Exact acceleration of complex real-time model checking based on overlapping cycle

Guoqing Wang[1], Lei Zhuang[1], Yu Song[1], Mengyang He[1], Ding Ma[2], Ling Ma[1,3]

[1] School of Information Engineering, Zhengzhou University, Zhengzhou, Henan, China

[2] College of Information Science and Engineering, Henan University of Technology, Zhengzhou, Henan, China

[3] Digital Medical Image Technique Research Center, Zhengzhou University, Zhengzhou, Henan, China

Corresponding Author:

Lei Zhuang[1]

Science Avenue No. 100, Zhengzhou, Henan, 450001, China

Email address: ielzhuang@zzu.edu.cn

Abstract

When real-time systems are modeled as timed automata, different time scales may lead to substantial fragmentation of the symbolic state space. Exact acceleration solves the fragmentation problem without changing system reachability. The relatively mature technology of exact acceleration has been used with an appended cycle or a parking cycle, which can be applied to the calculation of a single acceleratable cycle model. Using these two technologies to develop a complex real-time model requires additional states and consumes a large amount of time cost, thereby influencing acceleration efficiency. In this paper, a complex real-time exact acceleration method based on an overlapping cycle is proposed, which is an application scenario extension of the parking-cycle technique. By comprehensively analyzing the accelerating impacts of multiple acceleratable cycles, it is only necessary to add a single overlapping period with a fixed length without relying on the windows of acceleratable cycles. Experimental results show that the proposed timed automaton model is simple and effectively decreases the time costs of exact acceleration. For the complex real-time system model, the method based on an overlapping cycle can accelerate the large scale and concurrent states which cannot be solved by the original exact acceleration theory.

Introduction

In real-time embedded systems, especially complex real-time control systems, discrete logic control and continuous time behavior depend on and influence each other. Take the Internet of things (IoT) gateway security system (Wang et al., 2018) as an example: Its control center generally has many different control modes to deal with diverse security risks, such as tampering, intrusion, and identity forging. Important system parameters (e.g., sensor status, monitoring

instructions, and terminal feedback information) change continuously over time. To meet specific time constraints or parameter values in the IoT gateway security system, the management mode must be adjusted over time. The change rules of important parameters also differ by mode, and the response time to various events should be modified accordingly. In this type of system (Lee et al., 2019), logic control describes the logical control transformation of the system through models with high abstraction levels, such as finite state machine and Petri net. Time behavior can be simulated by clock variables and clock zone transformation. Between the two layers, signals of the continuous layer and control modes of the discrete layer are correlated and transformed by certain interfaces and rules.

Typically, test and simulation technologies are the main means of guaranteeing software quality; however, they cover problems when using the operating system as the main measure, which cannot guarantee test completeness. These approaches are thus incapable of traversing all states in a real-time system, leading to covert problems in system operations (Wang, Pastore & Briand, 2019). In the field of security-related systems with zero tolerance for system error, using formal theory and technology for security authentication results in clear descriptions and avoids the complexity of safety verification. Formal description analysis and refinement have thus become a focus of recent research in related fields.

In real-time model checking, timed automata can model the temporal behavior of real-time systems (Pinisetty et al., 2017). Clocks describe the state transitions, and clock constraints serve as the theoretical basis for real-time system model checking (Han, Yang & Xing, 2015). This approach can easily realize automatic combination and transformation with other methods. The method is widely used in polling control systems, railway interlocking systems, and similar applications. Due to clock variables, control programs and external environments often use different time measures, which can cause the number of states to increase exponentially when a timed automaton is transformed into a zone automaton. The reachability analysis algorithm generates many state fragments (Iversen et al., 2000; Chen & Cui, 2016), resulting in a sharp increase in the state space and considerably prolonged detection time.

The acceleration technique is a reduction method used to solve the fragmentation problem following from time measurement differences. Dubout & Fleuret (2013) applied an acceleration technique to linear target detection and effectively improved the detection performance. Jeong et al. (2014) applied an implicit Markov model as an improved framework to accelerate the inference model. For distributed and parallel computing, a workstation and a multicore processor were used to accelerate state-space searching (Konur, Fisher & Schewe, 2013). Lin, Chen & Xu (2017) studied an acceleration model using a Bayesian classifier by analyzing the behavior of heterogeneous population trends; results indicated that acceleration in the reliability assessment improved the analytic accuracy. The model checking of linear temporal logic (LTL) model was studied by Barnat et al. (2010), which employed computed unified device architecture for acceleration. Two SAT problem solvers were used to validate online models and accelerate the processing of complex behaviors (Qanadilo, Samara & Zhao, 2013).

79   The reachability problem is the first to consider in timed automata, which determines whether a
80   path exists from its initial state to a target state. This problem can be solved by computing the
81   zones that apply the abstraction technique in practice. State-of-the-art abstraction methods
82   (Behrmann et al.,2006; Herbreteau, Srivathsan & Walukiewicz, 2016) produce an approximation
83   closer to the actual reachable clock valuation, which includes coarser abstractions. Exact
84   acceleration is an excellent means of abstraction to reduce required storage space and can alleviate
85   state-space explosion. For practical issues such as protocol validation (Zhang et al., 2013), IoT
86   system modeling (Li et al., 2013), and smart contract security verification in blockchain (Cruz,
87   Kaji & Yanai, 2018; Grishchenko, Maffei & Schneidewind, 2018), exact acceleration technology
88   is an efficient way of minimizing required storage space and time.
89   When Iversen et al. (2000) used UPPAAL to verify the LEGO robotic system, a fragmentation
90   problem was identified and briefly described, and some ideas for further research were suggested.
91   An approximation technique was applied to a real-time system model for security and connectivity
92   analysis, which avoided repetitive control (Möller, 2002). After that, a real-time property language
93    $L_{\forall S}$ was proposed to check the rejection state of reachability and reduce safety and boundary
94   liveness simultaneously (Aceto et al., 2003). The problems and methods in these publications have
95   promoted the concept of exact acceleration and inspired further research. Related studies on exact
96   acceleration in real-time model checking include Hendriks & Larsen (2002), Yin, Song & Zhuang
97   (2010), Yin, Zhuang & Wang (2011), Gou et al. (2014), Boudjadar et al. (2016), and Chadli et al.
98   (2018). In the following four examples, the window of the acceleratable cycle is $[a, \ b]$.

99   • Hendriks & Larsen (2002) introduced a method of syntax adjustment to a subset of timed
100   automata by adding an appended cycle whose length was $[a\ /(b - a)]$ times longer than that of
101   the acceleratable cycle. This method accelerates forward symbolic reachability analysis, which
102   solves the fragmentation problem and optimizes the verification of the LEGO robotic system.

103   • Yin, Song & Zhuang (2010) proposed a method to identify the acceleratable cycle in timed
104   automata by introducing topological sorting for a large state space of a timed automaton; by
105   simplifying the scale of timed automata, the method operated efficiently.

106   • An exact acceleration method based on a parking cycle was proposed (Yin, Zhuang & Wang,
107   2011), in which the entry boundary condition was determined by the size of the acceleratable
108   cycle's window (the condition is $z \geq a \times \frac{a}{b-a} + n_0$); the automaton model improved the speed
109   of exact acceleration and reduced the cost.

110   • By analyzing the main parameters of the acceleration process, Gou et al. (2014) proposed a
111   method for determining whether exact acceleration was required. This approach can be used to
112   avoid adding an appended cycle to reduce verification speed when the number of fragments is
113   small, or fragments do not satisfy certain conditions.

114   • Boudjadar et al. (2016) proposed a development method to improve the utilization rate of
115   resources by using model-checking technology. In the design and development stage, exact
116   acceleration technology was used to greatly improve the capability of symbolic model checking
117   in a processing scheduling system. For the scheduling problem of network physical systems,
118   Chadli et al. (2018) modeled advanced specifications and validation frameworks with the help

119   of exact acceleration technology, automatically transforming high-level specifications into
120   formal models. The above two research works mainly applied exact acceleration to model a
121   system resource scheduling problem but did not improve the original exact acceleration theory.
122   When modeling a complex real-time system (Wang et al., 2019), multiple acceleratable cycles
123   may overlap at the same location. If the appended cycle method is used for exact acceleration, then
124   the added locations multiply as the number of acceleratable cycles increases, resulting in
125   insufficient memory for model checking. If the parking cycle method is used for exact acceleration,
126   acceleratable-cycle stacking leads to non-uniformity in parking-cycle entry conditions; differences
127   in the windows of multiple acceleratable cycles can increase time consumption drastically. In this
128   paper, we propose an exact acceleration method for complex real-time model checking based on
129   an overlapping cycle, which is an application scenario extension of parking-cycle technique. A
130   single overlapping cycle is developed by comprehensively analyzing the accelerating effects of
131   multiple acceleratable cycles and analyzing acceleration differences among these cycles. The
132   overlapping cycle is simple to create and has a fixed length, eliminating the need to add multiple
133   locations for complex real-time models. The overlapping cycle adds much less state space than
134   appended cycles or parking cycles in model checking, substantially reducing the acceleration cost.
135   The proposed method can be effectively applied to modeling and verification of complex real-time
136   systems such as the IoT gateway security system. It can also alleviate additional consumption of
137   time and space caused by state-space explosion while maintaining the original nature of the system.
138   The remainder of this paper is organized as follows. Section II briefly introduces timed
139   automata, forward symbolic reachability analysis, and the theory of exact acceleration. The exact
140   acceleration method for complex real-time models based on an overlapping cycle is proposed in
141   Section III, which outlines the method of creating a single, fixed-length overlapping cycle. A timed
142   automaton with an overlapping cycle is shown to accelerate the originally timed automaton with
143   reachability. In Section IV, the acceleration effects of the appended cycle, parking cycle, and
144   overlapping cycle with a complex real-time model example are compared using experiments.
145   Finally, Section V provides a few ideas for future research.
146

Preliminaries

Timed Automata

149   This part is based on work by Alur & Dill (1994). To illustrate the real-time clock of timed
150   automata more clearly, we define a clock constraint set $T(C)$ contain all clock constraints. We
151   assume that the set of clock variables is C, and the definition of the set of clock constraints τ is as
152   follows:

$$\tau := c \sim n | \tau_1 \ \wedge \ \tau_2$$

154   where $c \in C$, $n \in \mathbb{N}$, and $\sim$ denotes one of the binary relationships $\{ <, \ \leq, \ =, \geq, > \}$. The clock
155   constraint set $T(C)$ is the set of all clock constraints τ.

