_2, t_{02}, T_{d2}),\\
\mathcal{A}_3 &= (L_3, V_3, Inv_3, Flow_3, Act_3, T_3, t_{03}, T_{d3}),\\
\mathcal{S} &= (\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}, \{q\})
\end{aligned}$$

where

$$\begin{aligned}
L_1 &= \{Run, Wait\}\\
V_1 &= \{x\}\\
Inv_1 &= \{Run \mapsto x \leq 10, Wait \mapsto true\}\\
Flow_1 &= \{Run \mapsto \dot{x} = 1, Wait \mapsto \dot{x} = 0\}\\
Act_1 &= \{Dst_\mathcal{A}_3?, start_1, q!\mathcal{A}_3\}\\
T_1 &= \{(Run, x \geq 10, q!\mathcal{A}_3, \{\}, Wait),\\
&\quad (Wait, true, Dst_\mathcal{A}_3?, \{x := 0\}, Run)\}\\
t_{01} &= (Run, start_1, \{x := 0\})\\
T_{d1} &= \{\}\\
L_2 &= \{Idle, Create\}\\
V_2 &= \{y\}\\
Inv_2 &= \{Idle \mapsto true, Create \mapsto y \leq 0\}\\
Flow_2 &= \{Idle \mapsto \dot{y} = 1, Create \mapsto \dot{y} = 1\}\\
Act_2 &= \{Crt_\mathcal{A}_3!, start_2, q?\mathcal{A}_3\}\\
T_2 &= \{(Idle, true, q?\mathcal{A}_3, \{y := 0\}, Create),\\
&\quad (Create, y \geq 0, Crt_\mathcal{A}_3!, \{\}, Idle)\}\\
t_{02} &= (Idle, start_2, \{y := 0\})\\
T_{d2} &= \{\}\\
L_3 &= \{Execute\}\\
V_3 &= \{z\}\\
Inv_3 &= \{Execute \mapsto z \leq 50\}\\
Flow_3 &= \{Execute \mapsto \dot{z} = 1\}\\
Act_3 &= \{Crt_\mathcal{A}_3?, Dst_\mathcal{A}_3!\}\\
T_3 &= \{\}\\
t_{03} &= (Execute, Crt_\mathcal{A}_3?, \{z := 0\})\\
T_{d3} &= \{(Execute, z \geq 50, Dst_\mathcal{A}_3!)\}
\end{aligned}$$

This system runs as follows:

1. *$\mathcal{A}_1$ requires $\mathcal{A}_3$ to be created from $\mathcal{A}_2$ by enqueueing a message, and it waits for the message to return from $\mathcal{A}_3$.*
2. *When $\mathcal{A}_2$ receives the message, it creates $\mathcal{A}_3$.*
3. *After $\mathcal{A}_3$ finishes processing the job, it sends the message to $\mathcal{A}_1$ and is destroyed.*
4. *This system infinitely repeats steps 1) to 3).*

For example, (1) shows a run ρ of this system is shown.

$$\begin{aligned}\rho :&\langle((\mathit{Run}, x=0),(\mathit{Idle}, y=0),\bot),(\varepsilon)\rangle\\ &\to^{10}_{q!\mathcal{A}_3} \langle((\mathit{Wait}, x=10),(\mathit{Idle}, y=0),\bot),(\mathcal{A}_3)\rangle\\ &\to^{0}_{q?\mathcal{A}_3} \langle((\mathit{Wait}, x=10),(\mathit{Create}, y=0),\bot),(\varepsilon)\rangle\\ &\to^{0}_{\mathit{Crt\_}\mathcal{A}_3} \langle((\mathit{Wait}, x=10),(\mathit{Idle}, y=0),\\ &\qquad\qquad (\mathit{Execute}, z=0)),(\varepsilon)\rangle\\ &\to^{50}_{\mathit{Dst\_}\mathcal{A}_3} \langle((\mathit{Run}, x=0),(\mathit{Idle}, y=0),\bot),(\varepsilon)\rangle\\ &\to \cdots\end{aligned} \tag{1}$$

4 Reachability Analysis

4.1 Reachability Problem

We define reachability and the reachability problem for a DRS as follows:

Definition 8 (Reachability)**.** *For a DRS $\mathcal{S} = (A, \mathcal{Q})$ and a location l_t, $\mathcal{S}$ reaches l_t if there exists a path such that*

$$\begin{aligned}&s_0 \to^{\delta_0}_{a_0} \cdots \to^{\delta_{t-1}}_{a_{t-1}} s_t\\ &\wedge s_t = \langle(\sigma_1, \ldots, \sigma_{|A|}), \vec{w}_{\mathcal{Q}}\rangle, \exists k.\mathit{loc}\,(\sigma_k) = l_t,\end{aligned}$$

where

$$\mathit{loc}\,(\sigma) = \begin{cases} l & (\sigma = (l, \nu)) \\ \bot\ (\mathit{undefined}) & (\sigma = \bot) \end{cases}$$

Definition 9 (Reachability Problem)**.** *Given a DRS $\mathcal{S} = (A, \mathcal{Q})$ and a location l_t, we output "yes" if $\mathcal{S}$ can reach l_t, and "no" otherwise.*

4.2 Reachability Analysis

4.2.1 Convex Polyhedra

Our method introduces *convex polyhedra* for the reachability analysis in accordance with [17].

A polyhedron is convex if it can be defined by a formula which is a conjunction of linear formulae. For a set $V = \{x_1, \ldots, x_n\}$ of variables, a convex polyhedron ζ on V is a n-dimensional real space. In particular, we define *true* and *false* as $\mathit{true} = \mathbb{R}^n$ and $\mathit{false} = \varnothing$.

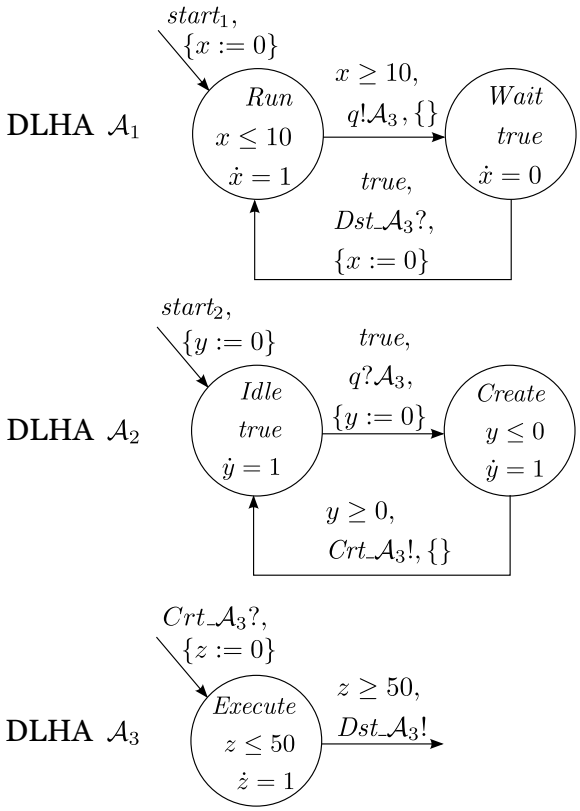

Figure 1: Example of DRS consisting of three DLHAs and one queue

Example 2 (Convex Polyhedron)**.** *The formula* $\exists x_1.\ x_1 \geq 5 \wedge x_2 \leq 1 \wedge x_1 - x_2 = 1 \wedge \mathit{true}$ *is a convex polyhedron. From linear formula, the existential quantifier can be eliminated effectively. Therefore, we obtain*

$$\begin{aligned}
&\exists x_1.\ x_1 \geq 5 \wedge x_2 \leq 1 \wedge x_1 - x_2 = 1 \wedge \mathit{true} \\
&= \exists x_1.\ x_1 \geq 5 \wedge x_2 \leq 1 \wedge x_1 - x_2 = 1 \\
&= x_2 \leq 1 \wedge x_2 \geq 4 \\
&= \mathit{false}.
\end{aligned}$$

4.2.2 Algorithm of Reachability Analysis

We define a state s in the reachability analysis as $(L, \zeta, \vec{w}_{\mathcal{Q}})$, where

- L is a finite set of locations.
- ζ is a convex polyhedron.
- $\vec{w}_{\mathcal{Q}}$ is a vector of the content of the queues.

Figure 2, Figure 3 and Figure 4 show the algorithm of the reachability analysis.

Figure 2 is an overview of the reachability analysis, and this algorithm is performed using the expanded method of [13] with a set $\mathcal{Q}$ of queues. The analysis is performed as follows:

1. Compute an initial state s_0 of the system $\mathcal{S}$ (ll.1–3).
2. Initialize a traversed set Visit and a untraversed set Wait of states by $\varnothing$ and $\{s_0\}$ (line 4).
3. While Wait is not empty, repeat the following process (ll.5–16).
   (a) Take a state $(L, \zeta, \vec{w_{\mathcal{Q}}})$ from Wait and remove the state from Wait (ll.6–7).
   (b) If the set L of locations contains the target location, return "yes" and terminate (ll.8–10).
   (c) If the state has not been traversed yet ($(L, \zeta, \vec{w_{\mathcal{Q}}}) \notin$ Visit) (line 11),
      i. add the state into Visit (line 12),
      ii. compute the set S_{post} of successors by using the subroutine Succ (line 13), and
      iii. add all components of S_{post} to Wait (line 14).

Subroutine Succ Figure 3 shows the subroutine Succ to compute the successors of a state. In this algorithm, we make the following assumptions.

$$\mathcal{S} = (A, \mathcal{Q}) = (\{\mathcal{A}_1, \ldots, \mathcal{A}_{|A|}\}, \{q_1, \ldots, q_{|\mathcal{Q}|}\}),$$
$$Inv_s = \bigcup_{k=1}^{|A|} Inv_k,$$

where

$$\mathcal{A}_i = (L_i, V_i, Inv_i, Flow_i, Act_i, T_i, t_{0i}, T_{di}) \text{ is a DLHA},$$
$$t_{0i} = (l_{0i}, a_{0i}, \lambda_{0i}).$$

Let the initial state of $\mathcal{S}$ be $\langle(\sigma_{01}, \ldots, \sigma_{0|A|})\vec{w}_{\mathcal{Q}0}\rangle$; s_0 is $(L_0, \zeta_0, \vec{w}_{\mathcal{Q}0})$, where

$$\text{zone}\,(\sigma) = \begin{cases} \nu & (\sigma = (l, \nu)) \\ true & (\sigma = \perp), \end{cases}$$
$$L_0 = \{\text{loc}\,(\sigma_{0i}) \mid \sigma_{0i} \neq \perp, i \in \{1, \ldots, |A|\}\},$$
$$\zeta_0 = \bigwedge_{i=0}^{|A|} \text{zone}\,(\sigma_{0i})\,.$$

Here, zone(σ) is a function that assigns a convex polyhedron to each state.

Tsucc(L, ζ) is a function that returns a convex polyhedron after performing a time transition on a given set L of locations and a convex polyhedron ζ (line 4). We define this function in accordance with [17] as follows:

Let the set of all variables in the system and their derivatives be $V_s = \bigcup_{k=1}^{|A|} V_k = \{x_1, \ldots, x_n\}$ and $\dot{V}_s = \{\dot{x}_1, \ldots, \dot{x}_n\}$.

$$\text{Tsucc}(L, \zeta) = \exists x_1, \ldots, x_n \in V_s. \exists \delta \in \mathbb{R}_{\geq 0}. \exists \dot{x}_1, \ldots, \dot{x}_n \in \dot{V}_s. \\ \zeta \wedge \mathit{Flow}(L) \wedge \bigwedge_{x \in V_s} x' = x + \delta \dot{x} \text{ and rename } x' \text{ as } x,$$

where

$$\mathit{Flow}_s = \bigcup_{k=1}^{|A|} \mathit{Flow}_k,$$
$$\mathit{Flow} = \bigwedge_{l \in L} \mathit{Flow}_s(l).$$

For a convex polyhedron ζ and a set λ of update expressions, $\zeta[\lambda]$ denotes the convex polyhedron updated by λ for ζ. Let the set of reset variables and set of shifted variables be $V_r = \{x \mid x := c \in \lambda\} = \{x_{r1}, \ldots, x_{rm}\}$ and $V_a = \{x \mid x := x + c \in \lambda\} = \{x_{a1}, \ldots, x_{an}\}$. $\zeta[\lambda]$ can be computed as

$$\zeta[\lambda] = (\exists x_{r1}, \ldots, x_{rm} \in V_r.\ \zeta_a) \wedge \bigwedge_{x \in V_r} x = m_r(x),$$

where

$$m_r = \{x \mapsto c \mid x := c \in \lambda\},$$
$$m_a = \{x \mapsto c \mid x := x + c \in \lambda\},$$
$$\zeta_a = \exists x_{a1}, \ldots, x_{an} \in V_a.\ \zeta \wedge \bigwedge_{x \in V_a} x' = x + m_a(x) \text{ and rename variables } x' \text{ as } x.$$

Given a state $(L, \zeta, \vec{w}_{\mathcal{Q}})$ and a system, the successors are computed using the procedure described below.

1. For each transition $(l, \phi, a, \lambda, l')$ (or destruction-transition $(l, \phi, a_l!)$) outgoing from a location $l \in L$, the set S_{post} of post states is computed as follows (ll.5–31):
   (a) Compute the convex polyhedron for the time transition (line 4).
   $$\zeta_\delta = \text{Tsucc}(L, \zeta) \wedge \bigwedge_{l_p \in L} \mathit{Inv}_s(l_p).$$
   (b) If a is an internal action, S_{post} is computed as follows:
      i. Compute the set of locations (line 8)
      $$L' = (L \setminus \{l\}) \cup \{l'\}.$$

ii. Compute the convex polyhedron for the discrete transition (line 9)

$$\zeta' = (\zeta \wedge \phi)[\lambda] \wedge \bigwedge_{l'_p \in L'} Inv_s(l'_p).$$

iii. If a is an enqueue action $q_k!w$ (ll.11–15),

$$S_{post} = \begin{cases} \{(L', \zeta', \vec{w}'_{\mathcal{Q}})\} & (\zeta' \neq false) \\ \varnothing & (otherwise), \end{cases}$$

where

$$\vec{w}'_{\mathcal{Q}} = (w_1, \ldots, w_{k-1}, w_k \cdot w, w_{k+1}, \ldots, w_{|\mathcal{Q}|}).$$

iv. If a is a dequeue action $q_k?w$ (ll.16–20),

$$S_{post} = \begin{cases} \{(L', \zeta', \vec{w}'_{\mathcal{Q}})\} & (\zeta' \neq false, w_k = w \cdot w'_k, \\ & \qquad \forall j \neq k. w_j = w'_j) \\ \varnothing & (otherwise). \end{cases}$$

v. If a is another internal action (line 22),

$$S_{post} = \begin{cases} \{(L', \zeta', \vec{w}_{\mathcal{Q}})\} & (\zeta' \neq false) \\ \varnothing & (otherwise). \end{cases}$$

(c) If a is an output action $a_l!$, S_{post} is computed with the subroutine Syncs (line 26 and 29).

(d) If a is an input action, $S_{post} = \varnothing$.