156   A clock interpretation v is a mapping from C to $\mathbb{R}^+ \cup \{0\}$, where $\mathbb{R}^+$ represents the set of
157   positive real numbers. Note that v assigns each clock variable in the set of clock variables C. For

158   a set $X \subseteq C$, $X := 0$ indicates that X assigns 0 to each $c \in X$ (i.e., clock reset), whereas the clock
159   variables in set $C - X$ have no effects.

160   **Definition 1** (Timed automaton). A timed automaton is defined as a six-tuple (C, L, L_0, A, I, E),
161   where C is a set of clocks, L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, A is a
162   set of action events, I represents mapping that provides every location $l \in L$ with some clock
163   constraint in $T(C)$, and $E \subseteq L \times A \times T(C) \times 2^C \times L$ is a set of edges. An edge $(l, a, \tau, \lambda, l')$ denotes
164   a transition: when the clock constraint in location l satisfies τ, the system can complete action
165   event a, move from location l to location l', and allow clocks in λ to be reset.

166   Figure 1 shows an example of a timed automaton. The timed automaton M represents a plain
167   and abstract model of the control program and the external environment in a real-time system. If
168   the control program sends instructions to the control center in an IoT security system, the
169   environment will be decided by sensors and actuators. The cycle of locations L1, L2, and L3 model
170   the control program labeled the control cycle, consisting of three atomic instructions, whose clock
171   is x. The external environment is modeled by clock y, which is checked each time in L2. The clock
172   y also called global clock. The size of the threshold constant LARGE determines how slow the
173   environment is relative to the control program. If $y \geq$ LARGE, the control cycle may be exited.

174   The semantics of a timed automaton M is defined by a transition system $S(M)$ with Alur & Dill
175   (1994). A state of $S(M)$ is a pair (l,v), where l is a location of M and v indicates a clock
176   interpretation for C such that v satisfies $I(l)$. Regarding this transition system, the traces of a timed
177   automaton have been defined by Hendriks & Larsen (2002).

Forward Symbolic Reachability Analysis

179   The forward symbolic reachability analysis algorithm is a core of the real-time model-checking
180   tool UPPAAL (Behrmann, David & Larsen, 2004). The model-checking engine uses an on-the-fly
181   strategy to search forward from the initial location to determine whether a symbolic state is
182   reachable. For each symbolic state that has not yet been explored, it is necessary to calculate
183   subsequent states based on their clocks and actions and compare them to searched symbolic states.
184   If they have been seen in the past, they are discarded; otherwise, they are added to the list of
185   explored symbolic states.

186   The reachability property φ of a timed automaton M can be presented as the timed computation
187   tree logic (TCTL) formula $E <> (P)$, where P is a state property of M. We describe that M satisfies
188   φ, denoted by $M \vDash \varphi$, if a trace exists in the form of $((l_0,v_0),(l_1,v_1),...) \in Tr(M)$, where $(l_i,v_i) \vDash P$
189   for some $i \geq 0$.

190   To describe the process of forward symbolic reachability analysis, we take automaton M in Fig.
191   1 as an example. Table 1 shows the symbolic states that timed automaton M searches forward from
192   the initial location after one execution.

193   In Table 1, symbolic states 6 and 3 are both L2. However, the clock zones are not identical;
194   these states represent two different symbolic states to be further forward searched. Therefore, every
195   execution of the control cycle results in new symbolic states. Because the threshold LARGE is
196   usually larger and clock y is especially smaller, the timed automaton M must execute a certain

197   number of control cycles to increase clock y effectively when verifying the reachability of L4. The
198   number of executions depends on LARGE; if LARGE is large, then there are more executions. Due
199   to different clocks, when the model checking tools detect a symbolic model cycle by cycle, many
200   unnecessary clock fragments may appear in the state space, causing a forward symbolic fragment
201   problem. For example, if we only observe the symbolic states 3, 6, 9 and 12 of location L2, we can
202   find that each clock zone overlaps with the zone in front of it, which is called clock zone
203   continuous. At this time, because of the small-time measurement in the cycle, the overlapped clock
204   zone is divided into infinite segments, which leads to the fragmentation problem. Table 1 lists
205   results from the UPPAAL simulator.

206   **Exact Acceleration**

207   Hendriks & Larsen (2002) proposed the concept of exact acceleration, based on which, we provide
208   basic definitions for our study. The acceleratable cycle is a key concept in exact acceleration. An
209   acceleratable cycle can use only one clock in clock constraints (including invariants, guards, and
210   resets).

211   **Definition 2** (Acceleratable cycle). Let $M=(C,L,l_0,A,I,E)$ be a timed automaton, $E_c = (e_0,...,$
212    $e_{n-1}) \in E^n$, and $x \in C$. An acceleratable cycle is defined as a two-tuple (E_c,x) when the following
213   conditions are satisfied:

214   • E_c is a cycle;

215   • for all locations in E_c, $I(l)$ is either empty or in the form of $\{x \leq c\}$;

216   • if $(l,a,\tau,\lambda,l') \in E_c$, then τ is empty or in the form of $\{x \geq c\}$, and λ is empty or only
217   contains x; and

218   • x must be reset on all in-going edges to $src(e_0)$.

219   Clock x is the clock of the cycle, $I(l)$ is the location invariant, and τ is the edge guard. The
220   location $src(e_0)$ is the reset location whose out-going edge is e_0, which indicates the external
221   clock's checking position in the acceleratable cycle. If a specific location's in-going edge is e_i in
222   the cycle, then the out-going edge of this position is e_{i+1}, where $i \in [0,n-2]$.

223   The cycle in automaton M (Fig. 1), composed of locations L1, L2, and L3, is an acceleratable
224   cycle. The clock of the cycle is x, and the reset location is L2. The invariants and guards are in
225   accordance with the defined form of clock x, and the clock resets to zero at the only in-going edge
226   of L2.

227   **Definition 3** (Window of acceleratable cycle). Let an acceleratable cycle in the timed automaton
228   M be (E_c,x). The compression sequence of all traces is expressed as $T_r(E_c) =$
229    $\left((l_0,v_0),(l_0,v_0'),(l_1,v_1),...,(l_{n-1},v_{n-1}),(l_{n-1},v_{n-1}'),(l_n,v_n)\right)$, where v_i and v_i' indicate different
230   clock interpretations, $i \in [0,n]$, and $l_0=l_n=src(e_0)$. $((l_j,v_j'),(l_{j+1},v_{j+1}))$ depends on the edge e_j
231   and can be understood as an action event of e_j, $j \in [0,n-1]$. The window of the acceleratable
232   cycle (E_c,x) is defined as the interval $[a,b]$, $a,b \in \mathbb{N}$ when the following conditions are satisfied:

233   • the total delay of $T_r(E_c)$ is an element of $[a,b]$; and

234   •   for any real number $d \in [a,b]$, we adjust the delays under legal clock constraints in
235          $T_r(E_c)$ to ensure the total delay is d.

236   The meaning of the total delay in this definition is an increase in the clock of the cycle, which
237 can be simply defined as the increment of the external clock when the acceleratable cycle returns
238 to the reset location once from the initial location. The window is the minimal and maximal time
239 it may take to pass through a cycle. According to this definition, the window of the acceleratable
240 cycle shown in Fig. 1 can be calculated as [3, 7].

241   **Definition 4** (Accelerated automaton based on appended cycle). Let $M = (C,L,l_0,A,I,E)$ be a
242 timed automaton, and let $Cycle = (E_c,x)$ be an acceleratable cycle of M, where $L=\{l_0,l_1,...,l_m\}$, $E_c =$
243 $(e_0,e_1,...,e_{n-1})$, $e_i=(l_i,a_i,\tau_i,\lambda_i,l_{i+1})$. Acceleration of M based on the appended cycle is a new
244 automaton $Acc_a(M,Cycle)$ defined as (C,L',l_0,A,I',E'), where

245   • $L' = L \cup \{l'_1,l'_2,...,l'_{n-1}\} \cup \{l'_0\} \cup \{l''_1,l''_2,...,l''_{n-1}\}$
246   • $I'(l_i) = I(l_i), 0 \leq i \leq m$
247   • $I'(l'_i) = I(l_i), 1 \leq i \leq n-1$
248   • $I'(l'_0) = \emptyset$
249   • $I'(l''_i) = I(l_i), 1 \leq i \leq n-1$
250   • $E' = E \cup \{(l_0,a_0,\tau_0,\lambda_0,l'_1),(l'_{n-1},a_{n-1},\tau_{n-1},\lambda_{n-1},l'_0)\} \cup$
251         $\{(l'_0,a_0,\tau_0,\lambda_0,l''_1),(l''_{n-1},a_{n-1},\tau_{n-1},\lambda_{n-1},l_0)\} \cup$
252         $\{(l'_i,a_i,\tau_i,\lambda_i,l'_{i+1}),(l''_i,a_i,\tau_i,\lambda_i,l''_{i+1})|1 \leq i \leq n-1\}$
253   •   in particular, $E' = E \cup \{(l_0,a_0,\tau_0,\lambda_0,l'_0),(l'_0,a_0,\tau_0,\lambda_0,l_0)\}$ when n = 1.

254   **Theorem 1**. Let $M = (C,L,l_0,A,I,E)$ be a timed automaton, and let $Cycle = (E_c,x)$ be an
255 acceleratable cycle of M with a window $[a,b]$. If φ is the reachability property of M, then

$$3a \leq 2b \Rightarrow (M \vDash \varphi \Leftrightarrow Acc_a(M,Cycle) \vDash \varphi)$$

257   Theorem 1 has been proved in Hendriks & Larsen (2002). In Definition 4, the appended cycle
258 is obtained by expanding the acceleratable cycle twice. If the timed automaton M is added to the
259 appended cycle by expanding the acceleratable cycle i times, then the precondition in Theorem 1
260 can be generalized to $(i+1)a \leq ib$.