Subroutine Syncs Figure 4 shows the subroutine Syncs of Succ to compute successors by using the transition that has an output action. Given a state $(L, \zeta, \vec{w_{\mathcal{Q}}})$, a transition (or destruction-transition) $t_s = (l, \phi_g, a_l!, \lambda, l')$, and a system $\mathcal{S} = (A, \mathcal{Q})$, a set S_{post} of successors is computed as follows:

1. Initialize S_{post} as $\varnothing$ (line 1).

2. Compute a convex polyhedron ζ_δ for the time transition (line 2).

$$\zeta_\delta = \mathrm{Tsucc}(L, \zeta) \wedge \bigwedge_{l_p \in L} Inv_s(l_p).$$

3. For each $\mathcal{A}_i$ in the system $\mathcal{S}$, compute the set T_{si} of transitions that are outgoing from the state by using an input action $a_l?$ (ll.3–5),

$$T_{si} = \{(l_i, \phi_i, a_l?, \lambda_i, l'_i) \in T_i \mid l_i \in (L \setminus \{l\})\}.$$

4. Compute the set Δ of combinations of T_{si} (line 6).

$$\Delta = \prod\{T_{si} | T_{si} \neq \varnothing, i \in \{1, \ldots, |A|\}\}.$$

5. For each combination $T = (t_1, \ldots, t_n) \in \Delta$, the successor $(L'_T, \zeta'_T, \vec{w}_{\mathcal{Q}})$ is computed as follows (ll.7–29):

   (a) Compute the set T_{sync} of transitions (line 9).

$$T_{sync} = \max_{|\Delta'|} \Delta' \subseteq \{t_1, \ldots, t_n\}$$
$$\text{s. t. } \zeta_\delta \wedge \phi \wedge \bigwedge\{\phi_s \mid (l_1, \phi_s, a_l?, \lambda_s, l_2) \in \Delta'\} \neq \mathit{false}.$$

   (b) Compute the set L'_T of locations (ll.9–14, line 21).

$$L'_T = (L \setminus L_{pre}) \cup L_{post},$$

   where

$$L_{sync} = \{l_1 \mid (l_1, \phi_s, a_l?, \lambda_s, l_2) \in T_{sync}\},$$
$$L'_{sync} = \{l_2 \mid (l_2, \phi_s, a_l?, \lambda_s, l_2) \in T_{sync}\},$$
$$L_{pre} = \{l\} \cup L_{sync}$$
$$L_{post} = \begin{cases} \{l', l_{j_0}\} \cup L'_{sync} & (a_l = \mathit{Crt\_}\mathcal{A}_j, L \cap L_j = \varnothing) \\ L'_{sync} & (a_l = \mathit{Dst\_}\mathcal{A}_j) \\ \{l'\} \cup L'_{sync} & (\mathit{otherwise}). \end{cases}$$

   (c) Compute the update expression λ_{sync} (ll.9–14).

$$\lambda_{sync} = \begin{cases} \lambda \cup \lambda_{0j} \cup \lambda_{in} & (a_l = \mathit{Crt\_}\mathcal{A}_j, L \cap L_j = \varnothing) \\ \lambda_{in} & (a_l = \mathit{Dst\_}\mathcal{A}_j) \\ \lambda \cup \lambda_{in} & (\mathit{otherwise}), \end{cases}$$

   where

$$\lambda_{in} = \bigcup\{\lambda_s \mid (l_1, \phi_s, a_l?, \lambda_s, l_2) \in T_{sync}\}.$$

   (d) Compute the conjunction of guard conditions (ll.9–14).

$$\phi_{sync} = \phi \wedge \bigwedge\{\phi_s \mid (l_1, \phi_s, a_l?, \lambda_s, l_2) \in T_{sync}\}.$$

   (e) Compute the convex polyhedron ζ'_T (ll.22–24).

$$\zeta'_T = \begin{cases} \exists x_{j_1}, \ldots, x_{j_{|V_j|}} \in V_j.\zeta' & (a_l = \mathit{Dst\_}\mathcal{A}_j) \\ \zeta' & (\mathit{otherwise}), \end{cases}$$

   where

$$\zeta' = (\zeta_\delta \wedge \phi_{sync})[\lambda_{sync}] \wedge \bigwedge_{l'_p \in L'} \mathit{Inv}_s(l'_p).$$

Input: a system $\mathcal{S}$ and a target location l_t
Output: "yes" or "no"
1: $L_0 \leftarrow \{l_{0i} \mid t_{0i} = (l_{0i}, a_{0i}, \lambda_{0i}), a_{0i} \neq \mathit{Crt\_}\mathcal{A}_i?\}$
2: $\lambda_0 \leftarrow \bigcup\{\lambda_{0i} \mid t_{0i} = (l_{0i}, a_{0i}, \lambda_{0i}), a_{0i} \neq \mathit{Crt\_}\mathcal{A}_i?\}$
3: $s_0 \leftarrow (L_0, \vec{0}[\lambda_0], (\varepsilon, \ldots, \varepsilon))$ /* Compute the initial state */
4: Visit $\leftarrow \varnothing$, Wait $\leftarrow \{s_0\}$ /* Initialize */
5: **while** Wait $\neq \varnothing$ **do**
6: $(L, \zeta, \vec{w}_{\mathcal{Q}}) \leftarrow s \in$ Wait
7: Wait $\leftarrow$ Wait $\setminus \{(L, \zeta, \vec{w}_{\mathcal{Q}})\}$
8: **if** $l_t \in L$ **then**
9: **return** "yes"
10: **end if**
11: **if** $(L, \zeta, \vec{w}_{\mathcal{Q}}) \notin$ Visit **then**
12: Visit $\leftarrow$ Visit $\cup \{(L, \zeta, \vec{w}_{\mathcal{Q}})\}$
13: $S_{post} \leftarrow \mathrm{Succ}((L, \zeta, \vec{w}_{\mathcal{Q}}), \mathcal{S})$ /* Compute the set of post-states */
14: Wait $\leftarrow$ Wait $\cup S_{post}$
15: **end if**
16: **end while**
17: **return** "no"

Figure 2: Reachability Analysis

(f) If $\zeta'_T \neq \mathit{false}$, the successor is added to S_{post} (ll.26–27).

The correctness of this algorithm is implied by Lemma 1 and Lemma 2.

Lemma 1. *If the algorithm terminates and returns "l_t is not reachable", the system $\mathcal{S}$ has the safety property.*

Lemma 2. *If this algorithm terminates and returns "l_t is reachable", the system $\mathcal{S}$ does not hold the safety property.*

By definition, all linear hybrid automata are DLHAs. Our system dynamically changes its structure by sending and receiving messages. However, the messages statically determine the structure, and the system is a linear hybrid automaton with a set of queues. It is basically equivalent to the reachability analysis of a linear hybrid automaton. Therefore, the reachability problem of DRSs is undecidable, and this algorithm might not terminate [13].

Moreover, in some cases, a system will run into an abnormal state in which the length of a queue becomes infinitely long, and the verification procedure does not terminate.

5 Practical Experiment

5.1 Model Checker

We implemented a model checker of DRSs consisting of DLHAs in Java (about 1,600 lines of code) by using the LAS, PPL, and QDD external libraries [10, 14, 15, 16]. For the verification,

Input: a state $(L, \zeta, \vec{w}_{\mathcal{Q}})$ and the system $\mathcal{S}$
Output: the set S_{post} of post-states

1: $T_N \leftarrow \bigcup_{i=1}^{|A|} \{(l, \phi_g, a, \lambda, l') \in T_i \mid l \in L\}$ /* Set of outgoing transitions */
2: $T_D \leftarrow \bigcup_{i=1}^{|A|} \{(l, \phi_g, a) \in T_{di} \mid l \in L\}$ /* Set of outgoing destruction-transitions */
3: $S_{post} \leftarrow \varnothing$, $T_{post} \leftarrow T_N \cup T_D$
4: $\zeta_\delta \leftarrow \mathrm{Tsucc}(L, \zeta) \wedge \bigwedge_{l_p \in L} Inv_s(l_p)$ /* Convex polyhedron for the time transition */
5: **for all** $t \in T_{post}$ **do**
6: **if** $t = (l, \phi_g, a, \lambda, l')$ **then**
7: **if** a is an internal action **then**
8: $L' \leftarrow (L \setminus \{l\}) \cup \{l'\}$ /* Locations of the post-state */
9: $\zeta' \leftarrow (\zeta_\delta \wedge \phi_g)[\lambda] \wedge \zeta'_i \wedge \bigwedge_{l'_p \in L'} Inv_s(l'_p)$ /* The convex polyhedron of the post-states */
10: **if** $\zeta' \neq false$ **then**
11: **if** a is a enqueue action $q_k!w$ **then**
12: $(w_1, \ldots, w_{|\mathcal{Q}|}) \leftarrow \vec{w}_{\mathcal{Q}}$
13: $w'_k \leftarrow w_k w$ /* Enqueue the message into q_k */
14: $\vec{w}'_{\mathcal{Q}} \leftarrow (w_1, \ldots, w_{k-1}, w'_k, w_{k+1}, \ldots, w_{|\mathcal{Q}|})$
15: $S_{post} \leftarrow S_{post} \cup \{(L', \zeta', \vec{w}'_{\mathcal{Q}})\}$
16: **else if** a is a dequeue action $q_k?w$ **then**
17: **if** $w_k = w w'_k$ **then**
18: $\vec{w}'_{\mathcal{Q}} \leftarrow (w_1, \ldots, w_{k-1}, w'_k, w_{k+1}, \ldots, w_{|\mathcal{Q}|})$ /* Dequeue the message from q_k */
19: $S_{post} \leftarrow S_{post} \cup \{(L', \zeta', \vec{w}'_{\mathcal{Q}})\}$
20: **end if**
21: **else**
22: $S_{post} \leftarrow S_{post} \cup \{(L', \zeta', \vec{w}_{\mathcal{Q}})\}$ /* For other internal action */
23: **end if**
24: **end if**
25: **else if** a is an output action $a_l!$ **then**
26: $S_{post} \leftarrow S_{post} \cup \mathrm{Syncs}((L, \zeta, \vec{w}_{\mathcal{Q}}), t, \mathcal{S})$ /* Compute the set of states using the synchronous transitions */
27: **end if**
28: **else**
29: $S_{post} \leftarrow S_{post} \cup \mathrm{Syncs}((L, \zeta, \vec{w}_{\mathcal{Q}}), t, \mathcal{S})$ /* Compute the set of states using the destruction-transition */
30: **end if**
31: **end for**
32: **return** S_{post}

Figure 3: Subroutine Succ

we input the DLHAs of the system, a *monitor automaton*, and the *error location* to the model checker, and it output "yes (reachable)" or "no (unreachable)" (Figure 5). The monitor automaton had a special location (we call it the error location), and checked the system without

Input: a state $(L, \zeta, \vec{w}_{\mathcal{Q}})$, a transition (or destruction-transition) $t_s = (l, \phi_g, a_l!, \lambda, l') \mid (l, \phi_g, a_l!)$ and the system $\mathcal{S}$
Output: the set S_{post} of post-states

1: $S_{post} \leftarrow \varnothing$
2: $\zeta_\delta \leftarrow \text{Tsucc}(L, \zeta) \wedge \bigwedge_{l_p \in L} Inv_s(l_p)$ /* Convex polyhedron for the time transition */
3: **for** $i = 1$ **to** $|A|$ **do**
4: $T_{si} \leftarrow \{(l_i, \phi_i, a_i, \lambda_i, l_i') \in T_i \mid l_i \in (L \setminus \{l\}), a_i = a_l?, \zeta_\delta \wedge \phi_i \neq \mathit{false}\}$ /* Synchronized transitions of $\mathcal{A}_i$ */
5: **end for**
6: $\Delta \leftarrow \prod\{T_{si} \mid T_{si} \neq \varnothing, i \in \{1, \ldots, |A|\}\}$ /* Combinations of transitions */
7: **for all** $(t_1, \ldots, t_n) \in \Delta$ **do**
8: $T_{sync} \leftarrow \max_{|\Delta'|} \Delta' \subseteq \{t_1, \ldots, t_n\}$ s. t. $\zeta_\delta \wedge \phi \wedge \bigwedge\{\phi_s \mid (l_1, \phi_s, a_l?, \lambda_s, l_2) \in \Delta'\} \neq \mathit{false}$ /* The set of transitions synchronized with t_s */
9: $L_{pre} \leftarrow \{l\}$, $L_{post} \leftarrow \varnothing$, $\phi \leftarrow \phi_g$, $\lambda_u \leftarrow \varnothing$
10: **for all** $t_{in} \in T_{sync}$ **do**
11: $(l_i, \phi_i, a_i, \lambda_i, l_i') \leftarrow t_{in}$
12: $L_{pre} \leftarrow L_{pre} \cup \{l_i\}$, $L_{post} \leftarrow L_{post} \cup \{l_i'\}$ /* Pre-locations and post-locations of T_{sync} */
13: $\phi \leftarrow \phi \wedge \phi_i$, $\lambda_u \leftarrow \lambda_u \cup \lambda_i$ /* Guard conditions and update expressions */
14: **end for**
15: **if** $t_s = (l, \phi_g, a, \lambda, l')$ **then**
16: $L_{post} \leftarrow L_{post} \cup \{l'\}$, $\lambda_u \leftarrow \lambda_u \cup \lambda$ /* Add the post-location and the update expressions of t_s */
17: **end if**
18: **if** $a_l = \mathit{Crt\_}\mathcal{A}_j$ and $L_j \cap L = \varnothing$ **then**
19: $L_{post} \leftarrow L_{post} \cup \{l_{0j}\}$, $\lambda_u \leftarrow \lambda_u \cup \lambda_{0i}$ /* If $\mathcal{A}_j$ is not yet created, add the initial location and update expressions */
20: **end if**
21: $L_T' \leftarrow (L \setminus L_{pre}) \cup L_{post}$ /* Locations of the post-state */
22: $\zeta_T' \leftarrow (\zeta_\delta \wedge \phi)[\lambda_u] \wedge \bigwedge_{l_p' \in L'} Inv_s(l_p')$ /* The convex polyhedron of the post-state */
23: **if** $t_s = (l, \phi_g, \mathit{Dst\_}\mathcal{A}_i!)$ **then**
24: $\zeta_T' \leftarrow \exists V_i.\zeta_T'$ /* If t_s is a destruction-transition, free variables for the convex polyhedron. */
25: **end if**
26: **if** $\zeta_T' \neq \mathit{false}$ **then**
27: $S_{post} \leftarrow S_{post} \cup \{(L_T', \zeta_T', \vec{w}_{\mathcal{Q}})\}$ /* If the transition is possible, add the post-state. */
28: **end if**
29: **end for**
30: **return** S_{post}

Figure 4: Subroutine Syncs

changing the system's behavior [17]. The monitor automata had to be specified to reach the

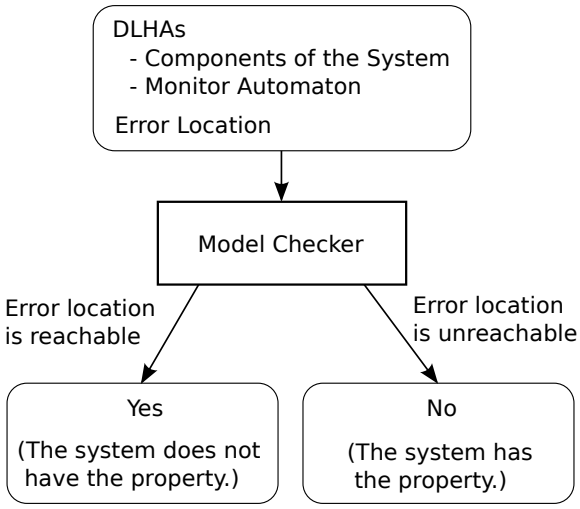

Figure 5: Model Checker for DRSs

error location if the system didn't satisfy the properties.

For the specification of the input model, we extended the syntax and semantics of DLHA as follows:

- A transition between locations can have a label *asap* (that means 'as soon as possible'). For a transition labeled *asap*, a time transition does not occur just before the discrete transition.
- Each DLHA can have constraints and update expressions for the variables of another DLHA in the same system. That is, for each DLHA, invariants, guard conditions, update expressions and flow conditions can be used by all DLHAs.

For example, Figure 6 shows the input file for checking whether the system in Figure 1 reaches the location *Execute*.

5.2 Specification of Dynamically Reconfigurable Embedded System

5.2.1 A cooperative system including CPU and DRP

We have specified a dynamically reconfigurable embedded system consisting of a CPU and DRP for the model described in our previous research [7]. A DRP is a processor that can execute exclusive processes at the same time by dynamically changing the circuit configuration, and it is used to accelerate CPU computations, for example, in image processing and cipher processing. A DRP has computation resources called *tiles* (or *processing elements*), and it dynamically sets the context of a process if there are enough free tiles. In addition, a DRP can change the operating frequency in accordance with running processes. In this paper, we assume that the number of tiles and the operating frequency for each process have been set in advance and that the operating frequency of the DRP is always the minimum frequency of the running co-tasks.

```
target: Execute
DLHA:
    A1 {
        var: x
        loc Run: x <= 10 [(x,1)]
        loc Wait: true [(x,0)]
        Run -> Wait: x >= 10, q!A3 []
        Wait -> Run: true, DST?A3 [x:=0]
        init: Run, start1 [x:=0]
    }
    A2 {
        var: y
        loc Idle: true [(y,1)]
        loc Create: y <= 0 [(y,1)]
        Idle -> Create: true, q?A3 [y:=0]
        Create -> Idle: y >= 0, CRT!A3 []
        init: Idle, start2 [y:=0]
    }
    A3 {
        var: z
        loc Execute: z <= 50 [(z,1)]
        init: Execute, CRT?A3 [z:=0]
        fin: Execute, z >= 50, DST!A3
    }
```

Figure 6: Example input file: description for checking the reachability of the system in Figure 1

Figure 7 shows an overview of the system. This system processes jobs submitted from the external environment through the cooperative operation of the CPU and DRP. The CPU Dispatcher creates a task when it receives a call message of the task from the external environment. When a task on the CPU uses the DRP, The CPU Dispatcher sends a message to the DRP Dispatcher. The DRP Dispatcher receives the message asynchronously and creates a *co-task* (it means 'cooperative task') in a first-come, first-served manner if there are enough free tiles. Here, we will assume that this system has two tasks and two co-tasks that have the parameters shown in Table 1-2.

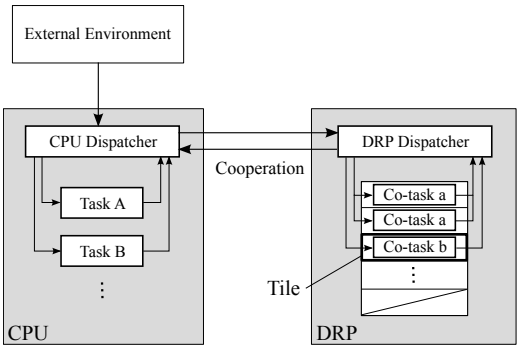

Figure 7: Overview of the CPU-DRP embedded system

The system, whose components are illustrated in Figure 8, consists of 11 DLHAs and 1 queue. We show part of the state-transition diagram in Figure 9. The external environment consists of EnvA (Figure 10) and EnvB (Figure 11) that periodically create TaskA (Figure 13)

Task	Period	Deadline	Priority	Process
A	70 ms	70 ms	high	20 ms, co-task a0, 10 ms, co-task b0
B	200 ms	200 ms	low	co-task a1, 97 ms

Table 1: Parameters of tasks

co-task	Processing time	Deadline	Tiles	Rate of Frequency
$a0, a1$	10 ms	15 ms	2	1
$b0$	5 ms	10 ms	6	1/2

Table 2: Parameters of co-tasks

and TaskB (Figure 14). That is, EnvA uses *Crt_taskA*! to create TaskA every 70 milliseconds, and EnvB uses *Crt_taskB*! to create TaskB with every 200 milliseconds. The Scheduler (Figure 12) performs scheduling in accordance with the priority and actions for creation and destruction of DLHAs. For example, when TaskA is created by EnvA with *Crt_taskA*! and TaskB is already running, The Scheduler receives *Crt_taskA?* from EnvA and sends *Act_Preempt*! to TaskA and TaskB. Then, *Act_Preempt*! causes TaskA to move to *RunA* and TaskB to move to *WaitB*.

TaskA and TaskB send a message to The Sender if they need a co-task. The Sender (Figure 15) enqueues the message to create a co-task to q when it receives a message from tasks. When TaskA sends *Act_Create_a0*! and moves to *RunA* from *WaitA*, The Sender receives *Act_Create_a0*? and enqueues *cotask_a0* in q with $q!$*cotask_a0*.

The DRP_Dispatcher (Figure 16) dequeues a message and creates cotask_a0 (Figure 18), cotask_a1 (Figure 19), and cotask_b0 (Figure 20) if there are enough free tiles. The Frequency_Manager (Figure 17) is a module that manages the operating frequency of the DRP. When a DLHA of a co-task is created, The Frequency_Manager moves to the location that sets the frequency to the minimum value.

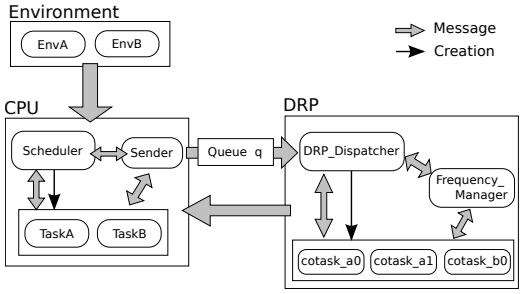

Figure 8: Components of the system

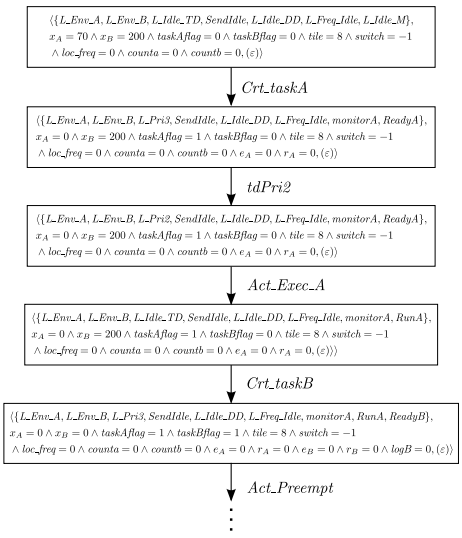

Figure 9: State-transition diagram of the system

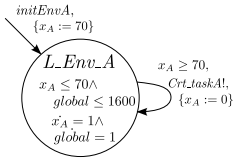

Figure 10: External environment: EnvA

Task	Period	Deadline	Priority	Process
A	90 ms	80 ms	high	20 ms, co-task b0, 20 ms, co-task a0
B	200 ms	150 ms	low	co-task a1, 70 ms

Table 3: Modified parameters of tasks

5.2.2 Other cases

We have the parameters of the model in subsection 5.2.1 and conducted experiments with it.

- Modified Tasks: We modified the parameters of the tasks on the CPU as shown in Table 3. Here, the parameters of the co-tasks are the same as those in Table 2.
- Modified co-tasks: We modified the parameters of the co-tasks on the DRP, as shown in Table 4. The parameters of the tasks are the same as those in Table 1.

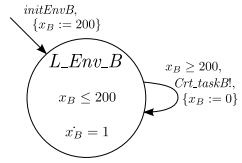

Figure 11: External environment: EnvB

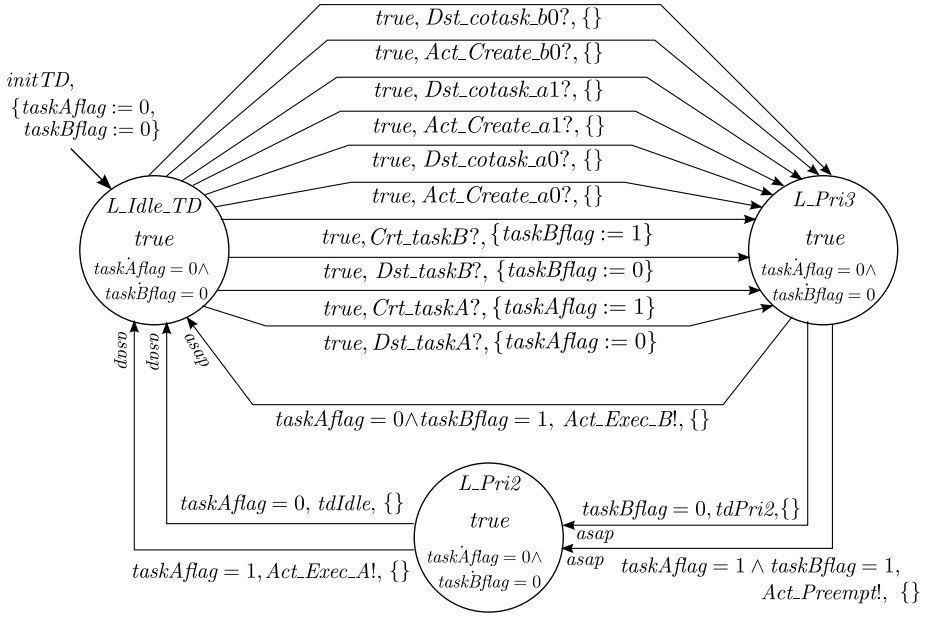

Figure 12: CPU Scheduler: Scheduler

co-task	Processing time	Deadline	Tiles	Rate of Frequency
$a0, a1$	5 ms	10 ms	4	1
$b0$	10 ms	20 ms	5	1/3

Table 4: Modified parameters of co-tasks

5.3 Verification Experiment

We verified that the embedded systems described in subsection 5.2 provide the following properties by using monitor automata (Figure 21-25). The verification experiment was performed on a machine with an Intel (R) Core (TM) i7-3770 (3.40GHz) CPU and 16GB RAM running Gentoo Linux (3.10.25-gentoo).

The experimental results shown in Table 5 indicate that the modified tasks cases and the

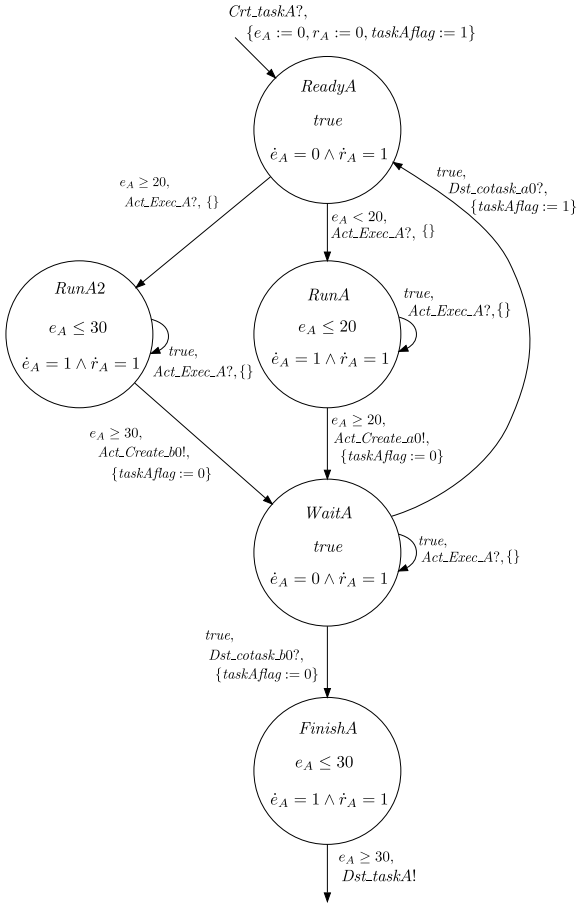

Figure 13: Task: TaskA

modified co-tasks cases were verified with less computation resources (memory and time) than were used by the original model. This reduction is likely due to the following reasons:

- Regarding the schedulability of the modified tasks model, the processing time is shorter than that of the original model since the verification terminates if a counterexample is found.
- In the cases of the modified co-tasks, the most obvious explanation is that the state-space is smaller than that of the original model since the number of branches in the search tree (i.e. nondeterministic transitions in this system) is reduced by changing the start timings of the tasks and co-tasks with the parameters.
- In cases other than those of the modified tasks, it is considered that the state-space is smaller than that of the original model because this system is designed to stop processing when a task exceeds its deadline.

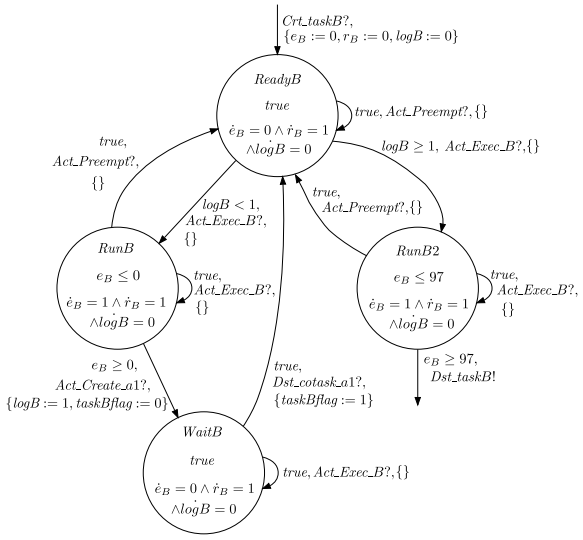

Figure 14: Task: TaskB

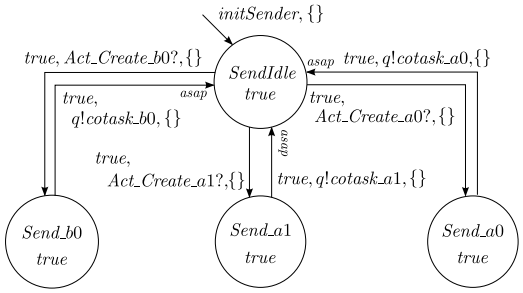

Figure 15: Message sender to DRP: Sender

5.3.1 Schedulability

Here, schedulability is a property in which each task of the system finishes before its deadline. Let E_A be the total processing time and D_A be the deadline in task A (Figure 13); the remaining processing time is represented as $E_A - e_A$, and the remaining time till the deadline is represented as $D_A - r_A$. Therefore, the monitor automaton moves the error location if the task A is created and it satisfies the condition $E_A - e_A > D_A - r_A$. In the case of Table 1, $E_A - e_A > D_A - r_A \iff 30 - e_A > 70 - r_A \iff r_A - e_A > 40$ since $E_A = 30$ and $D_A = 70$. Similarly, the condition for task B is $r_B - e_B > 103$.

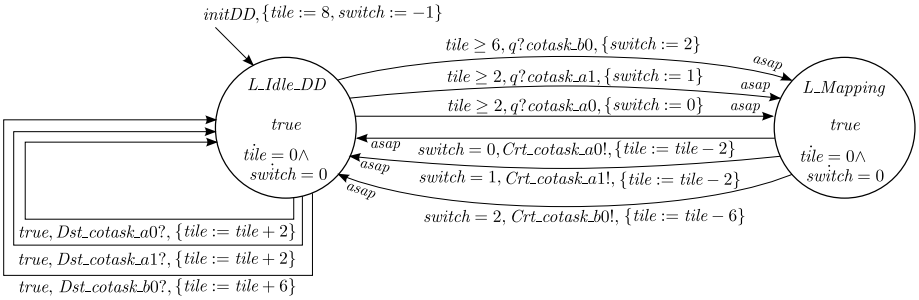

Figure 16: DRP_Dispatcher

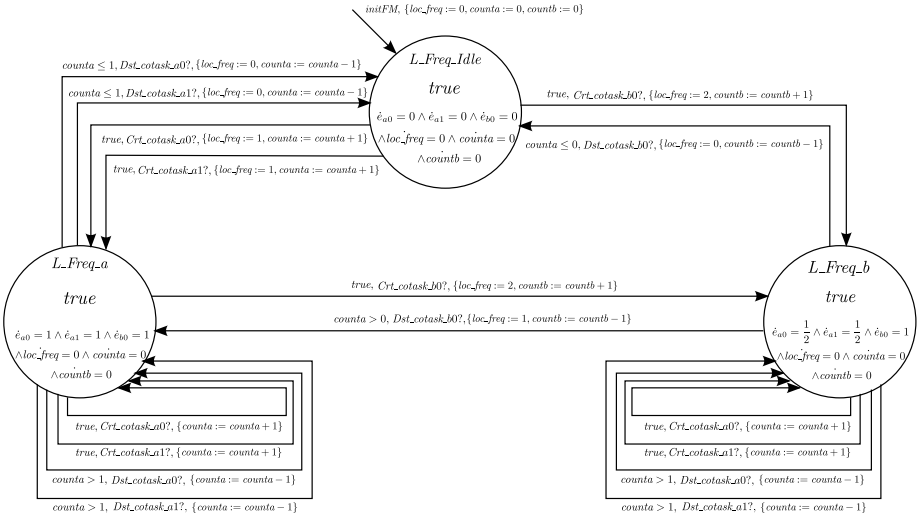

Figure 17: Frequency_Manager

5.3.2 Creation of co-tasks

In the embedded system, each co-task must be created before the remaining time in the task calling it reaches its deadline. When the message $create\_a0$ is received from task A, the monitor automaton starts counting time for co-task $a0$. If the waiting time exceeds the deadline of task A before it receives the message Crt_cotask_a0, the monitor moves to error location. Figure 22 shows The monitor automaton for the case of Table 1 for co-task $a0$. Monitor automata for co-tasks $a1$ and $b0$ can be similarly described.

5.3.3 Destruction of co-tasks

Each co-task must be destroyed before the waiting time reaches its deadline. For the co-task $a0$, when the message *Crt_cotask_a0* is received from the dispatcher DRP_Dispatcher, the monitor automaton checks the message *Dst_cotask_a0*. Figure 23 shows the monitor automaton for the

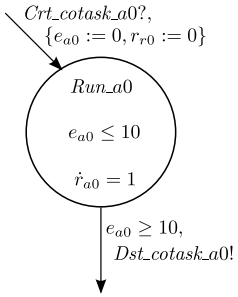

Figure 18: co-task: cotask_a0

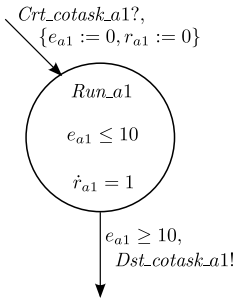

Figure 19: co-task: cotask_a1

case of Table 2.

5.3.4 Frequency management

Creating or destroying a co-task, the DRP changes the operating frequency corresponding to the co-tasks being processed. Since this system requires that the frequency is always at the minimum value, the monitor checks whether the frequency manager (Frequency_Manager) moves to the correct location when it receives a message for creating a co-task. For example, when co-task $a0$ and co-task $b0$ are running on the DRP, Frequency_Manager must be at location *L_Freq_b*. Figure 24 show the monitor automaton for the case of Table 2.

5.3.5 Tile Management

When the DRP receives a message for creating of a co-task and the number of free tiles is enough to process it, the dispatcher creates the co-task. The dispatcher then updates the number of used tiles. The monitor automaton checks whether the number *tiles* in DRP_Dispatcher is always between 0 and the maximum number, 8 in this case (Figure 25).

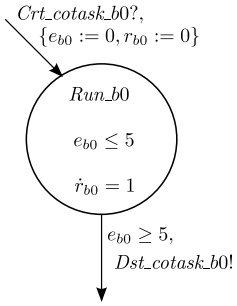

Figure 20: co-task: cotask_b0

6 Conclusion and future work

We proposed a dynamic linear hybrid automaton (DLHA) as a specification language for dynamically reconfigurable systems. We also devised an algorithm for reachability analysis and developed a model checker for verifying the system. Our future research will focus on a more effective method of verification, for example, model checking with CEGAR (Counterexample-guided abstraction refinement) and bounded model checking based on SMT (Satisfiability modulo theories) [18, 19].