261   **Definition 5** (Accelerated automaton based on parking cycle). Let $M = (C,L,l_0,A,I,E)$ be a timed
262 automaton, and let $Cycle = (E_c,x)$ be an acceleratable cycle of M with a window of $[a,b]$, where L
263 $=\{l_0,l_1,...,l_m\}$, $E_c=(e_0,e_1,...,e_{n-1})$, $e_i = (l_i,a_i,\tau_i,\lambda_i,l_{i+1})$. The global clock is y, and the maximum
264 value of y before entering the acceleratable cycle is n_0. The acceleration of M based on the parking
265 cycle is a new automaton $Acc_p(M,Cycle)$ defined as (C,L',l_0,A,I',E'), where

266   • $L' = L \cup \{l'_0\}$
267   • $I'(l_i) = I(l_i), 0 \leq i \leq m$
268   • $I'(l'_0) = \emptyset$

269   • $E^{'} = E \cup \{(l_0, a_0, \tau', \emptyset, l_0^{'}), (l_0^{'}, a_{n-1}, \emptyset, \lambda_{n-1}, l_0)\}$, τ' is $y \geq a \times \left\lceil \frac{a}{b-a} \right\rceil + n_0$

270   Definition 5 has been given in Yin, Zhuang & Wang (2011). The accelerated automaton Acc_a

271     $(M, Cycle)$ equals the timed automaton M with an appended cycle composed of locations $l_0, l_1^{'}, l_2^{'}, ...,$

272     $l_{n-1}^{'}, l_0^{''}, l_1^{''}, l_2^{''}, ..., l_{n-1}^{''}$. Accelerated automaton $Acc_p(M, Cycle)$ equals the timed automaton M with a

273   parking cycle whose edge guard y controls the acceleration timing. Only when the acceleratable

274   cycle has been executed at least $\lceil a/(b-a) \rceil$ times is the automaton permitted to enter the parking

275   cycle. Figures 2 and 3 display the acceleration of M (Fig. 1) based on the appended cycle and

276   parking cycle, respectively. They are labeled the accelerated automata M_a and M_p. Because the

277   window of the acceleratable cycle is [3, 7], the edge guard of the parking cycle is $y \geq 3$ in M_p.

278   **Theorem 2**. Let $M = (C, L, l_0, A, I, E)$ be a timed automaton, and let $Cycle = (E_c, x)$ be an

279   acceleratable cycle of M with a window of $[a, b]$. If φ is a reachability property of M, then

280     $$a < b \Rightarrow (M \vDash \varphi \Leftrightarrow Acc_p(M, Cycle) \vDash \varphi)$$

281   Yin, Zhuang & Wang (2011) gives this theorem form forward symbolic reachability analysis,

282   and this is the previous achievement of our working group. We will give its another proof in the

283   view of zone later.

284

Exact Acceleration of Complex Real-time System Model Based on Overlapping Cycle

287   The appended cycle and parking cycle technologies in exact acceleration apply to a real-time

288   system model with a single acceleratable cycle. For a complex real-time system model (as shown

289   in Fig. 4), using these two technologies for exact acceleration requires additional states and

290   consumes a large amount of time cost, which influences the acceleration effect.

291   Figure 4 presents an example of the IoT gateway security system (in Wang et al., 2018). The

292   timed automaton M' models a wireless sensor network including a reactive program and external

293   environment. The run-time behavior control several sensors, which can be transformed into clock

294   constrains in UPPAAL. Every location in the cycle represents a sensor model in the IoT system.

295   The cycle's clock is x, and clock y controls the execution time. The larger the constant LARGE is,

296   the more slowly the timed automaton M' runs. Using the algorithm described by Yin, Song &

297   Zhuang (2010) to identify the acceleratable cycle in M', we obtain four acceleratable cycles whose

298   reset locations are all L1 and share the clock of cycle x. For this complex real-time model, we

299   propose a method based on the overlapping cycle for exact acceleration.

300   **Theorem 3**. Let $M = (C, L, l_0, A, I, E)$ be a timed automaton, and let $Cycle = (E_c, x)$ be an

301   acceleratable cycle of M with a window of $[a, b]$. If $a < b$, there is a positive integer n in the

302   forward symbolic reachability analysis, which leads the reset location to obtain a continuous clock

303   zone after executing the $Cycle$ n times.

304   *Proof.* According to the forward symbolic reachability analysis, the problem of fragments in the

305   acceleratable cycle will inevitably lead to the overlap of clock zones, that is, the appearance of

306   continuous clock zone. If $a < b$, according to the definitions about exact acceleration, the

307 continuous clock zone will be got after several executions of the acceleratable cycle, and the point
308 of the proof is to determine the number of executions.
309   So, without loss of generality, we might assume that the execution number is a positive integer
310 n. Let n be the rounds of $Cycle$ execution, and let the interval $[c,d]$ be the clock zone at the reset
311 location before execution of the $Cycle$. At the reset location, the clock zone is continuous from the
312 $(n+1)$th time onward; therefore, the clock zones obtained in the $(n+1)$th time and the nth time
313 have an intersection that is

314 $$(n+1)a + c \leq nb + d \Rightarrow n \geq (a+c-d)/(b-a)$$

315   Because $b > a$, $d \geq c$, there must be an integer $n \geq a/(b-a)$. So, the number of executions
316 should be at least $\lceil a/(b-a) \rceil$. When the $Cycle$ is executed $\lceil a/(b-a) \rceil$ (that is n) times, the reset
317 location obtains a continuous clock zone, thereby completing the proof.
318
319   **Corollary 1**. If the timed automaton M has an acceleratable cycle with a window of $[a,b]$, $a < b$,
320 then the reset location will obtain a continuous clock zone after executing the $Cycle$ at least
321 $a/(b-a)$ times during forward symbolic reachability analysis.
322   *Proof.* According to Theorem 3, we easily know the reset location obtains a continuous clock
323 zone when the $Cycle$ is executed n times. For the integer n, we can calculate that $n \geq a/(b-a)$.
324 The proof is completed.
325
326   **Corollary 2**. Let the global clock be y, and let the maximum value of y before entering the
327 acceleratable cycle be n_0. If the timed automaton M has an acceleratable cycle with a window of
328 $[a,b]$, $a < b$, then the reset location will obtain a continuous clock zone when the following
329 condition is satisfied during forward symbolic reachability analysis:

330 $$y - n_0 \geq a \times a/(b-a)$$

331   *Proof.* According to Corollary 1, the reset location will obtain a continuous clock zone after
332 executing the acceleratable cycle at least $a/(b-a)$ times. Because the window of the acceleratable
333 cycle is $[a,b]$, the increment of y by executing the acceleratable cycle $a/(b-a)$ times is denoted
334 as $\Delta y \in [a \times a/(b-a), b \times a/(b-a)]$, where $\Delta y = y - n_0$. Therefore, when $y - n_0 \geq a \times a/(b-$
335 $a)$, the reset location will obtain a continuous clock zone. The proof is completed.
336
337   **Corollary 3**. Let the global clock be y, and let the maximum value of y before entering the
338 acceleratable cycle be n_0. If the timed automaton M has an acceleratable cycle with a window of
339 $[a,b]$, $a < b$, then every location in the acceleratable cycle will obtain a continuous clock zone
340 when $y - n_0 \geq a \times a/(b-a)$.
341   *Proof.* Because clock y is not the cycle clock, any invariant or guard in the acceleratable cycle
342 will not contain clock y according to Definition 2. Based on the theory of timed automata, clock
343 y exhibits monotonous growth when the acceleratable cycle is executed. Thus, when $y - n_0 \geq$
344 $a \times a/(b-a)$, per Corollary 2, the reset location begins to obtain a continuous clock zone,

345   indicating that every location in the acceleratable cycle is reachable. At this point, a constant clock
346   zone will also be received by any location in the acceleratable cycle, and the proof is completed.
347

348   Next, we will give the new proof of Theorem 2.
349   *Proof.* Sufficient Condition. The known condition is $a < b$. Because $Acc_p(M,Cycle)$ is obtained
350   by adding a parking cycle to M, the timed automaton M can clearly reach any reachable state in
351   original model. The accelerated automaton $Acc_p(M,Cycle)$ can also reach states by executing the
352   same time trace; that is, the state transition system $S(M)$ associated with M is included in the state
353   transition system $S(Acc_p(M,Cycle))$ associated with $Acc_p(M,Cycle)$.

354   Necessary Condition. The known condition is $a < b$. Let the global clock be y. When $y < a \times a$
355    $/(b-a) + n_0$, the accelerated automaton $Acc_p(M,Cycle)$ does not execute the parking cycle, and
356   reachable states in $Acc_p(M,Cycle)$ are also reachable in the timed automaton M. When $y \geq a \times a$
357    $/(b-a) + n_0$ (according to Corollary 3), every location in the acceleratable cycle of M will obtain
358   a continuous clock zone; that is, after $y \geq a \times a/(b-a) + n_0$ at any time, M can reach any
359   location in the acceleratable cycle and $Acc_p(M,Cycle)$ executes the parking cycle, satisfies the
360   edge guards, and returns to the reset location of any state, which guarantees that M is always
361   reachable.
362   In summary, when $a < b$, the accelerated automaton $Acc_p(M,Cycle)$ does not change the
363   reachability property φ of the timed automaton M, and the proof is completed.
364

365   According to our theorems and corollaries, we can prove that the exact acceleration method
366   based on the parking cycle is more concise and effective than that based on the appended cycle.
367   On one hand, there is no location invariant in the parking cycle to ensure the clock can stay in this
368   location for acceleration; on the other hand, the parking cycle contains an edge guard to ensure
369   that any location of the acceleratable cycle obtains a continuous clock zone, which provides
370   reachability. In this way, the parking cycle accelerates the search for the symbolic state by
371   controlling acceleration timing and ensures reachability of the timed automaton to realize exact
372   acceleration. The exact acceleration method for the complex real-time model based on an
373   overlapping cycle is an improved method based on the parking cycle. It attempts to extend the
374   application field of exact acceleration technology to complex real-time model checking to improve
375   efficiency and alleviate state explosion.
376   **Theorem 4**. Let $M = (C,L,l_0,A,I,E)$ be a timed automaton with several acceleratable cycles. Let
377    $Cycle_i=(E_{c_i},x)$ be the ith acceleratable cycle of M with a window of $[a_i,b_i]$, where i is a non-zero
378   natural number. All acceleratable cycles affect the cycle of clock x, and their reset locations are
379   uniform in $l_{reset} \in L$. There is a single acceleratable cycle whose effect is the most effective in
380   obtaining a continuous clock zone than multiple acceleratable cycles.