References

[1] P. Garcia, K. Compton, M. Schulte, E. Blem and W. Fu. An Overview of Reconfigurable Hardware in Embedded Systems. *EURASIP J. Embedded Syst.*, 2006(1):1–19, 2002.

[2] J. W. Lockwood, J. Moscola, M. Kulig, D. Reddick and T. Brooks. Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware. *In Military and Aerospace Programmable Logic Device (MAPLD)*, E10, 2003.

[3] M. Motomura, T. Fujii, K. Furuta, K. Anjo, Y. Yabe, K. Togawa, J. Yamada, Y. Izawa and R. Sasaki. New Generation Microprocessor Architecture (2):Dynamically Reconfigurable Processor (DRP). *IPSJ Magazine*, 46(11):1259–1265, 2005.

[4] H. Amano, Y. Adachi, S. Tsutsumi and K. Ishikawa. A context dependent clock control mechanism for dynamically reconfigurable processors. *Technical Report of IEICE*, 104(589):13–16, 2005.

[5] R. Alur, C. Courcoubetis, T. A. Henzinger and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. *Lecture Notes in Computer Science*, 736:209–229, 1993.

[6] V. Varshavsky and V. Marakhovsky. GALA (Globally Asynchronous – Locally Arbitrary) Design. *Lecture Notes in Computer Science*, 2549:61–107, 2002.

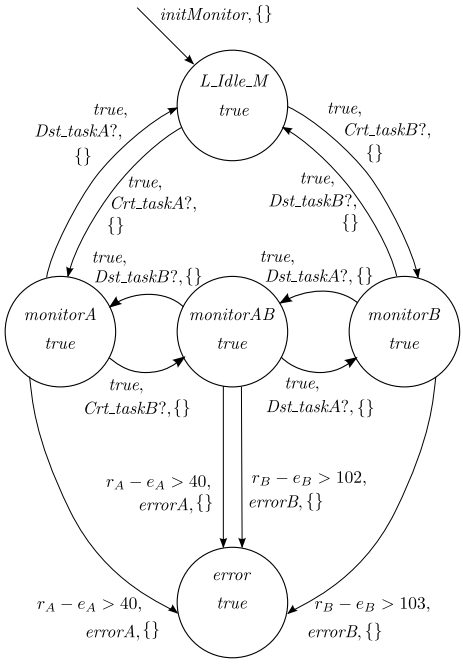

Figure 21: Monitor automaton for checking schedulability

[7] S. Minami, S. Takinai, S. Sekoguchi, Y. Nakai and S. Yamane. Modeling, Specification and Model checking of dynamically reconfigurable processors. *Computer Software*, 28(1):190–216, 2011.

[8] P. C. Attie and N. A. Lynch. Dynamic input/output automata, a formal model for dynamic systems. *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing (PODC '01)*, 2154:314–316, 2001.

[9] H. Yamada, Y. Nakai and S. Yamane. Proposal of Specification Language and Verification Experiment for Dynamically Reconfigurable System. *Journal of Information Processing Society of Japan, Programming*, 6(3):1–19, 2013.

[10] B. Boigelot and P. Godefroid. Symbolic Verification of Communication Protocols with Infinite StateSpaces using QDDs. *Form. Methods Syst. Des.*, 14(3):237–255, 1999.

[11] A. Bouajjani, J. Esparza and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. *Lecture Notes in Computer Science*, 1243:135–150, 1997.

[12] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Lecture Notes in Computer Science*,1256:560–570, 1997.

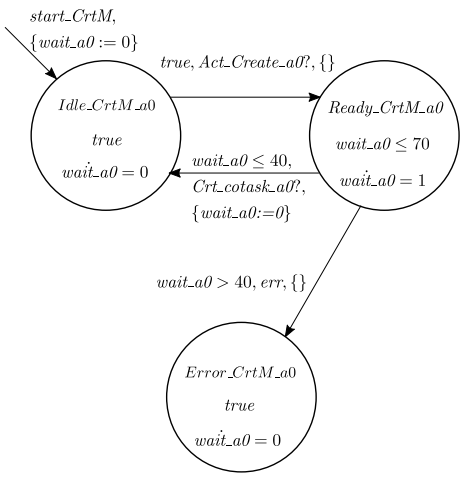

Figure 22: Monitor automaton for checking creation of co-task $a0$

[13] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[14] Y. Ono and S. Yamane. Computation of quantifier elimination of linear inequlities of first order predicate logic. *IEICE Technical Report. COMP, Computation*, 111(20): 55–59, 2011.

[15] R. Bagnara, P. M. Hill and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.*, 72(1–2): 3–21, 2008.

[16] B. Boigelot, P. Godefroid, B. Willems and P. Wolper. The Power of QDDs (Extended Abstract). *SAS*, 172-186, 1997.

[17] T. A. Henzinger, P. Ho and H. Wong-toi. HyTech : A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1: 460–463, 1997.

[18] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith. Counterexample-Guided Abstraction Refinement. *Proceedings of the 12th International Conference on Computer Aided Verification*, 1855:154–169, 2000.

[19] R. Nieuwenhuis, A. Oliveras and C. Tinelli. Abstract DPLL and abstract DPLL modulo theories. *In LPAR' 04, LNAI 3452*, 36–50, 2005.

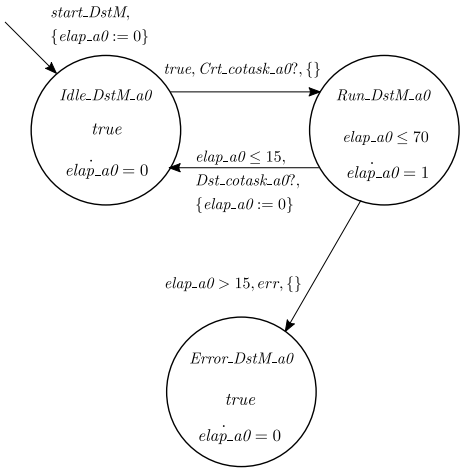

Figure 23: Monitor automaton for checking destruction of co-task $a0$

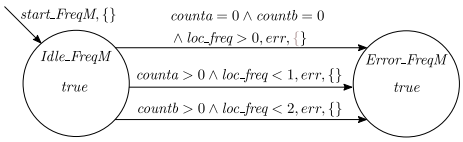

Figure 24: Monitor automaton for checking frequency management

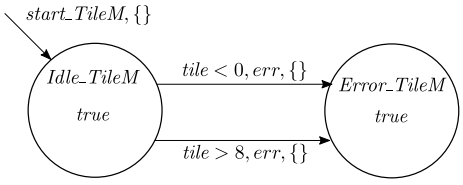

Figure 25: Monitor automaton for checking tile management

Model	Property	Satisfiability	Memory [MB]	Time [sec]	The number of states
Original:	Schedulability	yes	168	180	1220
	Creation of co-tasks	yes	92	315	1220
	Destruction of co-tasks	yes	154	233	1220
	Frequency Management	yes	173	265	1220
	Tile Management	yes	167	234	1220
Modified tasks:	Schedulability	no	105	10.2	91
	Creation of co-tasks	yes	117	145	771
	Destruction of co-tasks	yes	82	151	771
	Frequency Management	yes	197	115	771
	Tile Management	yes	135	107	771
Modified co-tasks:	Schedulability	yes	83	141	768
	Creation of co-tasks	yes	85	183	768
	Destruction of co-tasks	yes	86	191	768
	Frequency Management	yes	104	141	768
	Tile Management	yes	119	134	768

Table 5: Experimental results

A Source Code of the Model Checker

List 1: DLHA.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2  import java.io.BufferedWriter;
3  import java.io.File;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.io.PrintWriter;
7  import java.util.ArrayList;
8  import java.util.HashMap;
9  import java.util.List;
10 import java.util.Map;
11 import java.util.Set;
12 import java.util.HashSet;
13
14 /**
15  * DLHA(Dynamic Linear Hybrid Automaton)を表すクラスです。
16  *
17  */
18 public class DLHA {
19   private String      label;      // オートマトンの名前
20   private Set<Location> locationSet;  // ロケーションの集合
21   private Set<String>   clockSet;   // クロックの集合
22   private Set<String>   digitSet;   // 離散変数の集合
23   private Set<Transition> transitionSet;  // 遷移関係の集合
24   private Transition    tranInit;   // 初期遷移
25   private Set<Transition> tranEndSet;   // 破棄遷移の集合
26
27   /**
28    * 指定された名前をもつ空のオートマトンを作成します。
29    * @param label オートマトンの名前
30    */
31   public DLHA(String label) {
32     this.label      = label;
33     this.locationSet  = new HashSet<Location>();
34     this.clockSet   = new HashSet<String>();
35     this.digitSet   = new HashSet<String>();
36     this.transitionSet  = new HashSet<Transition>();
37     this.tranInit   = null;
38     this.tranEndSet   = new HashSet<Transition>();
39   }
40   /**
41    * 指定されたメンバ変数をもつオートマトンを作成します。
42    * @param label オートマトンの名前
43    * @param locationSet ロケーションの集合
44    * @param clockSet クロックの集合
45    * @param digitSet 離散変数の集合
46    * @param transitionSet 遷移関係の集合
47    * @param tranInit 初期遷移
48    * @param tranEndSet 破棄遷移の集合
49    */
50   public DLHA(String label, Set<Location> locationSet, Set<String> clockSet, Set<String> digitSet,
51       Set<Transition> transitionSet, Transition tranInit, Set<Transition> tranEndSet) {
52     this.label      = label;
53     this.locationSet  = new HashSet<Location>(locationSet);
54     this.clockSet   = new HashSet<String>(clockSet);
55     this.digitSet   = new HashSet<String>(digitSet);
56     this.transitionSet  = new HashSet<Transition>();
57     this.tranInit   = tranInit;
58     this.tranEndSet   = new HashSet<Transition>();
59     for (Transition t : transitionSet) {
60       this.addTransition(t);
61     }
62     for (Transition t : tranEndSet) {
63       this.addTranEnd(t);
64     }
65   }
66   /**
67    * オートマトンの名前を返します。
68    * @return オートマトンの名前
69    */
70   public String getLabel() {
71     return this.label;
72   }
73   /**
74    * 指定されたロケーションをロケーション集合に追加します。
75    * @param loc ロケーション
76    */
77   public void addLocation(Location loc) {
```

```
78      this.locationSet.add(loc);
79    }
80    /**
81     * クロックをクロックの集合に追加します。
82     * @param var クロック
83     */
84    public void addClock(String var) {
85      this.clockSet.add(var);
86    }
87    /**
88     * 離散変数を離散変数の集合に追加する。
89     * @param var 離散変数
90     */
91    public void addDigit(String var) {
92      this.digitSet.add(var);
93    }
94    /**
95     * 指定された遷移を遷移の集合に追加します。
96     * @param transition 遷移
97     */
98    public void addTransition(Transition transition) {
99      Location preLoc = transition.getPreLoc();
100     Location postLoc = transition.getPostLoc();
101     if(preLoc!=null && this.locationSet.contains(preLoc)
102         && postLoc!=null && this.locationSet.contains(postLoc)) {
103       this.transitionSet.add(transition);
104       preLoc.addTransition(transition);
105     }
106
107   }
108   /**
109    * 指定された遷移をオートマトンの初期遷移として保持します。
110    * @param transition 遷移
111    */
112   public void setTranInit(Transition transition) {
113     this.tranInit = transition;
114   }
115   /**
116    * 指定された遷移を破棄遷移の集合に追加します。
117    * @param transition 遷移
118    */
119   public void addTranEnd (Transition transition) {
120     Location preLoc = transition.getPreLoc();
121     if(preLoc!=null && this.locationSet.contains(preLoc)) {
122       this.tranEndSet.add(transition);
123       preLoc.addTransition(transition);
124     }
125   }
126   /**
127    * ロケーションの集合を返します。
128    * @return ロケーションの集合
129    */
130   public Set<Location> getLocationSet() {
131     return this.locationSet;
132   }
133   /**
134    * クロックの集合を返します。
135    * @return クロックの集合
136    */
137   public Set<String> getClockSet() {
138     return this.clockSet;
139   }
140   public Set<String> getVariables() {
141     Set<String> varSet = new HashSet<String>(clockSet);
142     varSet.addAll(digitSet);
143     return varSet;
144   }
145   /**
146    * 離散変数の集合を返します。
147    * @return 離散変数の集合
148    */
149   public Set<String> getDigitSet() {
150     return this.digitSet;
151   }
152   /**
153    * 初期遷移、破棄遷移を除く遷移の集合を返します。
154    * @return 遷移の集合
155    */
156   public Set<Transition> getTransitionSet() {
157     return this.transitionSet;
158   }
159   /**
160    * 初期遷移、破棄遷移を含むすべての遷移の集合を返します。
161    * @return 遷移の集合
```

```
162     */
163    public Set<Transition> getAllTransitions() {
164      Set<Transition> transitions =new HashSet<Transition>(transitionSet);
165      transitions.add(tranInit);
166      transitions.addAll(tranEndSet);
167      return transitions;
168    }
169    /**
170     * 初期遷移を返します。
171     * @return 初期遷移
172     */
173    public Transition getTranInit() {
174      return this.tranInit;
175    }
176    /**
177     * 破棄遷移を返します。
178     * @return 破棄遷移の集合
179     */
180    public Set<Transition> getTranEndSet() {
181      return this.tranEndSet;
182    }
183    /**
184     * 指定されたロケーションから出る遷移の集合を返します。
185     * @param loc ロケーション
186     * @return 遷移の集合
187     */
188    public Set<Transition> getOutgoingSet(Location loc) {
189      Set<Transition> tranSet = new HashSet<Transition>();
190
191      for(Transition t : transitionSet) {
192        if(loc.equals(t.getPreLoc())) {
193          tranSet.add(t);
194        }
195      }
196      return tranSet;
197    }
198    /**
199     * インスタンスが指定されたロケーションを含むとき true を返します。
200     * @param loc ロケーション
201     * @return 真理値
202     */
203    public boolean hasLocation(Location loc) {
204      if(this.locationSet.contains(loc)) {
205        return true;
206      }
207      return false;
208    }
209    /**
210     * graphviz用にファイルを書き出しますよー
211     * @param destinateFileName ファイル名だよー
212     */
213    public void outputFigure(String destinateFileName) {
214      File outputFile = new File(destinateFileName);
215      if(canWrite(outputFile)) {
216        PrintWriter printWriter;
217        try {
218          printWriter = new PrintWriter(new BufferedWriter(new FileWriter(outputFile)));
219          printWriter.println("digraph automaton {");
220          printWriter.println("graph [overlap=false, splines=true];");
221
222          int i = 0;
223          Map<String, String> locMap = new HashMap<String, String>();
224          for(Location location : locationSet) {
225            String nodeLabel = "n" + (i++);
226            locMap.put(location.getLabel(), nodeLabel);
227            printWriter.println("node [shape=circle, label=\"" + location.getLabel() + "\\n"
228                + location.getInvariant() + "\\n" + location.getFlow() + "\"] " + nodeLabel + ";");
229          }
230          String initialLabel = "n" + (i++);
231          printWriter.println("node [style=invis, label=\"\"] " + initialLabel + ";");
232          printWriter.println(initialLabel + " -> " + locMap.get(tranInit.getPostLoc().getLabel())
233              + " [labelfloat=false, label=\"" + tranInit.getAct() + "\\n" + tranInit.getUpdate() + "\"];");
234          for(Transition transition : transitionSet) {
235            printWriter.println(locMap.get(transition.getPreLoc().getLabel()) + " -> " + locMap.get(transition.
                   getPostLoc().getLabel())
236                + " [labelfloat=false, label=\"" + transition.getAct() + "\\n" + transition.getGrd() + "\\n" +
                       transition.getUpdate() + "\"];");
237          }
238          for(Transition transition : tranEndSet) {
239            String endLabel = "n" + (i++);
240            printWriter.println("node [style=invis, label=\"\"] " + endLabel + ";");
241            printWriter.println(locMap.get(transition.getPreLoc().getLabel()) + " -> " + endLabel
242                + " [labelfloat=false, label=\"" + transition.getAct() + "\\n" + transition.getGrd() + "\"];");
243          }
```