381   *Proof.* Let $n_j = \left\lceil \frac{a_j}{b_j - a_j} \right\rceil \times a_j$, $n_k = \left\lceil \frac{a_k}{b_k - a_k} \right\rceil \times a_k$, where $1 \leq j,k \leq i$, and j, k are non-zero natural

382   numbers. Then, n_j and n_k represent the edge guard of the jth and kth acceleratable cycle,

383   respectively, when adding a parking cycle. If these two acceleratable cycles are executed

384   simultaneously, the edge guard can be expressed as $n_{jk} = \left\lceil \frac{a_j + a_k}{(b_j - a_j) + (b_k - a_k)} \right\rceil \times (a_j + a_k)$. The

385   window of the ith acceleratable cycle is $[a_i,b_i]$ as a known condition, where $0 \leq a_j \leq b_j$, $0 \leq a_k \leq$

386   b_k, and there is $a_j + a_k \geq a_j, a_k$.

387   We make $\frac{a_j}{b_j - a_j} = \frac{U}{V}$, $\frac{a_k}{b_k - a_k} = \frac{X}{Y}$, such that $\frac{a_j + a_k}{(b_j - a_j) + (b_k - a_k)} = \frac{U + X}{V + Y}$. We assume that $\frac{U}{V}$ is smaller,

388   then $\frac{X}{Y} = \frac{U + \mu}{V}$, $\mu \in \mathbb{R}^+$. Therefore, there is $\frac{U + X}{V + Y} = \frac{U + U + \mu}{V + V} = \frac{U}{V} + \frac{\mu}{2V} > \frac{U}{V}$; that is, $\left\lceil \frac{U + X}{V + Y} \right\rceil \geq \left\lceil \frac{U}{V} \right\rceil$ and

389      $\left\lceil \frac{U + X}{V + Y} \right\rceil \geq \min (\left\lceil \frac{U}{V} \right\rceil, \left\lceil \frac{X}{Y} \right\rceil)$. In the positive-number condition, a larger number multiplied by a larger

390   number is either equal to or greater than a smaller number multiplied by a smaller number;

391   therefore,

392
$$\left\lceil \frac{U + X}{V + Y} \right\rceil \times (a_j + a_k) \geq \min (\left\lceil \frac{U}{V} \right\rceil \times a_j, \left\lceil \frac{X}{Y} \right\rceil \times a_k)$$

393   which is $n_{jk} \geq \min (n_j, n_k)$.

394   According to Corollary 1, the reset location will obtain a continuous clock zone after executing

395   the acceleratable cycle, which has a smaller value of $\left\lceil \frac{a_i}{b_i - a_i} \right\rceil \times a_i$, $\left\lceil \frac{a_i}{b_i - a_i} \right\rceil$ times during forward

396   symbolic reachability analysis. This solution is faster than using two acceleratable cycles

397   simultaneously to obtain a continuous clock zone, and it is better than using the larger one.

398   By extension, when comparing any two acceleratable cycles, a shorter time cycle always obtains

399   a continuous clock zone more quickly. When comparing all acceleratable cycles, we can achieve

400   the most effective acceleratable cycle for exact acceleration. This result indicates that the

401   acceleration effect of a single acceleratable cycle is more effective than that of multiple

402   acceleratable cycles, thereby completing the proof.

403

404   **Corollary 4**. Let $M = (C,L,l_0,A,I,E)$ be a timed automaton with several acceleratable cycles. Let

405      $Cycle_i = (E_{c_i},x)$ be the ith acceleratable cycle of M with a window of $[a_i,b_i]$, where i is a non-zero

406   natural number. All acceleratable cycles affect the cycle of clock x, and their reset locations are

407   uniform in $l_{reset} \in L$. If $a_i < b_i$, then the acceleratable cycle with the min $\left(\left\lceil \frac{a_i}{b_i - a_i} \right\rceil \times a_i \right)$ has the best

408   acceleration effect of obtaining a continuous clock zone in the shortest time.

409   *Proof.* According to Theorem 4, comparing any two acceleratable cycles, a cycle with smaller

410   value of $\left(\left\lceil \frac{a_i}{b_i - a_i} \right\rceil \times a_i \right)$ always obtains a continuous clock zone more quickly. When comparing all

444   *Proof.* For a certain acceleratable cycle, its window is set as $[a',b']$. According to Theorem 1,
445   when $3a' \leq 2b'$, the appended cycle does not change the subsequent reachability of the reset
446   location $src(e_0)$ in M. According to Theorem 2, when $a' \leq b'$, the parking cycle does not change
447   the subsequent reachability of the reset location $src(e_0)$ in M. In the case of multiple acceleratable
448   cycles superimposing on the same location in a complex real-time system, the subsequent
449   reachability of the reset location $src(e_0)$ in M can be guaranteed to remain unchanged if any part
450   of the acceleratable cycle is processed with exact acceleration.
451   According to Theorem 4, the accelerable cycle is more effective for obtaining a continuous clock
452   zone at reset location $src(e_0)$ than multiple acceleratable cycles. In particular, according to
453   Corollary 4, if $a_i < b_i$, then the exact acceleration based on overlapping cycle can obtain the
454   continuous clock zone in the shortest time. The proof is completed.
455
456   This theorem ensures the effectiveness of acceleration. For a single acceleratable cycle, if all
457   states are reachable by more than one execution of the acceleratable cycle, then exactly only one
458   execution of the acceleratable cycle of the appended cycle or parking cycle can guarantee
459   reachability of all states in the accelerative automaton. The complex real-time model checking
460   differs from the exact acceleration of a single acceleratable cycle. In depth-first forward symbolic
461   reachability analysis, it is necessary to verify whether subsequent states are reachable in priority
462   while ignoring the breadth-first search within cycles.
463   In our case study of an IoT gateway security system (Wang et al., 2018), the control center must
464   complete a security process and distribute it to each sensor node. Once a self-organizing sensor
465   network completes the process, it can respond to the command of the control center in a timely
466   manner. The control center can perform subsequent operations after receiving feedback regardless
467   of whether other sensor nodes can complete the process. Hence, the security system must ensure
468   its subsequent reachability regardless of who completes the process. This approach accelerates the
469   search of subsequent states, thus avoiding time and space consumption caused by superposition of
470   acceleratable cycles.
471

Experimental Results

473 To verify the validity of the exact acceleration method based on an overlapping cycle for complex
474 real-time model checking, we collected runtime data, including memory consumption and
475 verification time, from the timed automaton M'. We also gathered runtime data from the
476 accelerated automata M_a', M_p', and M_o', which use the appended cycle, parking cycle, and
477 overlapping cycle, respectively. We employed the model-checking tool UPPAAL with a depth-
478 first search order to verify whether location L15 was reachable, which can give the time and
479 memory consumption in verification automatically. Experimental results are displayed in Table 2.
480   Results show that the time consumption of M' increased with exponential growth of LARGE at
481 a rate of nearly ten times without using exact acceleration. The memory consumption of M'
482 increased slightly because no additional locations were added. The accelerated automaton M_a' used

483 an appended cycle, which reduced the time consumption, reflecting the advantages of exact
484 acceleration. However, due to a large number of additional locations, the memory consumption of
485 M'_a increased dramatically. The accelerated automaton M'_p used a parking cycle to reduce the time
486 consumption further compared to M'_a. The fixed length of the parking cycle reduced the number
487 of additional locations compared to the appended cycle; accordingly, the memory consumption
488 was much lower for M'_p than for M'_a but slightly higher than for M'. The accelerated automaton M'_o
489 that used the proposed overlapping cycle exhibited minimal time consumption and only required
490 the addition of a single, fixed-length location for the complex real-time model. The memory
491 consumption of M'_o was close to that of M', far less than that of M'_a, and slightly better than that of
492 M'_p. We can explain the time and memory consumption of M'_o by Theorem 5. The depth-first search
493 order ensures that the overlapping cycle accelerates exploration before complete exploration of all
494 accelerable cycles.
495   A less theoretical study case involves model verification of the IoT gateway security system
496 (Wang et al., 2018). The exact acceleration method based on the overlapping cycle was
497 successfully applied in this case, significantly improving verification efficiency. The technical
498 framework of the IoT gateway security system is illustrated in Fig. 6.
499   An essential technology in the IoT gateway security system is the time-stamped advanced
500 encryption standard algorithm. By introducing a timestamp into the key expansion phase, the round
501 key can be dynamically updated with change over time to realize a cipher text change that ensures
502 the security of confidential information. Due to the introduction of a timestamp, the system
503 generates acceleratable cycles when modeled as timed automata. Multiple acceleratable cycles are
504 overlaid on the same location in particular scenarios, which requires overlapping cycle technology
505 for exact acceleration.
506   Our theory can be used to simulated the parallel execution of processes and idle cycles.
507 However, the presence of urgent locations and synchronous channels may disturb exact
508 acceleration. For example, if broadcast channel coordination occurs in an urgent location, the
509 multi-party response of the broadcast should be completed before the next state location can be
510 migrated. The execution time of the response process is not controlled by the cycle control
511 program; thus, it is not appropriate to simply use exact acceleration technology for acceleration;
512 the space–time loss of using acceleration technology should be compared to the broadcasting
513 response. However, exact acceleration technology can often handle urgent locations and
514 synchronous channels. The following case of an IoT gateway security system also involves these
515 situations. As no extra time interference exists within the whole acceleration process, the exact
516 acceleration technology can finally be successfully applied to system modeling and verification.
517   The accelerated automaton based on an overlapping cycle is an approximation that can be
518 adapted to verify the accuracy of invariance and reachability properties. Consider the processes in
519 Fig. 7; these processes model a top architecture and a middleware control program consisting of
520 locations, edges, and channels.