```
244
245           printWriter.println("}");
246           printWriter.close();
247         } catch (IOException e) {
248           System.err.println(e);
249         }
250       }
251     }
252     private static boolean canWrite(File outputFile) {
253       if (outputFile.exists()){
254         if (outputFile.isFile() && outputFile.canWrite()){
255           return true;
256         }
257         else {
258           return false;
259         }
260       }
261       return true;
262     }
263     /**
264      * DLHAのハッシュコードを返します。
265      * DLHAのハッシュコードは、ラベルのハッシュコード値によって定義されています。
266      * @return ハッシュコード値
267      */
268     public int hashCode() {
269       return this.label.hashCode();
270     }
271     /*
272     @Override
273     public String toString() {
274       return label;
275     } */
276     @Override
277     public String toString() {
278       return "DLHA [label=" + label + ", locationSet=" + locationSet
279           + ", clockSet=" + clockSet + ", digitSet=" + digitSet
280           + ", transitionSet=" + transitionSet + ", tranInit=" + tranInit
281           + ", tranEndSet=" + tranEndSet + "]";
282     }
283
284 }
```

List 2: Location.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2  import java.util.HashSet;
3  import java.util.Set;
4
5  import jp.ac.kanazawa_u.t.ec.csl.las.RationalNumber;
6
7  /**
8   * ロケーションを表すクラスです。
9   */
10 public class Location {
11   private String      label;      // ロケーションラベル
12   private Flow        flow;     // フロー条件
13   private Zone        invariant;    // 不変条件
14   private Set<Transition> transitionSet;  // このロケーションから出ている遷移の集合
15
16   /**
17    * 指定されたメンバ変数をもつ新規インスタンスを作成します。
18    * @param label ロケーションのラベル
19    */
20   public Location(String label) {
21     this.label      = label;
22     this.flow     = new Flow();
23     this.invariant    = new Zone(Zone.TRUE);
24     this.transitionSet  = new HashSet<Transition>();
25   }
26   /**
27    * 指定されたメンバ変数をもつ新規インスタンスを作成します。
28    * @param label ロケーションのラベル
29    * @param flow フロー条件
30    * @param invariant 不変条件
31    */
32   public Location(String label, Flow flow, Zone invariant) {
33     this.label      = label;
34     this.flow     = flow;
35     this.invariant    = invariant.clone();
36     this.transitionSet  = new HashSet<Transition>();
37   }
```

```
38    /**
39     * フロー条件を追加します。
40     * @param variable 変数
41     * @param value フロー
42     */
43    public void setFlow(String variable, Integer value) {
44      this.flow.setFlow(variable, value);
45    }
46    public void setFlow(String variable, RationalNumber value) {
47      this.flow.setFlow(variable, value);
48    }
49    /**
50     * フロー条件を追加します。
51     * @param variable 変数
52     * @param ref 参照する変数
53     */
54    public void setFlow(String variable, String ref) {
55      this.flow.setRef(variable, ref);
56    }
57    public void setInvariant(String strFormula) {
58      invariant = new Zone(strFormula);
59    }
60    /**
61     * 指定された遷移をインスタンスの遷移集合に追加します。
62     * @param t 遷移
63     */
64    protected void addTransition(Transition t) {
65      this.transitionSet.add(t);
66    }
67    /**
68     * 指定された遷移集合の全ての要素をインスタンスの遷移集合に追加します。
69     * @param tranSet 遷移の集合
70     */
71    protected void addAllTransitions(Set<Transition> tranSet) {
72      this.transitionSet.addAll(tranSet);
73    }
74    /**
75     * ロケーションのラベルを返します。
76     * @return ラベル
77     */
78    public String getLabel() {
79      return this.label;
80    }
81    /**
82     * フロー条件を返します。
83     * @return フロー条件
84     */
85    public Flow getFlow() {
86      return this.flow;
87    }
88    /**
89     * 不変条件を返します。
90     * @return 不変条件を表すゾーン
91     */
92    public Zone getInvariant() {
93      return this.invariant;
94    }
95    public Set<Transition> getTransitions() {
96      return this.transitionSet;
97    }
98    /**
99     * 指定されたロケーションとそのインスタンスが等しいとき true を返します。
100    * @param loc ロケーション
101    * @return 真理値
102    */
103   @Override
104   public boolean equals(Object obj) {
105     if (this == obj)
106       return true;
107     if (obj == null)
108       return false;
109     if (getClass() != obj.getClass())
110       return false;
111     Location other = (Location) obj;
112     if (label == null) {
113       if (other.label != null)
114         return false;
115     } else if (!label.equals(other.label))
116       return false;
117     return true;
118   }
119   /**
120    * このクラスの文字列表現を返します。
121    * @return このクラスの文字列表現
```

```
122     */
123    public String toString() {
124      return "(" + this.label + ": " + invariant + ", " + flow + ")";
125    }
126    /**
127     * ロケーションのハッシュコードを返します。
128     * ロケーションのハッシュコードは、ラベル(文字列)のハッシュコード値によって定義されます。
129     * @return ハッシュコード値
130     */
131    @Override
132    public int hashCode() {
133      final int prime = 31;
134      int result = 1;
135      result = prime * result + ((label == null) ? 0 : label.hashCode());
136      return result;
137    }
138
139
140  }
```

List 3: Flow.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2   import java.util.ArrayList;
3   import java.util.HashMap;
4   import java.util.List;
5   import java.util.Set;
6
7   import jp.ac.kanazawa_u.t.ec.csl.las.RationalNumber;
8   import jp.ac.kanazawa_u.t.ec.csl.las.TimeDeltaExecData;
9
10  /**
11   * フロー条件を表すクラスです。
12   *
13   */
14  public class Flow {
15    //private String flowCondition;
16      private HashMap<String, String>   ref;
17      private HashMap<String, RationalNumber>   flow;
18
19      /**
20       * このクラスの新規インスタンスを生成します。
21       */
22      public Flow() {
23        this.ref = new HashMap<String, String>();
24        this.flow = new HashMap<String, RationalNumber>();
25      }
26
27      /**
28       *  フロー条件を追加します。
29       * @param var 変数
30       * @param c 傾き
31       */
32      public void setFlow(String var, Integer c) {
33        this.flow.put(var, new RationalNumber(c));
34      }
35      public void setFlow(String var, RationalNumber c) {
36        this.flow.put(var, c);
37      }
38      /**
39       * 変数の参照を追加します。
40       * @param var 変数
41       * @param refVar 被参照変数
42       */
43      public void setRef(String var, String refVar) {
44        this.ref.put(var, refVar);
45      }
46      /**
47       * 複数のフロー条件を追加します。
48       * @param flowMap  フロー条件のマップ
49       */
50      public void setAllFlow(HashMap<String, RationalNumber> flowMap) {
51        this.flow.putAll(flowMap);
52      }
53      /**
54       * 複数の変数参照を追加します。
55       * @param confMap 参照のマップ
56       */
57      public void setAllRef(HashMap<String,String> refMap) {
58        this.ref.putAll(refMap);
59      }
```

```
60      /**
61       * 指定されたフロー条件と変数参照を全て追加します。
62       * @param f フロー条件
63       */
64      public void and(Flow f) {
65        this.setAllFlow(f.getAllFlow());
66        this.setAllRef(f.getAllRef());
67      }
68      /**
69       * 全てのフロー条件を返します。
70       * @return フロー条件のマップ
71       */
72      public HashMap<String, RationalNumber> getAllFlow() {
73        return this.flow;
74      }
75      /**
76       * 全ての変数参照を返します。
77       * @return 参照のマップ
78       */
79      public HashMap<String,String> getAllRef() {
80        return this.ref;
81      }
82      public boolean hasFlow(String variable) {
83        if(this.flow.keySet().contains(variable) || this.ref.keySet().contains(variable))
84          return true;
85        return false;
86      }
87      /**
88       * このインスタンスをTimeDeltaExecData型に変換して返します。
89       * @return TimeDeltaExecDataの配列
90       */
91      public TimeDeltaExecData[] toDelta() {
92        List<TimeDeltaExecData> deltaList = new ArrayList<TimeDeltaExecData>();
93        Set<String> flowKeySet  = this.flow.keySet();
94        Set<String> refKeySet = this.ref.keySet();
95
96        // フロー条件
97        if(flowKeySet != null) {
98          for(String key : flowKeySet) {
99            deltaList.add(new TimeDeltaExecData(key, this.flow.get(key)));
100         }
101       }
102       // 変数の参照
103       if(refKeySet != null) {
104         for(String key : refKeySet) {
105           if(this.flow.containsKey(this.ref.get(key))) {
106             deltaList.add(new TimeDeltaExecData(key, this.flow.get(this.ref.get(key))));
107           }
108         }
109       }
110       return (TimeDeltaExecData[]) deltaList.toArray(new TimeDeltaExecData[0]);
111     }
112     /**
113      * このクラスの文字列表現を返します。
114      */
115     public String toString() {
116       String    stringFlow  = new String("");
117       Set<String> flowKeySet  = this.flow.keySet();
118       Set<String> refKeySet = this.ref.keySet();
119
120       // フロー条件
121       if(flowKeySet != null) {
122         for(String key : flowKeySet) {
123           stringFlow = stringFlow + "d(" + key + ")=" + flow.get(key) + " ";
124         }
125       }
126       // 変数の参照
127       if(refKeySet != null) {
128         for(String key : refKeySet) {
129           stringFlow = stringFlow + "d(" + key + ")=" + ref.get(key) + " ";
130         }
131       }
132       return stringFlow;
133     }
134
135     public Flow clone() {
136       Flow ans = new Flow();
137       ans.setAllFlow(new HashMap<String, RationalNumber>(this.flow));
138       ans.setAllRef(new HashMap<String, String>(this.ref));
139
140       return ans;
141     }
142 }
```

List 4: Zone.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3  import java.util.Set;
4
5  import javax.naming.InitialContext;
6
7  import jp.ac.kanazawa_u.t.ec.csl.las.RationalNumber;
8  import jp.ac.kanazawa_u.t.ec.csl.las.TimeDeltaExecData;
9  import jp.ac.kanazawa_u.t.ec.csl.las.ZoneClass;
10 import jp.ac.kanazawa_u.t.ec.csl.las.parser.TokenException;
11
12
13 public class Zone {
14
15   public static final String TRUE = ZoneClass.TRUE;
16   public static final String FALSE = ZoneClass.FALSE;
17
18   ZoneClass zoneClass;
19
20   public Zone() {
21     zoneClass = null;
22     try {
23       zoneClass = new ZoneClass(TRUE);
24     } catch (TokenException e) {
25       // TODO Auto-generated catch block
26       e.printStackTrace();
27     }
28   }
29   public Zone(String string) {
30     zoneClass = null;
31     try {
32       zoneClass = new ZoneClass(string);
33     } catch (TokenException e) {
34       // TODO Auto-generated catch block
35       e.printStackTrace();
36     }
37   }
38   public boolean includes(Zone other) {
39     // TODO debug
40 //    System.out.print("Checking inclusive....\n    [" + zoneClass + "]\n    [" + other.zoneClass + "]");
41     boolean b = zoneClass.isContain(other.zoneClass);
42 //    System.out.println("[ok]");
43     return b;
44   }
45
46   public Zone timeSuccessor(TimeDeltaExecData[] deltaData) {
47     ZoneClass alterZone = zoneClass.clone();
48     alterZone.timeSuccessor(deltaData);
49
50     Zone resultZone = new Zone();
51     resultZone.zoneClass = alterZone;
52
53     return resultZone;
54   }
55
56   public Zone timePredeccessor(TimeDeltaExecData[] deltaData) {
57     ZoneClass alterZone = zoneClass.clone();
58     alterZone.timePre(deltaData);
59
60     Zone resultZone = new Zone();
61     resultZone.zoneClass = alterZone;
62
63     return resultZone;
64   }
65
66   public Zone and(Zone other) {
67     ZoneClass alterZone = zoneClass.clone();
68     alterZone.and(other.zoneClass);
69
70     Zone resultZone = new Zone();
71     resultZone.zoneClass = alterZone;
72
73     return resultZone;
74   }
75
76   public Zone or(Zone other) {
77     ZoneClass alterZone = zoneClass.clone();
78     alterZone.or(other.zoneClass);
79
80     Zone resultZone = new Zone();
81     resultZone.zoneClass = alterZone;
82
```

```
83      return resultZone;
84    }
85
86    public Zone updates(Update update) {
87      ZoneClass alterZone = zoneClass.clone();
88
89      Set<String> varSet = update.getVariables();
90      for(String var : varSet) {
91        // 場合分け
92        if(update.getOperation(var).getType()==OperationType.RESET) {
93          // リセットのとき
94          String varArray[] = new String[1];
95          varArray[0] = var;
96          alterZone.reset(varArray, new RationalNumber(update.getOperation(var).getValue()));
97        }
98        else {
99          // インクリメント or デクリメントのとき
100         try {
101           alterZone.update(var, var + update.getOperation(var).toString());
102         } catch (TokenException e) {
103           e.printStackTrace();
104         }
105       }
106     }
107     Zone resultZone = new Zone();
108     resultZone.zoneClass = alterZone;
109
110     return resultZone;
111   }
112
113   public Zone revUpdates(Update update) {
114     ZoneClass alterZone = zoneClass.clone();
115
116     Update reversedUpdate = update.reverse();
117
118     Set<String> varSet = reversedUpdate.getVariables();
119     for(String var : varSet) {
120       // 場合分け
121       if(update.getOperation(var).getType()==OperationType.RESET) {
122         // リセットのとき
123         String varArray[] = new String[1];
124         varArray[0] = var;
125         alterZone.and(new ZoneClass(var + "==" +update.getOperation(var).getValue()));
126         alterZone.free(varArray);
127       }
128       else {
129         // インクリメント or デクリメントのとき
130         try {
131           alterZone.update(var, var + update.getOperation(var).toString());
132         } catch (TokenException e) {
133           e.printStackTrace();
134         }
135       }
136     }
137     Zone resultZone = new Zone();
138     resultZone.zoneClass = alterZone;
139
140     return resultZone;
141   }
142
143   public static Zone updateToGuard(Update update) {
144     Zone alterZone = new Zone(Zone.TRUE);
145
146     Set<String> varSet = update.getVariables();
147     for(String var : varSet) {
148       if(update.getOperation(var).getType()==OperationType.RESET) {
149         alterZone = alterZone.and(new Zone(var + "==" +update.getOperation(var).getValue()));
150       }
151
152     }
153     return alterZone;
154   }
155
156   public Zone shift(Update update) {
157     ZoneClass alterZone = zoneClass.clone();
158     Set<String> varSet = update.getVariables();
159     for(String var : varSet) {
160
161       if(update.getOperation(var).getType()==OperationType.RESET) {
162
163       }
164       else {
165         // インクリメント or デクリメントのとき
166         try {
```

```
167             alterZone.update(var, var + update.getOperation(var).toString());
168           } catch (TokenException e) {
169             e.printStackTrace();
170           }
171         }
172       }
173       Zone resultZone = new Zone();
174       resultZone.zoneClass = alterZone;
175
176       return resultZone;
177     }
178
179     public Zone free(Set<String> varSet) {
180       ZoneClass alterZone = zoneClass.clone();
181       alterZone.free(varSet.toArray(new String[0]));
182
183       Zone resultZone = new Zone();
184       resultZone.zoneClass = alterZone;
185
186       return resultZone;
187     }
188     public Zone negation() {
189       ZoneClass alterZone = zoneClass.clone();
190       alterZone.negation();
191
192       Zone resultZone = new Zone();
193       resultZone.zoneClass = alterZone;
194
195       return resultZone;
196     }
197     @Override
198     public Zone clone() {
199       Zone clonedZone = new Zone();
200       clonedZone.zoneClass = this.zoneClass.clone();
201       return clonedZone;
202     }
203     @Override
204     public int hashCode() {
205       int result = 1;
206       return result;
207     }
208     @Override
209     public boolean equals(Object obj) {
210       if (this == obj)
211         return true;
212       if (obj == null)
213         return false;
214       if (getClass() != obj.getClass()) {
215         if (obj instanceof String) {
216           Zone otherZone = new Zone((String)obj);
217           if (!zoneClass.equals(otherZone.zoneClass))
218             return false;
219           return true;
220         }
221         else return false;
222       }
223
224       Zone other = (Zone) obj;
225       if (zoneClass == null) {
226         if (other.zoneClass != null)
227           return false;
228       } else if (!zoneClass.equals(other.zoneClass))
229         return false;
230       return true;
231     }
232     @Override
233     public String toString() {
234       return "" + zoneClass;
235     }
236
237     public String toConstSeparationString() {
238       return zoneClass.toConstSeparationString();
239     }
240   }
```

List 5: Guard.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3
4  /**
```

```
5   * ガード条件を表すクラスです。
6   */
7  public class Guard {
8    private boolean isAsap; // ASAPかどうか
9    private Zone zone; // ゾーン
10
11   /**
12    * このクラスの新規インスタンスを生成します。
13    */
14   public Guard() {
15     this.isAsap = false;
16     this.zone = new Zone("true");
17
18   }
19   /**
20    * 指定されたゾーンによって初期化された新規インスタンスを生成します。
21    * @param zone ゾーン
22    */
23   public Guard(Zone zone) {
24     this.isAsap = false;
25     this.zone = zone.clone();
26   }
27
28   private Guard(Builder builder) {
29     if(builder.zone != null) {
30       zone  = builder.zone.clone();
31     }
32     isAsap  = builder.isAsap;
33   }
34   /**
35    * ASAPであればtrueを返します。
36    * @return 真理値
37    */
38   public boolean isASAP() {
39     return this.isAsap;
40   }
41   /**
42    * ガード条件のゾーンを返します。
43    * @return ゾーン
44    */
45   public Zone getZone() {
46     return this.zone.clone();
47   }
48   /**
49    * ASAPかどうか設定します。
50    * @param b 真理値
51    */
52   public void setASAP(boolean b) {
53     this.isAsap = b;
54   }
55   /**
56    * ゾーンを設定します。
57    * @param zone ゾーン
58    */
59   public void setZone(Zone zone) {
60     this.zone = zone.clone();
61   }
62
63   /**
64    * インスタンスのディープコピーを返します。
65    */
66   public Guard clone() {
67     Guard grd = new Guard();
68     grd.zone = this.zone.clone();
69     grd.isAsap = this.isAsap;
70
71     return grd;
72   }
73
74   public static class Builder {
75     private Zone zone = null;
76     private boolean isAsap = false;
77
78     public Builder() {
79       zone = new Zone(Zone.TRUE);
80     }
81
82     public Builder zone(String strFormula) {
83       zone = new Zone(strFormula);
84       return this;
85     }
86     public Builder asap(boolean isAsap) {
87       this.isAsap = isAsap;
88       return this;
```