521   The IoT gateway runs from the Start location, reads configuration information and performs
522   gateway identity authentication. The underlying unified authentication service is invoked through
523   channel StartAC for security authentication, and *GatewayStatus* is returned after authentication. If
524   *GatewayStatus*=true, then the system enters the location EnterMiddle and transforms to the polling
525   module of the middle layer through the StartMiddle channel. For the middle-layer polling module,
526   polling begins through the StartMiddle channel, and the top-layer main module is returned by the
527   FinishMiddle channel. Location CheckCategory controls whether polling logic ends up at a
528   perception terminal or an execution terminal, which each have different processing methods. In
529   the two polling processes, the underlying security service is invoked through the synchronization
530   channel according to different requirements. The specific process can be interpreted by the
531   meaning of state locations, synchronization channels, and variables as described by Wang et al.
532   (2018). In particular, clock constraints are added during the stage of waiting for timing and the
533   stage of keeping the equipment running. The top-layer main module model and middle-layer
534   polling module model constitute the general framework of the IoT gateway security system.
535   Security implementation depends on the underlying security service modules ultimately, hence it
536   is necessary to improve model construction of the underlying security service modules. For
537   complex system modeling with underlying services, we apply the exact acceleration method
538   proposed in this paper, which can effectively improve the verification speed.
539   To demonstrate the effect of exact acceleration, we checked all security properties of the IoT
540   gateway security system model in Fig. 7. Several examples are presented below.
541   (1) A[] not deadlock
542   Property 1 is used to check deadlock and ensure all state locations will be reachable.
543   (2) E<> Top.CheckGS
544   (3) E<> Top.EnterMiddle imply Middle.CheckCategory
545   Properties 2 and 3 are used to explore part of the state space. The truth of these two properties
546   indicates that the implementation accelerated model is an exact acceleration with the overlapping
547   cycle.
548   (4) A[] Top.Restart imply $c<=300$
549   (5) A[] Top.Record imply $c<=600$
550   (6) A[] Middle.RetrieveData imply $Middle.y>=30$
551   (7) A[] Middle.WaitDevice imply $Middle.y<=5$
552   Properties 4–7 are examined in terms of whether subsequent states of the reset location are
553   reachable. Clock c is a global clock and clock y is used to model the duration of one process.
554   We measured time and memory consumption and explored states for these properties. The IoT
555   gateway security system was modeled as a timed automaton M_{IoT}, and the acceleration of M_{IoT}
556   with overlapping cycles was modeled as an automaton M_{IoT_o}. We used model checkers UPPAAL
557   and KRONOS to verify security system properties automatically, such as confidentiality,
558   availability, and authenticity in parallel processes. KRONOS is able to complete the statistics of
559   the state scale traversed by the whole verification process. It makes up for the fact that UPPAAL
560   can't do this. Table 3 lists the experimental results.

561   On the premise of guaranteeing the security of IoT gateway system, a large number of
562 underlying services and various applications can be embedded in the system framework. At this
563 time, the security requirements of IoT gateway system are mainly for various new access services,
564 and the framework security of the gateway itself can be maintained by its own mechanism. After
565 access to a large number of services and applications, the original model will become complex,
566 concurrent, real-time with large-scale. The verification of the system needs to be processed by the
567 exact acceleration method based on overlapping cycle.

568   With the increase of the number of access services, the system model becomes more and more
569 complex, and the scale of access number greatly affect the efficiency of model verification.
570 Appended cycle and parking cycle methods are more suitable for single accelerating cycle
571 scenarios. In this complex scenario, when the number of services reaches a certain level, the
572 acceleration process may not be completed. According to the change of the number of access
573 services, Table 4 gives the comparison of the acceleration effects of different exact acceleration
574 methods (from the perspective of time).

575   The results show that for complex real-time systems, the acceleration efficiency of overlapping
576 cycle is much higher than that of appended cycle and parking cycle, and the verification can still
577 be completed when the state scale reaches 10^7 with proposed method. So, the exact accelerating
578 technology substantially reduced the time required for verification in complex real-time model
579 checking. Overlapping cycle acceleration demonstrated the highest efficiency compared to the
580 appended cycle and parking cycle. In the simple example of automaton M' in Fig. 4, 55 additional
581 locations were required when using the appended cycle, much higher than the number of locations
582 in the original model. Although the appended cycle reduced verification time, it increased the
583 difficulty of adding locations to the model in an early stage. When many acceleratable cycles were
584 stacked in the same reset position, more than one location needed to be added to M' when the
585 parking cycle was used, although the length of the parking cycle was fixed. The parking cycle was
586 neither simpler nor faster than the overlapping cycle, and its previous calculation was larger than
587 that of the overlapping cycle.

588   With the exception of this IoT case, our approach can be applied to other scenarios, such as
589 security validation of blockchain smart contracts. The complete code and UPPAAL model can be
590 found at https://github.com/iegqwang/UPPAAL.

591

Conclusions

593 To solve the fragmentation problem for complex real-time model checking, we propose an exact
594 acceleration method based on an overlapping cycle, which is an application scenario extension of
595 parking-cycle technique, to accelerate forward symbolic reachability analysis. Compared with the
596 appended cycle or parking cycle for exact acceleration, the proposed method can be applied to the
597 model acceleration of large-scale complex real-time systems and only requires the addition of a
598 single, fixed-length location to the system's timed automaton model. The addition of an
599 overlapping cycle introduces far fewer symbolic states than using either an appended cycle or
600 parking cycle. Rather than relying on windows of acceleratable cycles, the proposed accelerated

601 automaton model is more straightforward and reduces the space–time overhead of exact
602 acceleration.
603   Two aspects warrant exploration in future research. First, we must continue to study the
604 algorithm for the acceleratable cycle, try to simplify the original automaton model, guarantee its
605 original property, and rapidly identify the deadlock. Second, we plan to develop a simple exact
606 acceleration automatic checking platform that can consider other practical conditions such as
607 action transitions, urgent locations, and synchronous channels to solve actual modeling problems
608 more efficiently.
609

References

611 **Aceto L, Bouyer P, Burgueño A, Larsen KG. 2003.** The power of reachability testing for
612 timed automata. *Theoretical Computer Science* 300(1):411-475 DOI 10.1016/S0304-
613 3975(02)00334-1.
614 **Alur R, Dill DL. 1994.** A theory of timed automata. *Theoretical Computer Science* 126(2):183-
615 235 DOI 10.1016/0304-3975(94)90010-8.
616 **Barnat J, Bauch P, Brim L, Češka M. 2010.** Employing multiple CUDA devices to accelerate
617 LTL model checking. In: *International Conference on Parallel and Distributed Systems*, *2010*.
618 Piscataway: IEEE, 259-266 DOI 10.1109/ICPADS.2010.82.
619 **Behrmann G, Bouyer P, Larsen KG, Pelánek R. 2006.** Lower and upper bounds in zone-based
620 abstractions of timed automata. *International Journal on Software Tools for Technology Transfer*
621 8(3):204-215 DOI 10.1007/s10009-005-0190-0.
622 **Behrmann G, David A, Larsen KG. 2004.** A tutorial on Uppaal. In: *International School on*
623 *Forman Methods for the Design of Computer, Communication and Software Systems*, 2004.
624 Berlin: Springer Verlag, 200-236 DOI 10.1007/978-3-540-30080-9_7.
625 **Boudjadar A, David A, Kim JH, Larsen KG, Mikučionis M, Nyman U, Skou A. 2016.**
626 Statistical and exact schedulability analysis of hierarchical scheduling systems. *Science of*
627 *Computer Programming* 127: 103-130 DOI 10.1016/j.scico.2016.05.008.
628 **Chadli M, Kim JH, Larsen KG, Legay A, Naujokat S, Steffen B, Traonouez LM. 2018.**
629 High-level frameworks for the specification and verification of scheduling problems.
630 *International Journal on Software Tools for Technology Transfer* 20(4):397-422 DOI
631 10.1007/s10009-017-0466-1.
632 **Chen H, Cui L. 2016.** Design and model checking of service-oriented software architecture for
633 Internet of things: A survey. *Chinese Journal of Computers* 39(5):853-871 DOI
634 10.11897/SP.J.1016.2016.00853.
635 **Cruz JP, Kaji Y, Yanai N. 2018.** RBAC-SC: Role-based access control using smart contract.
636 *IEEE Access* 6:12240-12251 DOI 10.1109/ACCESS.2018.2812844.
637 **Dubout C, Fleuret F. 2013.** Accelerated training of linear object detectors. In: *Computer Vision*
638 *and Pattern Recognition Workshops, 2013.* Piscataway: IEEE, 572-577 DOI
639 10.1109/CVPRW.2013.156.

640   **Gou L, Li Z, Wang C, Zhuang L. 2014.** A method to determine the exact acceleration
641   efficiency in model checking. *Journal of Zhongyuan University of Technology* 25(4):37-41 DOI
642   10.3969/j.issn.1671-6906.2014.04.009.
643   **Grishchenko I, Maffei M, Schneidewind C. 2018.** Foundations and tools for the static analysis
644   of ethereum smart contracts. In: *Computer Aided Verification, 2018.* Berlin: Springer Verlag, 51-
645   78 DOI 10.1007/978-3-319-96145-3_4.
646   **Han D, Yang Q, Xing J. 2015.** UML-based modeling and formal verification for software self-
647   adaptation. *Journal of Software* 26(4):730-746 DOI 10.13328/j.cnki.jos.004758.
648   **Hendriks M, Larsen KG. 2002.** Exact acceleration of real-time model checking. *Electronic*
649   *Notes in Theoretical Computer Science* 65(6):120-139 DOI 10.1016/S1571-0661(04)80473-0.
650   **Herbreteau F, Srivathsan B, Walukiewicz L. 2016.** Better abstractions for timed automata.
651   *Information and Computation* 251:67-90 DOI 10.1016/j.ic.2016.07.004.
652   **Iversen T K, Kristoffersen K J, Larsen K G, Laursen M, Madsen R G, Mortensen S K,**
653   **Pettersson P, Thomasen C B. 2000.** Model-checking real-time control programs: Verifying
654   LEGO® MINDSTROMS™ systems using UPPAAL. In: *Euromicro Conference on Real-Time*
655   *Systems, 2000.* Piscataway: IEEE, 147-155 DOI 10.1109/EMRTS.2000.854002.
656   **Jeong H, Yoo Y, Yi K M, Choi J Y. 2014.** Two-stage online inference model for traffic pattern
657   analysis and anomaly detection. *Machine Vision and Applications* 25(6):1501-1517 DOI
658   10.1007/s00138-014-0629-y.
659   **Konur S, Fisher M, Schewe S. 2013.** Combined model checking for temporal, probabilistic, and
660   real-time logics. *Theoretical Computer Science* 503:61-88 DOI 10.1016/j.tcs.2013.07.012.
661   **Lee J, Yu S, Park K, Y Park, Park Y. 2019.** Secure three-factor authentication protocol for
662   multi-gateway IoT environments. *Sensors* 19(10):2358-2383 DOI 10.3390/s19102358.
663   **Li G, Wei Q, Li L, Jin Z, Xu Y, Zheng L. 2013.** Environment based modeling approach for
664   services in the Internet of things. *Science China Press*, 43(10):1198-1218 DOI
665   10.1360/N112013-00031.
666   **Lin K, Chen Y, Xu D. 2017.** Reliability assessment model considering heterogeneous
667   population in a multiple stresses accelerated test. *Reliability Engineering & System Safety*
668   165:134-143 DOI 10.1016/j.ress.2017.03.013.
669   **Möller MO. 2002.** Parking can get you there faster: Model augmentation to speed up real-time
670   model checking. *Electronic Notes in Theoretical Computer Science* 65(6):202-217 DOI
671   10.1016/S1571-0661(04)80477-8.
672   **Pinisetty S, Jéron T, Tripakis S, Falcone Y, Marchand H, Preoteasa V. 2017.** Predictive
673   runtime verification of timed properties. *Journal of Systems and Software* 132:353-365 DOI
674   10.1016/j.jss.2017.06.060.
675   **Qanadilo M, Samara S, Zhao Y. 2013.**c. In: *Latin-American Symposium on Dependable*
676   *Computing, 2013.* Piscataway: IEEE, 40-47 DOI 10.1109/LADC.2013.20.
677   **Wang C, Pastore F, Briand L. 2019.** Oracles for testing software timeliness with uncertainty.
678   *ACM Transactions on Software Engineering and Methodology* 8(1):1-30 DOI 10.1145/3280987.

679   **Wang G, Zhuang L, Wang R, Song Y, Zhang K. 2018.** Modeling and verifying based on
680   timed automata of Internet of things gateway security system. *Journal on Communications*
681   39(3):63-75 DOI 10.11959/j.issn.1000-436x.2018042.
682   **Wang H, Zhong D, Zhao T, Ren F. 2019.** Integrating model checking with SysML in complex
683   system safety analysis. *IEEE Access* 7:16561-16571 DOI 10.1109/ACCESS.2019.2892745.
684   **Yin C, Song W, Zhuang L. 2010.** Method of acceleratable cycles in identify timed automata.
685   *Computer Engineering and Design* 31(23):5113-5115 DOI 10.16208/j.issn1000-
686   7024.2010.23.030.
687   **Yin C, Zhuang L, Wang C. 2011.** Exact acceleration of real-time model checking based on
688   parking cycle. *Acta Electronica Sinica* 39(3):489-493 DOI 10.3969/j.issn.0372-
689   2112.2011.03.001.
690   **Zhang F, Bu L, Wang L, Zhao J, Li X. 2013.** Modeling and analysis of wireless sensor
691   network protocols by stochastic timed automata and statistical model checking. *Scientia Sinica*
692   *Informationis* 43(1):90-107 DOI 10.1360/112012-498.

Figure 1

Timed automaton *M.*

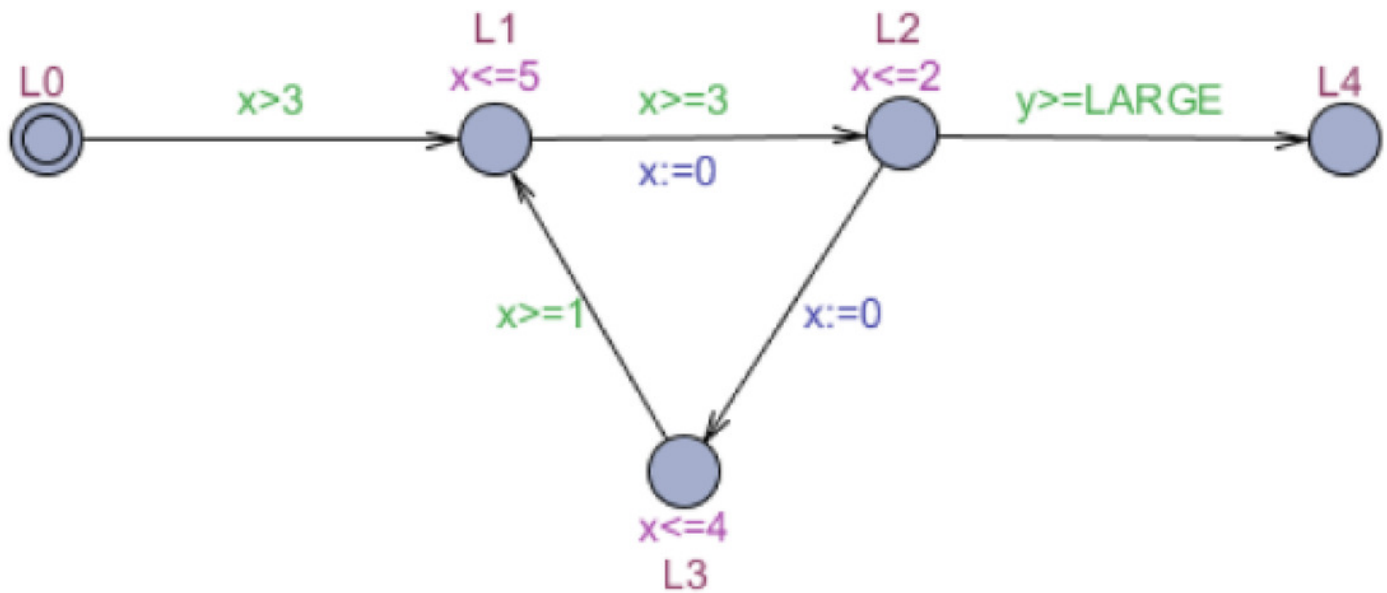



Figure 2

Accelerated automaton M_a based on appended cycle.

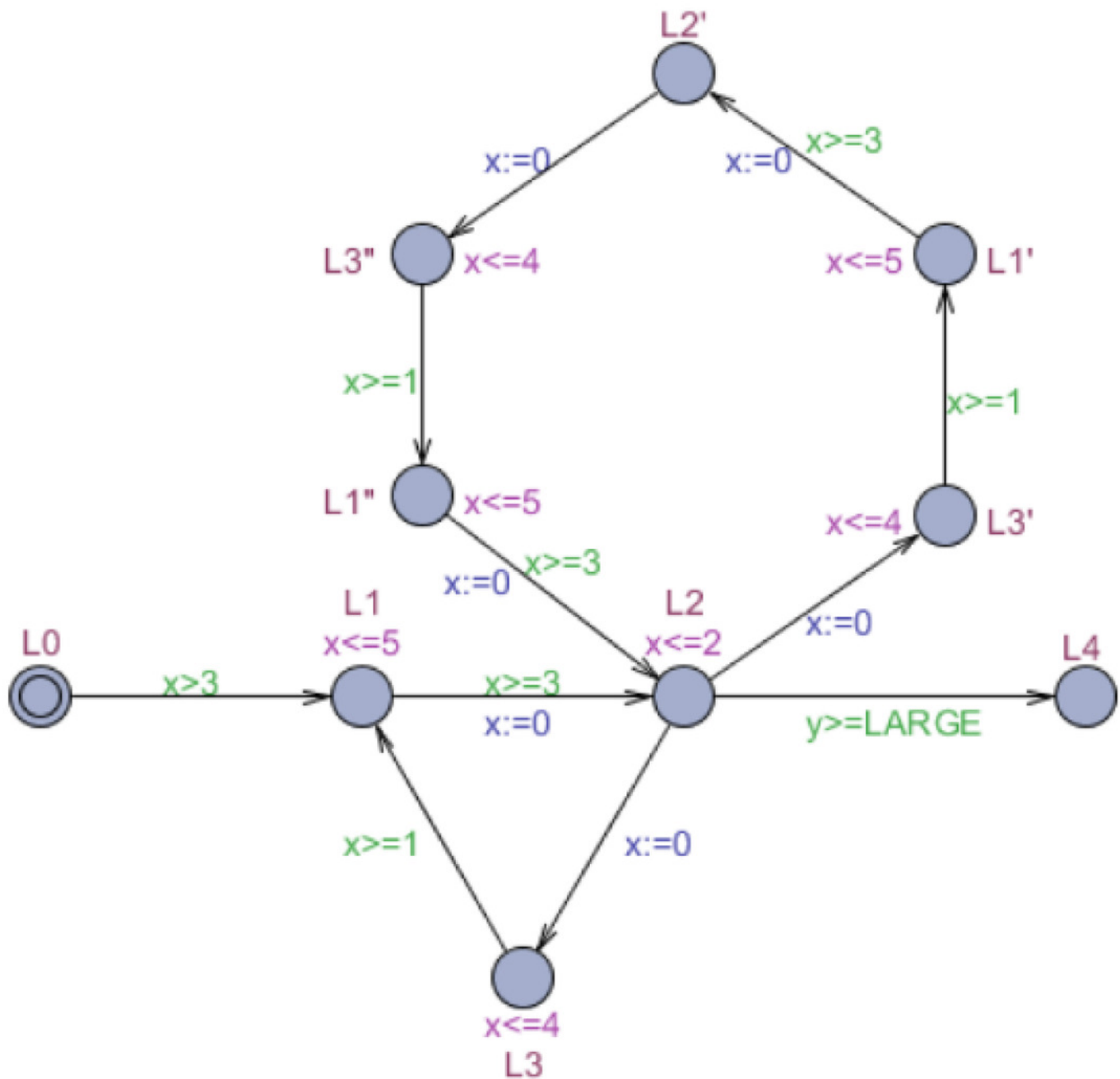



Figure 3

Accelerated automaton M_p based on parking cycle.