```
89      }
90      public Guard build() {
91        return new Guard(this);
92      }
93    }
94
95    @Override
96    public int hashCode() {
97      final int prime = 31;
98      int result = 1;
99      result = prime * result + (isAsap ? 1231 : 1237);
100     return result;
101   }
102   /**
103    * 指定されたガード条件と同一の場合 true を返します。
104    * @param obj ガード条件
105    * @return 真理値
106    */
107   @Override
108   public boolean equals(Object obj) {
109     if (this == obj)
110       return true;
111     if (obj == null)
112       return false;
113     if (getClass() != obj.getClass())
114       return false;
115     Guard other = (Guard) obj;
116     if (isAsap != other.isAsap)
117       return false;
118     if (zone == null) {
119       if (other.zone != null)
120         return false;
121     } else {
122       Zone zone1 = this.zone.clone();
123       Zone zone2 = other.zone.clone();
124       if (!zone1.equals(zone2))
125         return false;
126     }
127     return true;
128   }
129   @Override
130   public String toString() {
131     return (isAsap?"asap && ":"") + zone;
132   }
133
134
135
136 }
```

List 6: Action.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2   /**
3    * アクションを表すクラスです。
4    */
5   public class Action {
6     private String    label;  // ラベル
7     private ActionType  type; // アクションの種類
8
9     /**
10     * 指定されたメンバ変数をもつ新規インスタンスを生成します。
11     * @param label アクションのラベル
12     * @param type アクションの種類
13     */
14    public Action(String label, ActionType type) {
15      this.label  = label;
16      this.type = type;
17    }
18    /**
19     * アクションのラベルを返します。
20     * @return アクションのラベル
21     */
22    public String getLabel() {
23      return this.label;
24    }
25    /**
26     * アクションの種類を返します。
27     * @return アクションの種類
28     */
29    public ActionType getActionType() {
30      return this.type;
```

```
31    }
32
33    @Override
34    public String toString() {
35      String str = "";
36
37      switch (type) {
38      case ENQUEUE:
39        str += "Q!";
40        break;
41      case DEQUEUE:
42        str += "Q?";
43        break;
44      case CREATION:
45        str += "Crt_";
46        break;
47      case DESTRUCTION:
48        str += "Dst_";
49        break;
50      default:
51        break;
52      }
53
54      return str + label;
55    }
56    @Override
57    public int hashCode() {
58      final int prime = 31;
59      int result = 1;
60      result = prime * result + ((label == null) ? 0 : label.hashCode());
61      result = prime * result + ((type == null) ? 0 : type.hashCode());
62      return result;
63    }
64    /**
65     * 指定されたアクションが同一のアクションであれば true を返します。
66     * @param obj アクション
67     * @return 真理値
68     */
69    @Override
70    public boolean equals(Object obj) {
71      if (this == obj)
72        return true;
73      if (obj == null)
74        return false;
75      if (getClass() != obj.getClass())
76        return false;
77      Action other = (Action) obj;
78      if (label == null) {
79        if (other.label != null)
80          return false;
81      } else if (!label.equals(other.label))
82        return false;
83      if (type != other.type)
84        return false;
85      return true;
86    }
87  }
```

List 7: ActionType.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2   /**
3    * アクションの種類
4    */
5   public enum ActionType {
6     CREATION, // 生成アクション
7     DESTRUCTION,  // 破棄アクション
8     ENQUEUE,    // 待ち行列への追加
9     DEQUEUE,    // 待ち行列からの削除
10    TAU;        // その他の内部アクション
11  }
```

List 8: SynchronousAction.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   /**
4    * 同期遷移を表すクラスです。
```

```
5   *
6   */
7   public class SynchronousAction extends Action {
8     public SynchronousAction(String label, ActionType type) {
9       super(label, type);
10    }
11  }
```

List 9: OutputAction.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   public class OutputAction extends SynchronousAction {
4     public OutputAction(String label, ActionType type) {
5       super(label, type);
6     }
7
8     @Override
9     public String toString() {
10      return super.toString() + "!";
11    }
12  }
```

List 10: InputAction.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   public class InputAction extends SynchronousAction {
4     public InputAction(String label, ActionType type) {
5       super(label, type);
6     }
7
8     @Override
9     public String toString() {
10      return super.toString() + "?";
11    }
12  }
```

List 11: Update.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   import java.util.HashMap;
4   import java.util.HashSet;
5   import java.util.Map;
6   import java.util.Set;
7
8   /**
9    * 値の更新の式を表すクラスです。
10   */
11  public class Update {
12    private Map<String, Operation> updateMap;
13
14    /**
15     * 新しいインスタンスを作成します。
16     */
17    public Update() {
18      this.updateMap = new HashMap<String, Operation>();
19    }
20
21    private Update(Builder builder) {
22      this.updateMap = builder.updateMap;
23    }
24    /**
25     * 指定された変数に対する更新式(演算)を追加します。
26     * @param variable 変数
27     * @param operation 演算
28     */
29    public void addUpdate(String variable, Operation operation) {
30      this.updateMap.put(variable, operation);
31    }
32    /**
33     * 更新式が1つもないとき trueを返します。
34     * @return 更新式が1つもないとき true
```

```
35      */
36     public boolean isEmpty() {
37       return this.updateMap.isEmpty();
38     }
39     /**
40      * 指定された変数に対して更新式(演算)が設定されているとき true を返します。
41      * @param variable 変数
42      * @return 変数の更新式があるとき true
43      */
44     public boolean hasOperation(String variable) {
45       return this.updateMap.containsKey(variable);
46     }
47     /**
48      * 指定された変数に対する演算を返します。
49      * @param variable 変数
50      * @return 変数に対する演算
51      */
52     public Operation getOperation(String variable) {
53       return this.updateMap.get(variable);
54     }
55     /**
56      * 更新の対象となっている変数の集合を返します。
57      * @return 変数の集合
58      */
59     public Set<String> getVariables() {
60       return updateMap.keySet();
61     }
62     public Set<String> getResetVariables() {
63       Set<String> resetVar = new HashSet<String>();
64       Set<String> variables = updateMap.keySet();
65
66       for (String var : variables) {
67         Operation operation = updateMap.get(var);
68         if (operation.getType() == OperationType.RESET) {
69           resetVar.add(var);
70         }
71       }
72
73       return resetVar;
74     }
75     public static Update union(Update u1, Update u2) {
76       Update update = new Update();
77       for (String var : u1.getVariables()) {
78         update.updateMap.put(var, u1.updateMap.get(var));
79       }
80       for (String var : u2.getVariables()) {
81         update.updateMap.put(var, u2.updateMap.get(var));
82       }
83       return update;
84     }
85     /**
86      * 足し算もしくは引き算の形の式の符号を反転させます。
87      * @return 更新式
88      */
89     public Update reverse() {
90       Update revUpdate = new Update();
91       Set<String> variables = updateMap.keySet();
92
93       for (String var : variables) {
94         Operation operation = updateMap.get(var);
95         if (operation.getType() == OperationType.ADD) {
96           revUpdate.addUpdate(var, new Operation(OperationType.ADD, (-1)*operation.getValue()));
97         } else {
98           revUpdate.addUpdate(var, new Operation(operation.getType(), operation.getValue()));
99         }
100      }
101
102      return revUpdate;
103    }
104
105    public String toString() {
106      String str = "";
107      Set<String> variableSet = this.updateMap.keySet();
108      for(String var : variableSet) {
109        if(!("".equals(str))) {
110          str = str + ",";
111        }
112        Operation op = this.updateMap.get(var);
113        if(op.getType()==OperationType.RESET) {
114          str = str + var + ":=" + op;
115        }
116        else {
117          str = str + var + ":=" + var + op;
118        }
```

```
119       }
120       return str;
121     }
122
123     public static class Builder {
124       private Map<String, Operation> updateMap;
125
126       public Builder() {
127         updateMap = new HashMap<String, Operation>();
128       }
129       public Builder put(String str, Operation opr) {
130         updateMap.put(str, opr);
131         return this;
132       }
133       public Update build() {
134         return new Update(this);
135       }
136     }
137
138     @Override
139     public int hashCode() {
140       final int prime = 31;
141       int result = 1;
142       result = prime * result
143           + ((updateMap == null) ? 0 : updateMap.hashCode());
144       return result;
145     }
146
147     @Override
148     public boolean equals(Object obj) {
149       if (this == obj)
150         return true;
151       if (obj == null)
152         return false;
153       if (getClass() != obj.getClass())
154         return false;
155       Update other = (Update) obj;
156       if (updateMap == null) {
157         if (other.updateMap != null)
158           return false;
159       } else if (!updateMap.equals(other.updateMap))
160         return false;
161       return true;
162     }
163
164 }
```

List 12: Operation.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2  /**
3   * 更新式の演算を表すクラスです。
4   */
5  public class Operation {
6    private OperationType type; // 演算の種類
7    private int       value;  // 値
8
9    /**
10    * 指定された演算のインスタンスを新しく作成します。
11    * @param type 演算の種類
12    * @param value 値
13    */
14   public Operation(OperationType type, int value) {
15     this.type = type;
16     this.value  = value;
17   }
18   /**
19    * 演算の種類を返します。
20    * @return 演算の種類
21    */
22   public OperationType getType() {
23     return this.type;
24   }
25   /**
26    * 更新に用いられる値を返します。
27    * @return 更新に用いられる値
28    */
29   public int getValue() {
30     return this.value;
31   }
32   /**
```

```
33      * 演算の文字列表現を返します。
34      */
35     public String toString() {
36       String str = "";
37
38       if(type==OperationType.ADD) {
39         str += (this.value<0)?"":"+";
40         str += this.value;
41       }
42       else {
43         str += this.value;
44       }
45       return str;
46     }
47     /**
48      * 指定された演算とインスタンスが同一である場合にtrueを返します。
49      * @param op 演算
50      * @return 指定された演算と同一のときtrue
51      */
52     @Override
53     public boolean equals(Object obj) {
54       if (this == obj)
55         return true;
56       if (obj == null)
57         return false;
58       if (getClass() != obj.getClass())
59         return false;
60       Operation other = (Operation) obj;
61       if (type != other.type)
62         return false;
63       if (value != other.value)
64         return false;
65       return true;
66     }
67     /**
68      * インスタンスのハッシュコード値を返します。
69      * 演算のハッシュコードは演算に用いられる値によって定義されます。
70      * @return ハッシュコード値
71      */
72     @Override
73     public int hashCode() {
74       return value;
75     }
76
77   }
```

List 13: OperationType.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   public enum OperationType {
4     RESET,
5     ADD;
6   }
```

List 14: Transition.java

```
1   package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3   /**
4    * 遷移を表すクラスです。
5    */
6   public class Transition {
7     private TransitionType  type;   // 遷移の種類
8     private Location    preLoc;   // 遷移前のロケーション
9     private Location    postLoc;  // 遷移後のロケーション
10    private Guard     grd;    // ガード条件
11    private Action      act;    // アクション
12    private Update      update;   // 更新式
13
14    /**
15     * 遷移のインスタンスを新しく作成します。
16     * @param loc1 遷移前のロケーション
17     * @param grd ガード条件
18     * @param act アクション
19     * @param upd 更新式
20     * @param loc2 遷移後のロケーション
21     */
```

```
22    public Transition(Location loc1, Guard grd, Action act, Update upd, Location loc2) {
23      this.type   = TransitionType.TRANSITION;
24      this.preLoc   = loc1;
25      this.postLoc  = loc2;
26      this.grd    = grd;
27      this.act    = act;
28      this.update   = upd;
29    }
30    /**
31     * 初期遷移のインスタンスを新しく作成します。
32     * @param act アクション
33     * @param upd 更新式
34     * @param loc 初期ロケーション
35     */
36    public Transition(Action act, Update upd, Location loc) {
37      this.type   = TransitionType.TRANSITION_INIT;
38      this.preLoc   = null;
39      this.postLoc  = loc;
40      this.act    = act;
41      this.update   = upd;
42      this.grd    = new Guard();
43    }
44    /**
45     *  オートマトン破棄を示す遷移のインスタンスを新しく作成します。
46     * @param loc ロケーション
47     * @param grd ガード条件
48     * @param act アクション
49     */
50    public Transition(Location loc, Guard grd, Action act) {
51      this.type   = TransitionType.TRANSITION_END;
52      this.preLoc   = loc;
53      this.postLoc  = null;
54      this.grd    = grd;
55      this.act    = act;
56      this.update   = new Update();
57    }
58    /**
59     * 遷移の種類を返します。
60     * @return 遷移の種類
61     */
62    public TransitionType getType() {
63      return this.type;
64    }
65    /**
66     * 遷移前のロケーションを返します。
67     * @return ロケーション
68     */
69    public Location getPreLoc() {
70      return this.preLoc;
71    }
72    /**
73     * 遷移後のロケーションを返します。
74     * @return ロケーション
75     */
76    public Location getPostLoc() {
77      return this.postLoc;
78    }
79    /**
80     * ガード条件を返します。
81     * @return ガード条件
82     */
83    public Guard getGrd() {
84      return this.grd;
85    }
86    /**
87     * アクションを返します。
88     * @return アクション
89     */
90    public Action getAct() {
91      return this.act;
92    }
93    /**
94     * 更新式を返します。
95     * @return 更新式
96     */
97    public Update getUpdate() {
98      return this.update;
99    }
100   /**
101    * インスタンスの文字列表現を返します。
102    * @return インスタンスの文字列表現
103    */
104   public String toString() {
105     String str = "(";
```

```
106       switch (type) {
107       case TRANSITION:
108         if(preLoc != null)  str += preLoc.getLabel() + ", ";
109         if(grd != null)   str += grd.toString() + ", ";
110         if(update != null)  str += "{" + update.toString() + "}, ";
111         if(act != null)   str += act.toString() + ", ";
112         if(postLoc != null) str += postLoc.getLabel();
113         break;
114       case TRANSITION_INIT:
115         if(update != null)  str += "{" + update.toString() + "}, ";
116         if(act != null)   str += act.toString() + ", ";
117         if(postLoc != null) str += postLoc.getLabel();
118         break;
119       case TRANSITION_END:
120         if(preLoc != null)  str += preLoc.getLabel() + ", ";
121         if(grd != null)   str += grd.toString() + ", ";
122         if(act != null)   str += act.toString();
123         break;
124       default:
125         break;
126       }
127       return str + ")";
128     }
129     @Override
130     public int hashCode() {
131       final int prime = 31;
132       int result = 1;
133       result = prime * result + ((act == null) ? 0 : act.hashCode());
134       result = prime * result + ((postLoc == null) ? 0 : postLoc.hashCode());
135       result = prime * result + ((preLoc == null) ? 0 : preLoc.hashCode());
136       result = prime * result + ((type == null) ? 0 : type.hashCode());
137       result = prime * result + ((update == null) ? 0 : update.hashCode());
138       return result;
139     }
140     /**
141      * 指定された遷移とインスタンスが同一である場合に true を返します。
142      * @param obj 遷移
143      * @return 指定された遷移と同一であるとき true
144      */
145     @Override
146     public boolean equals(Object obj) {
147       if (this == obj)
148         return true;
149       if (obj == null)
150         return false;
151       if (getClass() != obj.getClass())
152         return false;
153       Transition other = (Transition) obj;
154       if (act == null) {
155         if (other.act != null)
156           return false;
157       } else if (!act.equals(other.act))
158         return false;
159       if (grd == null) {
160         if (other.grd != null)
161           return false;
162       } else if (!grd.equals(other.grd))
163         return false;
164       if (postLoc == null) {
165         if (other.postLoc != null)
166           return false;
167       } else if (!postLoc.equals(other.postLoc))
168         return false;
169       if (preLoc == null) {
170         if (other.preLoc != null)
171           return false;
172       } else if (!preLoc.equals(other.preLoc))
173         return false;
174       if (type != other.type)
175         return false;
176       if (update == null) {
177         if (other.update != null)
178           return false;
179       } else if (!update.equals(other.update))
180         return false;
181       return true;
182     }
183   }
```

List 15: TransitionType.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2  /**
3   * 遷移の種類
4   *
5   */
6  public enum TransitionType {
7    /**
8     * 普通の遷移
9     */
10   TRANSITION,
11   /**
12    * 初期遷移
13    */
14   TRANSITION_INIT,
15   /**
16    * 破棄遷移
17    */
18   TRANSITION_END;
19 }
```

List 16: State.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2
3  import java.util.HashSet;
4  import java.util.Set;
5
6
7  /**
8   * 状態を表すクラスです。
9   *
10  */
11 public class State {
12   private Set<Location> locationSet;    // Set of Locations
13   private Zone         zone;     // Zone
14   private QueueContent    queueContent; // Queue content
15
16   public State() {
17     this.locationSet    = new HashSet<Location>();
18     this.zone     = new Zone(Zone.TRUE);
19     this.queueContent = new QueueContent();
20   }
21   /**
22    * 指定されたロケーション集合とゾーンを持つ新しいインスタンスを生成します。
23    * @param locationSet ロケーション集合
24    * @param zone ゾーン
25    * @param queueContent キューの内容
26    */
27   public State(Set<Location> locationSet, Zone zone, QueueContent queueContent) {
28     this.locationSet    = locationSet;
29     this.zone       = zone;
30     this.queueContent   = queueContent;
31   }
32   /**
33    * 指定されたロケーションをインスタンスのロケーション集合に追加します。
34    * @param location ロケーション
35    */
36   public void addLocation(Location location) {
37     this.locationSet.add(location);
38   }
39   /**
40    * 指定されたロケーションの集合をインスタンスのロケーション集合に追加します。
41    * @param locationSet ロケーション集合
42    */
43   public void addAllLocations(Set<Location> locationSet) {
44     this.locationSet.addAll(locationSet);
45   }
46   /**
47    * 指定された文字列をキューの内容として設定します。
48    * @param queueContent
49    */
50   public void setQueueContent(QueueContent queueContent) {
51     this.queueContent = queueContent.clone();
52   }
53   /**
54    * ロケーション集合を返します。
55    * @return ロケーション集合
56    */
57   public Set<Location> getLocationSet() {
58     return this.locationSet;
59   }
```

```
60    /**
61     * ゾーンを返します。
62     * @return ゾーン
63     */
64    public Zone getZone() {
65      return this.zone;
66    }
67    /**
68     * キューの内容を返します。
69     * @return キューの内容
70     */
71    public QueueContent getQueueContent() {
72      return this.queueContent;
73    }
74    /**
75     * 指定されたロケーション集合を持つ場合にtrueを返します。
76     * @param locationSet ロケーション集合
77     * @return インスタンスのロケーション集合に指定されたロケーション集合が含まれる場合はtrue
78     */
79    public boolean hasLocations(Set<Location> locationSet) {
80      return this.locationSet.containsAll(locationSet);
81    }
82    public boolean includes(Object obj) {
83      if (this == obj)
84        return true;
85      if (obj == null)
86        return false;
87      if (getClass() != obj.getClass())
88        return false;
89      State other = (State) obj;
90      if (locationSet == null) {
91        if (other.locationSet != null)
92          return false;
93      } else if (!locationSet.equals(other.locationSet))
94        return false;
95      if (queueContent == null) {
96        if (other.queueContent != null)
97          return false;
98      } else if (!other.queueContent.isIncludedBy(queueContent))
99          return false;
100     if (zone == null) {
101       if (other.zone != null)
102         return false;
103     } else {
104       // TODO debug
105 //      System.out.print("check " + zone + " and " + other.zone);
106       if (!zone.includes(other.zone))
107 //        System.out.println("[ok]");
108         return false;
109     }
110 //    System.out.println("[ok]");
111     return true;
112   }
113   @Override
114   public String toString() {
115     String str = "<<(";
116
117     // ロケーション
118     for(Location loc : this.locationSet) {
119       if(!"<<(".equals(str)) {
120         str += ",";
121       }
122       str += loc.getLabel();
123     }
124
125     str = str + "), " + zone + ", " + queueContent + ">>";
126
127     return str;
128   }
129   @Override
130   public boolean equals(Object obj) {
131     if (this == obj)
132       return true;
133     if (obj == null)
134       return false;
135     if (getClass() != obj.getClass())
136       return false;
137     State other = (State) obj;
138     if (locationSet == null) {
139       if (other.locationSet != null)
140         return false;
141     } else if (!locationSet.equals(other.locationSet))
142       return false;
143     if (queueContent == null) {
```

```
144         if (other.queueContent != null)
145           return false;
146       } else if (!queueContent.equals(other.queueContent))
147         return false;
148       if (zone == null) {
149         if (other.zone != null)
150           return false;
151       } else {
152         // TODO debug
153 //        System.out.print("check " + zone + " and " + other.zone);
154         if (!zone.equals(other.zone)) {
155 //          System.out.println("[ok]");
156           return false;
157         }
158 //        System.out.println("[ok]");
159       }
160       return true;
161     }
162     @Override
163     public int hashCode() {
164       final int prime = 31;
165       int result = 1;
166       result = prime * result
167           + ((locationSet == null) ? 0 : locationSet.hashCode());
168       return result;
169     }
170     /* (non-Javadoc)
171      * @see java.lang.Object#clone()
172      */
173     @Override
174     protected State clone() {
175       State clonedState = new State();
176       clonedState.locationSet   = new HashSet<Location>(locationSet);
177       clonedState.zone      = zone.clone();
178       clonedState.queueContent  = queueContent.clone();
179       return clonedState;
180     }
181
182 }
```

List 17: QueueContent.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic;
2  import java.util.List;
3
4  import jp.ac.kanazawa_u.t.ec.csl.ryanase.qdd.*;
5
6  /**
7   * キューの内容を表すクラスです。
8   *
9   */
10 public class QueueContent {
11   private QDD qdd;
12   /**
13    * 新しいインスタンスを生成します。
14    * このとき、キューは空であるものとして初期化されます。
15    */
16   public QueueContent() {
17     qdd = new QDD();
18   }
19   /**
20    * 指定されたシンボルをキューに格納します。
21    * @param contentString シンボル
22    */
23   public void enqueue(String contentString) {
24     this.qdd = qdd.send(contentString).determinize();
25   }
26   /**
27    * 指定されたシンボルをキューから削除します。
28    * @param contentString シンボル
29    */
30   public void dequeue(String contentString) {
31     this.qdd = qdd.receive(contentString).determinize();
32   }
33   /**
34    * 指定された操作列についてメタ遷移を行います。
35    * @param operationList キュー操作のリスト
36    */
37   public void applyLoop(List<QDDOperation> operationList) {
38     System.out.println(" opList: " + operationList);
39     qdd = qdd.applyStar(operationList.toArray(new QDDOperation[0]));
```

```
40    }
41    /**
42     * 指定されたシンボルがキューの先頭にある場合にtrueを返します。
43     * @param contentString
44     * @return
45     */
46    public boolean isHead(String contentString) {
47      if(qdd == null) return false;
48      return qdd.isHead(contentString);
49    }
50    /**
51     * 指定されたキューの内容にインスタンスの内容が包含される場合にtrueを返します。
52     * @param obj 比較するキューの内容
53     * @return 指定されたキューの内容にインスタンスの内容が包含されるときtrue
54     */
55    public boolean isIncludedBy(Object obj) {
56      if (this == obj)
57        return true;
58      if (obj == null)
59        return false;
60      if (getClass() != obj.getClass())
61        return false;
62      QueueContent other = (QueueContent) obj;
63      if (qdd == null) {
64        if (other.qdd != null)
65          return false;
66      }
67
68      return qdd.isIncludedBy(other.qdd);
69    }
70    public boolean isEmpty() {
71      return this.qdd.isEmpty();
72    }
73    public QueueContent or(QueueContent other) {
74      QueueContent resContent = new QueueContent();
75      resContent.qdd = QDD.or(qdd, other.qdd);
76      return resContent;
77    }
78    @Override
79    public int hashCode() {
80      final int prime = 31;
81      int result = 1;
82      result = prime * result
83          + ((qdd == null) ? 0 : qdd.hashCode());
84      return result;
85    }
86    @Override
87    public boolean equals(Object obj) {
88      if (this == obj)
89        return true;
90      if (obj == null)
91        return false;
92      if (getClass() != obj.getClass())
93        return false;
94      QueueContent other = (QueueContent) obj;
95      if (qdd == null) {
96        if (other.qdd != null)
97          return false;
98      }
99
100     return qdd.equals(other.qdd);
101   }
102   @Override
103   public String toString() {
104     return "" + qdd;
105   }
106
107   public QueueContent clone() {
108     QueueContent clonedQueueContent = new QueueContent();
109     clonedQueueContent.qdd = this.qdd.clone();
110     return clonedQueueContent;
111   }
112
113   public String print() {
114     return this.qdd.print();
115   }
116 }
```

List 18: ReachabilityAnalyzer.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.reachability_analyzer;
```

```
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.Collections;
6  import java.util.HashMap;
7  import java.util.HashSet;
8  import java.util.LinkedList;
9  import java.util.List;
10 import java.util.Map;
11 import java.util.Queue;
12 import java.util.Set;
13 import java.util.concurrent.Callable;
14 import java.util.concurrent.ExecutionException;
15 import java.util.concurrent.ExecutorService;
16 import java.util.concurrent.Executors;
17 import java.util.concurrent.Future;
18 import java.util.concurrent.atomic.AtomicInteger;
19
20 import jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic.*;
21 import jp.ac.kanazawa_u.t.ec.csl.ryanase.qdd.*;
22
23
24 /**
25  * 到達可能性解析器のクラスです。
26  */
27 public class ReachabilityAnalyzer implements analyzerController{
28
29   private static final ReachabilityAnalyzer analyzer = new ReachabilityAnalyzer();
30
31     private Set<DLHA>   automatonSet;
32     private VisitStatesData visitStatesData;
33     private Map<Location, DLHA> locationMap;
34     private Set<String>   variablesSet;
35     private Map<String, Set<DLHA>> syncMap;
36     private Map<Set<Location>, Zone> invMap;
37     private Map<Set<Transition>, Zone> grdMap;
38
39     private AtomicInteger          numberOfStates;
40     private List<Set<Transition>>  counterExample;
41     private List<State>         counterExampleWithStates;
42     private SearchTree             searchTree;
43     private Queue<State>            stateQueue;
44
45     private ExecutorService executorService;
46     public static boolean isDebugMode = false;
47
48     private ReachabilityAnalyzer() {
49       this.automatonSet   = new HashSet<DLHA>();
50       this.visitStatesData    = null;
51       this.locationMap      = new HashMap<Location, DLHA>();
52       this.variablesSet   = new HashSet<String>();
53       this.syncMap        = new HashMap<String, Set<DLHA>>();
54       this.invMap       = new HashMap<Set<Location>, Zone>();
55       this.grdMap       = :new HashMap<Set<Transition>, Zone>();
56       this.numberOfStates   = new AtomicInteger(0);
57       this.counterExample   = null;
58       this.counterExampleWithStates   = null;
59       this.searchTree     = null;
60       this.stateQueue     = null;
61     }
62
63     public static ReachabilityAnalyzer getAnalyzer() {
64       return analyzer;
65     }
66
67     private void initialize(Set<DLHA> automatonSet) {
68       this.automatonSet       = new HashSet<DLHA>(automatonSet);
69       this.visitStatesData      = new VisitStatesData();
70       this.locationMap        = new HashMap<Location, DLHA>();
71       this.variablesSet     = new HashSet<String>();
72       this.syncMap          = new HashMap<String, Set<DLHA>>();
73       this.numberOfStates     = new AtomicInteger(0);
74       this.counterExample       = new LinkedList<Set<Transition>>();
75       this.counterExampleWithStates = new LinkedList<State>();
76       this.stateQueue         = new LinkedList<State>();
77
78       State initialState      = this.composeInitialState();
79
80       if(initialState != null && !initialState.getZone().equals(Zone.FALSE)) {
81         stateQueue.add(initialState);
82         searchTree     = new SearchTree(initialState);
83         visitStatesData.add(composeFlowSuccessor(initialState));
84         numberOfStates.incrementAndGet();
85       }
```

```
86
87          this.composeLocationMap();
88          for(DLHA automaton : this.automatonSet) variablesSet.addAll(automaton.getVariables());
89          this.composeSyncMap();
90        }
91
92        public boolean analyzeReachability(Set<DLHA> automatonSet, Set<Location> targetLocationSet) {
93          /**
94           * State-Space
95           */
96          initialize(automatonSet);
97
98          //  debug
99          System.out.println("
                ==========================================================================================\n
                *** Reachability Analysis *** \n");
100         System.out.println("initial state: " + stateQueue.peek());
101         System.out.print("target locations: ");
102         for(Location location : targetLocationSet) {
103           System.out.print("  " + location.getLabel());
104         }
105         System.out.println("\n");
106         System.out.print("Start analyzing...\n");
107
108         return compute(targetLocationSet);
109       }
110
111       private boolean compute(Set<Location> targetLocationSet) {
112         while(!stateQueue.isEmpty()) {
113           // state at the top of queue
114           State s = stateQueue.poll();
115
116           // debug
117           //System.out.print(".");
118           if(isDebugMode) System.out.println("state: " + s);
119           for(Location loc : targetLocationSet) {
120             if(s.getLocationSet().contains(loc)) {
121               counterExample = searchTree.getPathTo(s);
122               counterExampleWithStates = searchTree.getPathWithStatesTo(s);
123               return true;
124             }
125           }
126
127           Set<Location> locSet = s.getLocationSet();
128           List<Transition> asapTranList   = new ArrayList<Transition>();
129           List<Transition> wholeTranList  = new ArrayList<Transition>();
130           for(Location loc : locSet) {
131             Set<Transition> tranSet = loc.getTransitions();
132             for(Transition t : tranSet) {
133               if(t.getGrd().isASAP()) {
134                 asapTranList.add(t);
135               }
136               wholeTranList.add(t);
137             }
138           }
139
140           /**
141            * 遷移から次状態を求める
142            */
143           boolean    postStateFound = false;
144           State      jumpSuccessor  = null;
145           SearchEdge edge           = null;
146
147           for(Transition t : asapTranList) {
148             edge = new SearchEdge(s, t);
149             if(isDebugMode) System.out.println("    transition: " + t);
150
151             Action act = t.getAct();
152             if(!(act instanceof SynchronousAction)) {
153               jumpSuccessor = composeJumpSuccessor(s, t);
154
155               if(jumpSuccessor!=null && !jumpSuccessor.getZone().equals(Zone.FALSE)) {
156                 postStateFound = true;
157                 if(!visitStatesData.contains(jumpSuccessor)) {
158                   numberOfStates.incrementAndGet();
159
160                   State addedState = composeFlowSuccessor(jumpSuccessor);
161
162                   // debug
163                   if(isDebugMode) {
164                     System.out.println("    successor: " + jumpSuccessor);
165                     System.out.print("    Adding state...");
166                   }
167                   stateQueue.add(jumpSuccessor);
```

```
168                 visitStatesData.add(addedState);
169                 Set<Transition> transitions = new HashSet<Transition>();
170                 transitions.add(t);
171                 searchTree.grow(s, transitions, jumpSuccessor);
172                 if(isDebugMode) {
173                   System.out.println("[ok]");
174                 }
175               }
176             }
177           }
178           /**
179            * 同期の場合
180            */
181           else {
182             if(isDebugMode) System.out.print("    Computing SyncTranLists...");
183             List<List<Transition>> syncTranLists = getSyncTransitions(s, t);
184             if(isDebugMode) System.out.println("[ok]");
185             for(List<Transition> tranList : syncTranLists) {
186               jumpSuccessor = composeJumpSuccessor(s, tranList);
187
188               if(isDebugMode) System.out.println("      jumpSucc: " + jumpSuccessor);
189
190               if(jumpSuccessor!=null && !jumpSuccessor.getZone().equals(Zone.FALSE)) {
191                 postStateFound = true;
192                 if(!visitStatesData.contains(jumpSuccessor)) { // 未訪問なら
193                   numberOfStates.incrementAndGet();
194
195                   State addedState = composeFlowSuccessor(jumpSuccessor);
196
197                   // debug
198                   if(isDebugMode) {
199                     System.out.println("    successor: " + jumpSuccessor);
200                     System.out.print("    Adding state...");
201                   }
202                   stateQueue.add(jumpSuccessor);
203                   visitStatesData.add(addedState);
204                   Set<Transition> transitions = new HashSet<Transition>(tranList);
205                   searchTree.grow(s, transitions, jumpSuccessor);
206
207                   if(isDebugMode) System.out.println("[ok]");
208                 }
209               }
210             }
211           }
212         }
213         /**
214          * ASAPの遷移
215          */
216         if(!postStateFound) {
217           for(Transition t : wholeTranList) {
218             // debug
219             if(isDebugMode) System.out.println("    transition: " + t);
220
221             edge = new SearchEdge(s, t);
222             Action act = t.getAct();
223             /**
224              * 非同期の場合
225              */
226             if(!(act instanceof SynchronousAction)) {
227               State flowSuccessor = composeFlowSuccessor(s);
228
229               jumpSuccessor = composeJumpSuccessor(flowSuccessor, t);
230
231               if(isDebugMode) System.out.println("    jump succ:" + jumpSuccessor);
232
233               if(jumpSuccessor!=null && !jumpSuccessor.getZone().equals(Zone.FALSE)) {
234                 postStateFound = true;
235                 if(!visitStatesData.contains(jumpSuccessor)) { // 未訪問なら
236                   numberOfStates.incrementAndGet();
237
238                   State addedState = composeFlowSuccessor(jumpSuccessor);
239
240                   //  debug
241                   if(isDebugMode) System.out.print("    Adding state...");
242
243                   stateQueue.add(jumpSuccessor);
244                   visitStatesData.add(addedState);
245                   Set<Transition> transitions = new HashSet<Transition>();
246                   transitions.add(t);
247                   searchTree.grow(s, transitions, jumpSuccessor);
248
249                   if(isDebugMode) System.out.println("[ok]");
250                 }
251               }
```