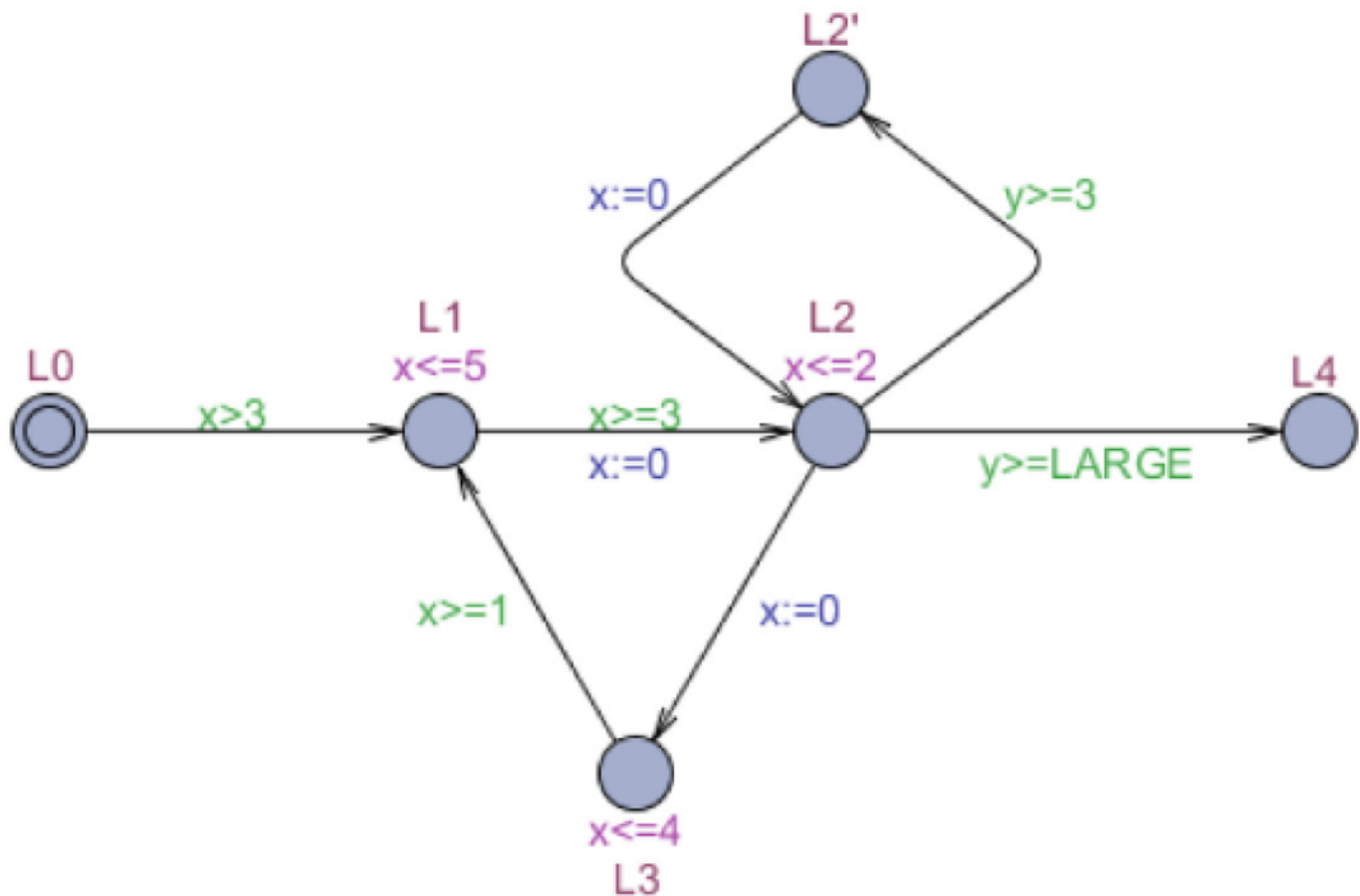



Figure 4

Timed automaton M'.

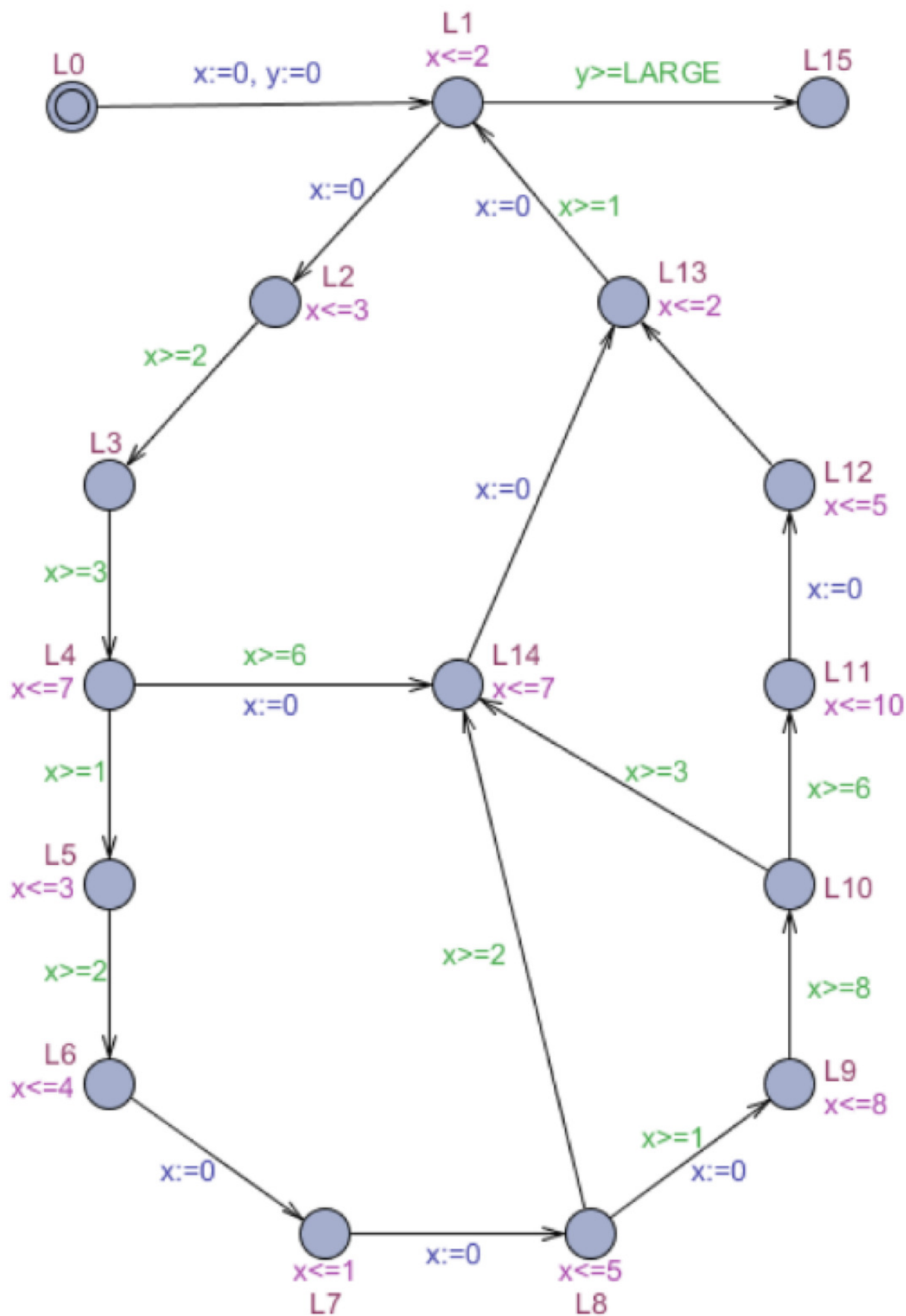



Figure 5

Automaton M'_o: Acceleration of M' based on an overlapping cycle.