```
252             }
253             /**
254              * 同期の場合
255              */
256             else {
257               /**
258                *  これと同期する遷移を全部とってくる
259                */
260               if(isDebugMode) System.out.print("    computing syncTranList...");
261
262               List<List<Transition>> syncTranLists = getSyncTransitions(s, t);
263
264               if(isDebugMode) System.out.println("[ok]");
265
266               /**
267                *  時間遷移してから離散遷移
268                */
269               for(List<Transition> tranList : syncTranLists) {
270
271                 // debug
272                 if(isDebugMode) {
273                   System.out.println("  tranList = " + tranList);
274                   System.out.println("  Computing jump successor now...");
275                 }
276                 State flowSuccessor = composeFlowSuccessor(s);
277                 if(isDebugMode) System.out.println("    flow successor = " + flowSuccessor);
278
279                 jumpSuccessor = composeJumpSuccessor(flowSuccessor, tranList);
280
281                 if(isDebugMode) System.out.println("    jump successor = " + jumpSuccessor);
282
283                 if(jumpSuccessor!=null && !jumpSuccessor.getZone().equals(Zone.FALSE)) {
284                   postStateFound = true;
285                   if(!visitStatesData.contains(jumpSuccessor)) { // 未訪問なら
286                     if(isDebugMode) System.out.println("  unvisited state.");
287
288                     numberOfStates.incrementAndGet();
289                     State addedState = composeFlowSuccessor(jumpSuccessor);
290                     stateQueue.add(jumpSuccessor);
291
292                     if(isDebugMode) System.out.println("      state to add:" + addedState);
293
294                     visitStatesData.add(addedState);
295                     Set<Transition> transitions = new HashSet<Transition>(tranList);
296
297                     if(isDebugMode) System.out.print("    Adding state...");
298
299                     searchTree.grow(s, transitions, jumpSuccessor);
300
301                     if(isDebugMode) System.out.println("[ok]");
302                   }
303                 }
304               }
305             }
306           }
307         }
308       }
309
310       return false;
311     }
312
313     public int getNumberOfStates() {
314       return this.numberOfStates.get();
315     }
316
317     public List<Set<Transition>> getPath() {
318       return this.counterExample;
319     }
320     public List<State> getPathWithStates() {
321       return this.counterExampleWithStates;
322     }
323     public SearchTree getSearchTree() {
324       return this.searchTree;
325     }
326
327     private State composeInitialState() {
328       State initialState;
329       Set<Location> locationSet = new HashSet<Location>();
330       String strFormula = "True"; // String of formula describing zone
331
332       for(DLHA automaton : automatonSet) {
333         Transition initialTransition = automaton.getTranInit();
334         if(initialTransition!=null) {
335           ActionType actionType = initialTransition.getAct().getActionType();
```

```
336             if(actionType != ActionType.CREATION) {
337               String initAssString = getInitialAssignments(automaton);
338               if(initAssString!=null && !"".equals(initAssString)) strFormula += " && " + initAssString;
339               locationSet.add(initialTransition.getPostLoc());
340             }
341           }
342         }
343
344         // ゾーンの生成
345         if("".equals(strFormula)) {
346           strFormula += Zone.TRUE;
347         }
348         Zone zone = new Zone(strFormula);
349
350
351         // 初期状態インスタンスの生成
352         initialState = new State(locationSet, zone, new QueueContent());
353
354         return initialState;
355       }
356
357       private String getInitialAssignments(DLHA automaton) {
358         String    strFormula      = "";
359         Transition  initialTransition = automaton.getTranInit();
360         Update    upd         = initialTransition.getUpdate();
361
362         for(String var : automaton.getVariables()) {
363           if(!("".equals(strFormula))) {
364             strFormula = strFormula + " && ";
365           }
366           if(upd.hasOperation(var)) {
367             if(upd.getOperation(var).getType()==OperationType.RESET) {
368               strFormula = strFormula + var + "==" + upd.getOperation(var);
369             }
370             else {
371               strFormula = strFormula + var + "==0";
372             }
373           }
374           else {
375             strFormula = strFormula + var + "==0";
376           }
377         }
378
379         return strFormula;
380       }
381
382       private State composeFlowSuccessor(State s) {
383         Set<Location> locSet = s.getLocationSet();
384         Zone zone = s.getZone().clone();
385
386         zone = zone.timeSuccessor(this.composeflows(locSet).toDelta());
387         zone = zone.and(this.composeInvariants(locSet));
388
389         return new State(locSet, zone, s.getQueueContent());
390       }
391
392       private State composeJumpSuccessor(State s, Transition t) {
393         ActionType type = t.getAct().getActionType();
394         Set<Location> postLocSet = new HashSet<Location>(s.getLocationSet());
395         if(t.getPreLoc()!=null) {
396           postLocSet.remove(t.getPreLoc());
397         }
398         if(t.getPostLoc()!=null) {
399           postLocSet.add(t.getPostLoc());
400         }
401
402         Zone postZone = s.getZone().and(t.getGrd().getZone());      // 遷移のガード条件とのAND
403         if(postZone.equals(Zone.FALSE)) return null;
404
405         postZone = postZone.updates(t.getUpdate());
406         if(postZone.equals(Zone.FALSE)) return null;
407
408         postZone = postZone.and(this.composeInvariants(postLocSet));
409
410         if(!postZone.equals(Zone.FALSE)) {
411           QDDOperation operation = null;
412           switch (type) {
413           case DEQUEUE:
414             operation = new QDDOperation(QDDOperationType.RECEIVE, t.getAct().getLabel());
415             break;
416           case ENQUEUE:
417             operation = new QDDOperation(QDDOperationType.SEND, t.getAct().getLabel());
418             break;
419           default:
```

```
420             break;
421           }
422
423           if(type == ActionType.DEQUEUE && !s.getQueueContent().isHead(t.getAct().getLabel())) {
424             return null;
425           }
426
427           QueueContent qc = searchTree.detectCycleAndGetQueueContent(s, operation, postLocSet);
428           if (qc != null && !qc.isEmpty()) {
429             return new State(postLocSet, postZone, qc);
430           }
431           else {
432             qc = s.getQueueContent().clone();
433
434             if(type != ActionType.DEQUEUE) {
435               if(type == ActionType.ENQUEUE) {
436                 qc.enqueue(t.getAct().getLabel());
437                 if(isDebugMode) System.out.println("      Equeued queue-content = " + qc);
438               }
439
440               return new State(postLocSet, postZone, qc);
441             } else {
442               qc.dequeue(t.getAct().getLabel());
443               return new State(postLocSet, postZone, qc);
444             }
445           }
446         }
447         return null;
448       }
449
450       private State composeJumpSuccessor(State s, List<Transition> syncTranList) {
451         Set<Location> postLocSet = new HashSet<Location>(s.getLocationSet());
452
453         Zone postZone = s.getZone().and(composeGuardCondition(syncTranList)); // 遷移のガード条件とのAND
454         if(postZone.equals(Zone.FALSE)) return null;
455
456         Update allUpd = new Update();
457         for(Transition t : syncTranList) {
458           if(t.getPreLoc()!=null)   postLocSet.remove(t.getPreLoc());
459           if(t.getPostLoc()!=null)  postLocSet.add(t.getPostLoc());
460
461           Update upd      = t.getUpdate();
462           Set<String> varSet  = upd.getVariables();
463           for(String var : varSet) {
464             allUpd.addUpdate(var, upd.getOperation(var));
465           }
466
467           switch (t.getType()) {
468           case TRANSITION_INIT:
469             DLHA   createdDlha  = getDLHA(t.getAct().getLabel());
470             if(createdDlha!=null) {
471               Zone  zone      = new Zone(getInitialAssignments(createdDlha));
472               postZone = postZone.and(zone);
473               this.variablesSet.addAll(createdDlha.getVariables());
474             }
475             else {
476               System.err.println("DLHA " + t.getAct().getLabel() + " is NOT found on a transition " + t + ".");
477             }
478             break;
479           case TRANSITION_END:
480             DLHA   destroyedDlha = getDLHA(t.getAct().getLabel());
481             if(destroyedDlha!=null) {
482               Set<String> freeVarSet = new HashSet<String>(destroyedDlha.getVariables());
483               postZone = postZone.free(freeVarSet);
484               this.variablesSet.removeAll(destroyedDlha.getVariables());
485             }
486             else {
487               System.err.println("DLHA " + t.getAct().getLabel() + " is NOT found.");
488             }
489             break;
490           default:
491             break;
492           }
493         }
494
495         postZone = postZone.updates(allUpd);
496         if(postZone.equals(Zone.FALSE)) return null;
497         postZone = postZone.and(composeInvariants(postLocSet));
498         if(!postZone.equals(Zone.FALSE)) {
499           QueueContent qc = searchTree.detectCycleAndGetQueueContent(s, null, postLocSet);
500           if (qc != null && !qc.isEmpty()) {
501             return new State(postLocSet, postZone, qc);
502           }
503
```

```
504           qc = s.getQueueContent().clone();
505           return new State(postLocSet, postZone, qc);
506         }
507         return null;
508       }
509
510       private Zone composeInvariants(Set<Location> locSet) {
511         if(locSet==null) {
512           return null;
513         }
514
515         if(invMap.containsKey(locSet)) {
516           return invMap.get(locSet);
517         }
518
519         Zone result = new Zone(Zone.TRUE);
520         for(Location loc : locSet) {
521           result = result.and(loc.getInvariant());
522         }
523         invMap.put(locSet, result);
524
525         return result;
526       }
527
528       private Flow composeflows(Set<Location> locSet) {
529         if(locSet==null) {
530           return null;
531         }
532         Flow result = new Flow();
533         for(Location loc : locSet) {
534           result.and(loc.getFlow());
535         }
536         for(String variable : variablesSet) {
537           if(!result.hasFlow(variable)) {
538             result.setFlow(variable, 0);
539           }
540         }
541         return result;
542       }
543       private Zone composeGuardCondition(Collection<Transition> transitions) {
544         if(transitions==null) {
545           return null;
546         }
547         Set<Transition> tranSet = new HashSet<Transition>(transitions);
548         if(grdMap.containsKey(tranSet)) {
549           return grdMap.get(tranSet);
550         }
551
552         Zone result = new Zone(Zone.TRUE);
553         for(Transition transition : transitions) {
554           result = result.and(transition.getGrd().getZone());
555         }
556         grdMap.put(new HashSet<Transition>(transitions), result);
557
558         return result;
559       }
560
561       private List<List<Transition>> getSyncTransitions(State s, Transition t) {
562         List<List<Transition>> inputLists = new ArrayList<List<Transition>>();
563         List<Transition> outputList = new ArrayList<Transition>();
564         List<List<Transition>> otherSyncList = new ArrayList<List<Transition>>();
565         ActionType type = t.getAct().getActionType();
566         String label = t.getAct().getLabel();
567
568         // 同期遷移かどうか
569         if(t.getAct() instanceof SynchronousAction) {
570           // 各ロケーションから出る遷移を探索
571           for(Location loc : s.getLocationSet()) {
572             List<Transition> inputList = new ArrayList<Transition>();
573             List<Transition> otherList = new ArrayList<Transition>();
574             for(Transition tran : loc.getTransitions()) {
575               Action act = tran.getAct();
576               if(label.equals(act.getLabel()) && type == act.getActionType()) {
577                 if(act instanceof OutputAction) {
578                   outputList.add(tran);
579                 }
580                 else if(act instanceof InputAction) {
581                   inputList.add(tran);
582                 }
583                 else {
584                   otherList.add(tran);
585                 }
586               }
587           }
```

```
588             // 空でなければ遷移のリストを追加
589             if(!inputList.isEmpty()) {
590               inputLists.add(inputList);
591             }
592             if(!otherList.isEmpty()) {
593               otherSyncList.add(otherList);
594             }
595           }
596           // 生成アクションを伴う遷移の場合は生成されるオートマトンの初期遷移も必要
597           if(type == ActionType.CREATION) {
598             DLHA dlha = this.getDLHA(label);
599             if(dlha!=null) {
600               if(!this.getRunningDLHAs(s.getLocationSet()).contains(dlha)) {
601                 List<Transition> list = new ArrayList<Transition>();
602                 list.add(dlha.getTranInit());
603                 inputLists.add(list);
604               }
605             }
606             else {
607               System.err.println("DLHA_NOT_FOUND_ERROR: " + label);
608             }
609           }
610         }
611
612         if(outputList.isEmpty() || inputLists.isEmpty()) {
613           if(otherSyncList.isEmpty()) {
614             return new ArrayList<List<Transition>>();
615           }
616           if(allows(label, otherSyncList)==false) {
617             return new ArrayList<List<Transition>>();
618           }
619           return combinate(otherSyncList);
620         }
621
622         // 遷移を1つにまとめる
623         inputLists.add(outputList);
624         inputLists.addAll(otherSyncList);
625         return combinate(inputLists);
626       }
627
628       private List<List<Transition>> combinate(List<List<Transition>> lists) {
629         List<List<Transition>> comb = new ArrayList<List<Transition>>();
630         if(lists != null) {
631           combinate(lists, 0, new LinkedList<Transition>(), comb);
632         }
633         return comb;
634       }
635
636       private void combinate(List<List<Transition>> lists, int index,
637           List<Transition> stack, List<List<Transition>> result) {
638         if(!(index < lists.size())) return;
639
640         List<Transition> list = lists.get(index);
641
642         for(Transition t : list) {
643           stack.add(t); // push
644           combinate(lists, index+1, stack, result);
645           if(index == lists.size()-1) {
646             result.add(new ArrayList<Transition>(stack));
647           }
648           if(stack.size() > 0) stack.remove(stack.size()-1);
649         }
650       }
651
652       private DLHA getDLHA(String label) {
653         for(DLHA automaton : automatonSet) {
654           if(label.equals(automaton.getLabel())) {
655             return automaton;
656           }
657         }
658         return null;
659       }
660
661       private Set<DLHA> getRunningDLHAs(Collection<Location> locations) {
662         Set<DLHA> automata = new HashSet<DLHA>();
663         for(Location location : locations) {
664           automata.add(locationMap.get(location));
665         }
666         return automata;
667       }
668
669       private boolean allows(String label, List<List<Transition>> tranLists) {
670         Set<DLHA> dlhas = new HashSet<DLHA>();
671         for(List<Transition> transitions : tranLists) {
```

```
672         for(Transition transition : transitions) {
673           Action action = transition.getAct();
674           if(label.equals(action.getLabel())) {
675             if(!(action instanceof InputAction) || action.getActionType() != ActionType.CREATION) {
676               dlhas.add(locationMap.get(transition.getPreLoc()));
677             } else {
678               dlhas.add(locationMap.get(transition.getPostLoc()));
679             }
680           }
681         }
682       }
683       return dlhas.containsAll(syncMap.get(label));
684     }
685 }
```

List 19: SearchEdge.java

```
1  package jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.reachability_analyzer;
2  import jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic.State;
3  import jp.ac.kanazawa_u.t.ec.csl.ryanase.dlha.basic.Transition;
4
5  /**
6   * 探索木を構成する辺のクラスです。
7   * @author ryo-yana
8   *
9   */
10 class SearchEdge {
11   private State s;
12   private Transition t;
13
14   public SearchEdge(State s, Transition t) {
15     this.s = s;
16     this.t = t;
17   }
18
19   public State getState() {
20     return this.s;
21   }
22   public Transition getTransition(){
23     return this.t;
24   }
25   public String toString() {
26     return s + ", " + t;
27   }
28
29   @Override
30   public int hashCode() {
31     final int prime = 31;
32     int result = 1;
33     result = prime * result + ((s == null) ? 0 : s.hashCode());
34     result = prime * result + ((t == null) ? 0 : t.hashCode());
35     return result;
36   }
37
38   @Override
39   public boolean equals(Object obj) {
40     if (this == obj)
41       return true;
42     if (obj == null)
43       return false;
44     if (getClass() != obj.getClass())
45       return false;
46     SearchEdge other = (SearchEdge) obj;
47     if (s == null) {
48       if (other.s != null)
49         return false;
50     } else if (!s.equals(other.s))
51       return false;
52     if (t == null) {
53       if (other.t != null)
54         return false;
55     } else if (!t.equals(other.t))
56       return false;
57     return true;
58   }
59 }
```