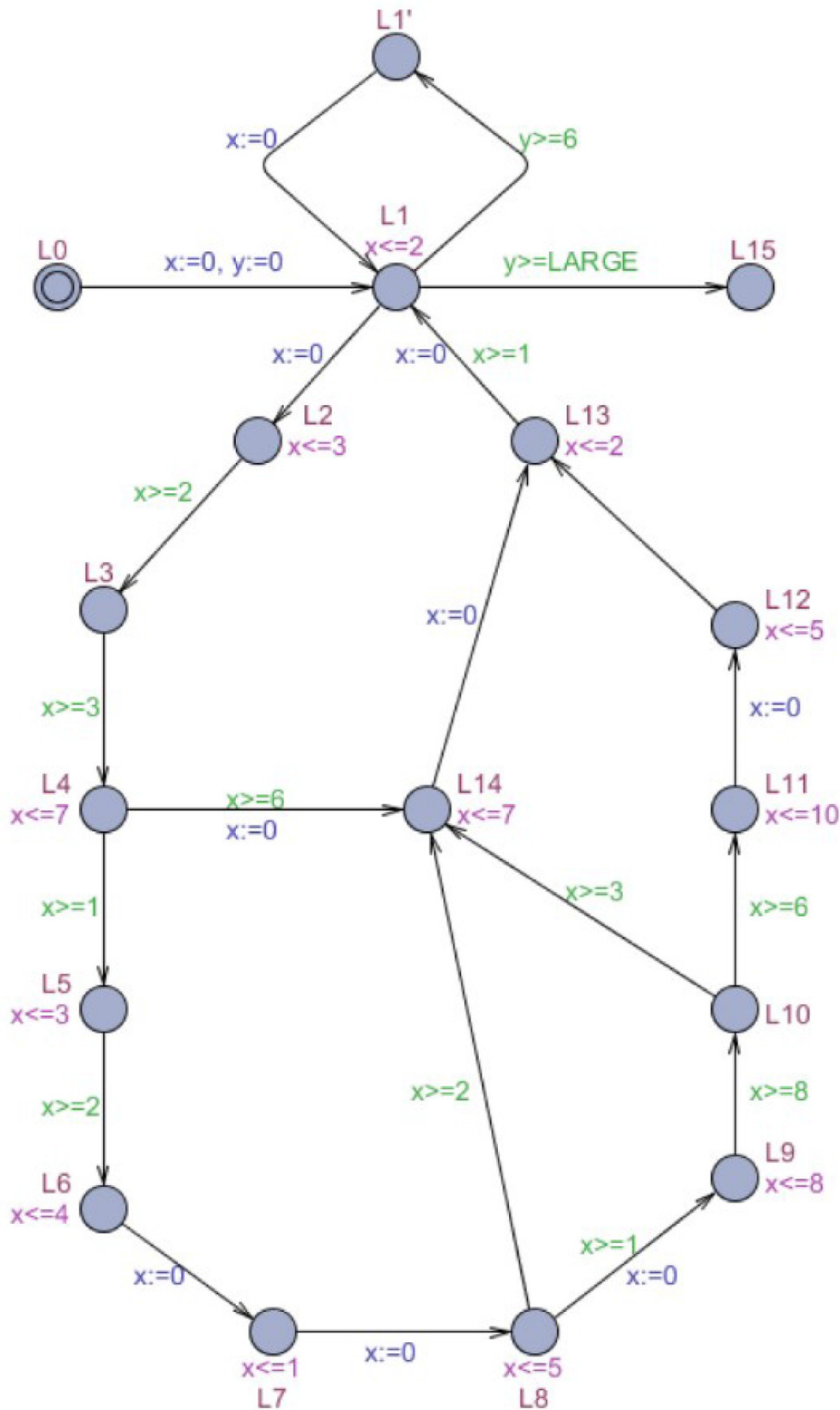
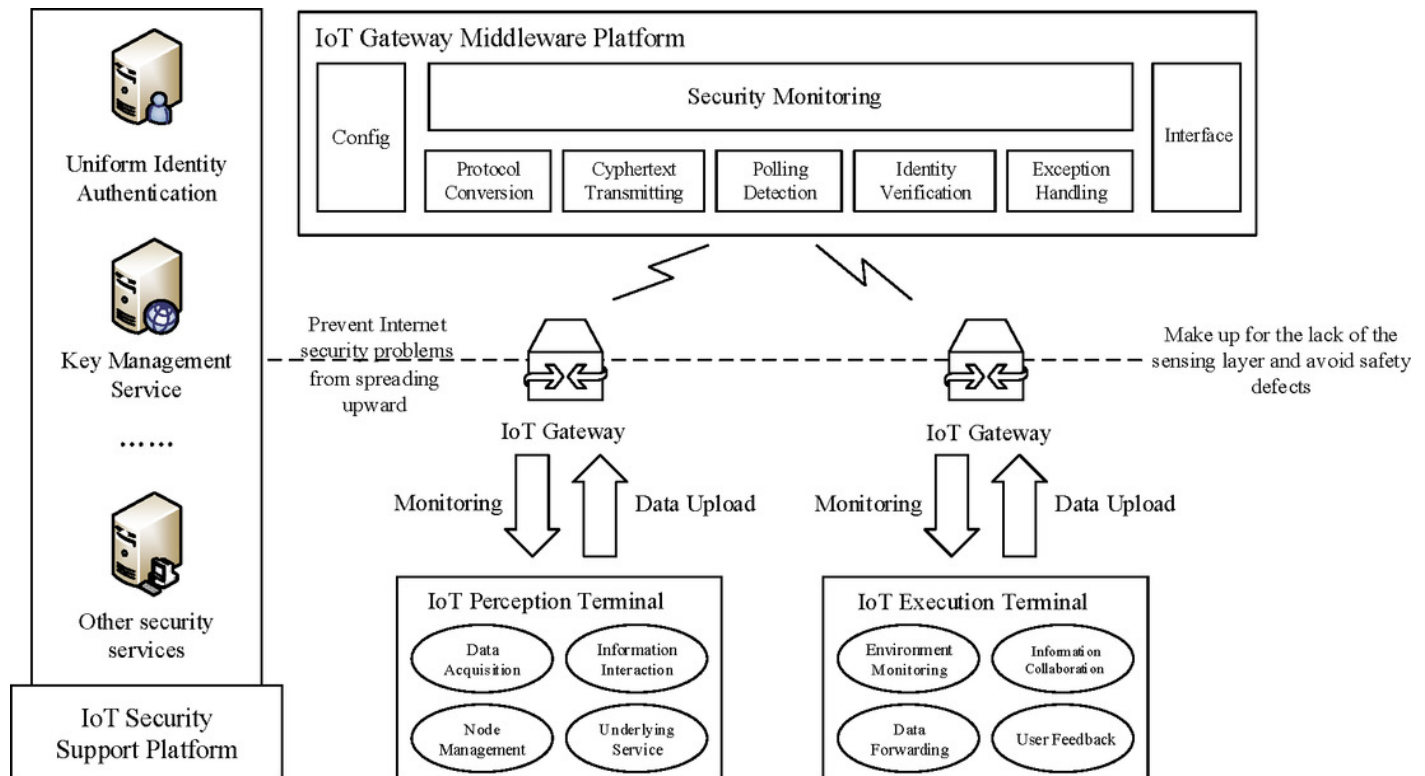
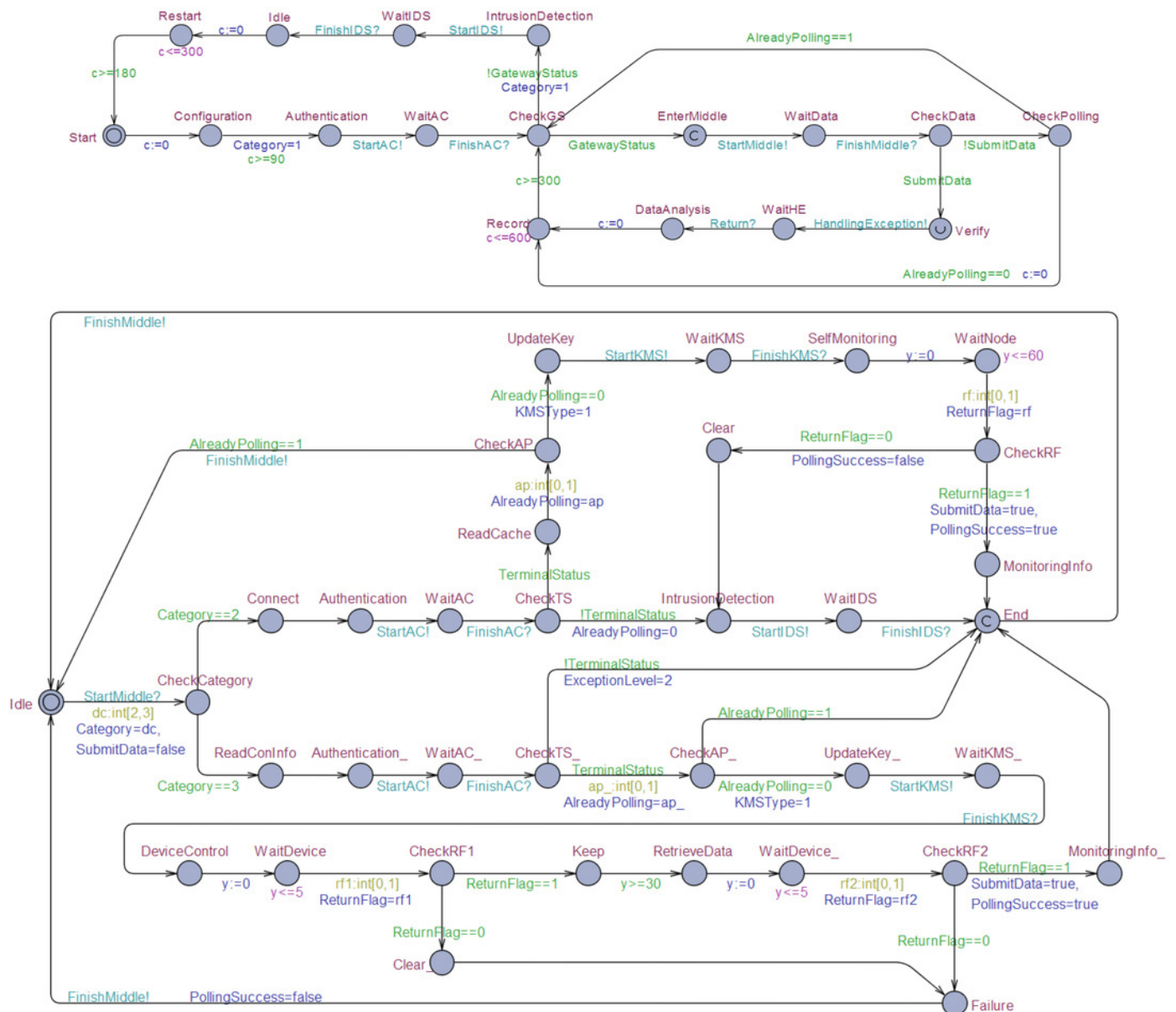



Figure 6

Technical framework of IoT gateway security system.

Figure 7

Part of the IoT gateway security system model.

Results of symbolic states from a forward symbolic exploration by timed automaton M.

1 **Table 1. Results of symbolic states from a forward symbolic exploration by timed automaton**

2 **_M_.**

State	Location	Zone		
1	L0	$y=0$	$x=0$	$y\text{-}x=0$
2	L1	$3<y\leq5$	$3<x\leq5$	$y\text{-}x=0$
3	L2	$3<y\leq7$	$0\leq x\leq2$	$3<y\text{-}x\leq5$
4	L3	$3<y\leq11$	$0\leq x\leq4$	$3<y\text{-}x\leq7$
5	L1	$4<y\leq12$	$1\leq x\leq5$	$3<y\text{-}x\leq7$
6	L2	$6<y\leq14$	$0\leq x\leq2$	$6<y\text{-}x\leq12$
7	L3	$6<y\leq18$	$0\leq x\leq4$	$6<y\text{-}x\leq14$
8	L1	$7<y\leq19$	$1\leq x\leq5$	$6<y\text{-}x\leq14$
9	L2	$9<y\leq21$	$0\leq x\leq2$	$9<y\text{-}x\leq19$
10	L3	$9<y\leq25$	$0\leq x\leq4$	$9<y\text{-}x\leq21$
11	L1	$10<y\leq26$	$1\leq x\leq5$	$9<y\text{-}x\leq21$
12	L2	$12<y\leq28$	$0\leq x\leq2$	$12<y\text{-}x\leq26$

Runtime data comparing M' and its accelerated versions M'_a, M'_p and M'_o.

1 **Table 2. Runtime data comparing M' and its accelerated versions M_a', M_p' and M_o'.**

		LARGE					
		10^4	10^5	10^6	10^7	10^8	10^9
M'	Mem (KB)	27020	26892	27392	27544	27952	28572
	Time (s)	0.032	0.256	1.688	12.045	120.33	1191.02
M_a'	Mem (KB)	28328	29220	30468	31688	32508	34104
	Time (s)	0.007	0.008	0.008	0.008	0.009	0.009
M_p'	Mem (KB)	27184	27384	27788	28060	28208	29276
	Time (s)	0.005	0.004	0.003	0.003	0.003	0.004
M_o'	Mem (KB)	27164	27036	27472	27824	27988	28788
	Time (s)	0.002	0.002	0.002	0.003	0.002	0.002

Runtime data comparing M_{IoT} and M_{IoTo}.

1   **Table 3. Runtime data comparing M_{IoT} and M_{IoT_o}.**

	Explored	Time(s)	Memory(KB)
M_{IoT}	108302	71.151	29660
M_{IoT_o}	47545	1.049	30840

Comparing the performance of different exact acceleration techniques for large-scale IoT systems.

1   **Table 4. Comparing the performance of different exact acceleration techniques for large-**
2   **scale IoT systems.**

System States-scale	Exact Acceleration Technique	Verification Time(s)
10^4	Appended cycle	277.860
10^4	Parking cycle	0.893
10^4	Overlapping cycle	0.015
10^5	Appended cycle	∞
10^5	Parking cycle	72.218
10^5	Overlapping cycle	1.020
10^6	Parking cycle	364.720
10^6	Overlapping cycle	43.292
10^7	Parking cycle	∞
10^7	Overlapping cycle	409.132