On the determinization of event-clock input-driven pushdown automata*

Mizuhito Ogawa[1] and Alexander Okhotin[2][0000−0002−1615−2725]

[1] Japan Advanced Institute of Science and Technology, Japan, mizuhito@jaist.ac.jp
[2] Department of Mathematics and Computer Science, St. Petersburg State University, Russia alexander.okhotin@spbu.ru

Abstract. A timed extension of input-driven pushdown automata (also known as visibly pushdown automata and as nested word automata) under the event-clock model was introduced by Nguyen and Ogawa ("Event-clock visibly pushdown automata", 2009), who showed that this model can be determinized using the method of region construction. This paper proposes a new, direct determinization procedure for these automata: an n-state nondeterministic automaton with k different clock constraints is transformed to a deterministic automaton with 2^{n^2} states, 2^{n^2+k} stack symbols and the same clock constraints as in the original automaton. The construction is shown to be asymptotically optimal with respect to both the number of states and the number of stack symbols.

Keywords: Timed systems, input-driven pushdown automata, visibly pushdown automata, determinization, state complexity.

1 Introduction

Timed automata (TA), introduced by Alur and Dill [2], are finite automata operating in real time. These automata enjoy decidability of the emptiness problem (equivalently, the state reachability problem) and are implemented as UPPAAL [7] for safety checking. The decidability of emptiness holds under various extensions of the model equipped with a pushdown store, such as the *Dense-Timed Pushdown Automata* (DTPDA) of Abdulla et al. [1] with *ages* (representing local clocks), which are further analyzed by Clemente and Lasota [11].

Although the emptiness problem for timed automata is decidable, timed automata are not closed under complementation, and their nondeterministic case cannot generally be determinized. Their inclusion problem is decidable only in the case of a single clock [18], and becomes undecidable for two clocks [2].

As an alternative timed device, the class of *event-clock automata* (ECA) was introduced by Alur et al. [3] and further studied by Geeraerts et al. [12]: this class allows determinization and complementation, and hence it enjoys decidable inclusion problem. An ECA is defined with a "prophecy clock" and a "history

* Supported by the Russian Foundation for Basic Research under grant 20-51-50001.

clock" bound to each input symbol. The history clock $\overleftarrow{x_a}$ associated with an input symbol a is always reset when a is read, and the prophecy clock $\overrightarrow{x_a}$ predicts the next occurrence of a.

In general, when a stack is introduced, this often destroys the decidability of the inclusion problem, since asynchronous behavior of two stacks disrupts a direct product of two devices. Even starting from finite automata, adding the stack makes the inclusion undecidable.

To remedy this, a constraint on the synchronous behaviour of stacks is imposed upon the model. The resulting *input-driven pushdown automata* [14,10] (IDPDA), also known as *visibly pushdown automata* [5] and as *nested word automata* [6], are defined over an alphabet split into three parts: *left brackets* Σ_{+1}, on which the automaton must push one stack symbol, *right brackets* Σ_{-1}, on which the automaton must pop one stack symbol, and *neutral symbols* Σ_0, on which the automaton ignores the stack. Unlike the standard pushdown automata, IDPDA are closed under all Boolean operations, and they can be determinized [10]. An extensive study of this model was initiated by Alur and Madhusudan [5,6], who, in particular, established a lower bound on the determinization complexity, accordingly starting a line of research on the succinctness of description for this model [16], and also defined a Büchi-like extension for infinite strings, which has also received further attention [13,17].

Event-clock visibly pushdown automata, which combine the ideas of input-driven pushdown and event-clock automata, were proposed by Nguyen and Ogawa [19], who proved that this model can be determinized. Their work was followed and extended by Bhave et al. [8] and Bozzelli et al. [9]. This paper revisits this model, with the aim to improve the determinization procedure. In addition, the model is further extended by introducing special event clocks recording the duration of the call/return relation. The resulting model is called *event-clock input-driven pushdown automata* (ECIDPDA).

The proposed determinization procedure is *direct*, in the sense that it *does not rely on the classical discretization* or "untime translation" method, and *is not based on the region construction*, which handles the extension by the age of a stack symbol in Bhave et al. [8]. Even though direct determization was once used by Alur and Madhusudan [4] for determinizing event-clock finite automata with only history clocks ($\overleftarrow{x_a}$), this idea, up to the authors' knowledge, did not receive any further development in the literature; in particular, all the existing work on input-driven/visibly pushdown event-clock automata relies on more sophisticated determinization constructions.

As per the proposed construction, presented in Section 3, any given n-state nondeterministic automaton with k different clock constraints and with any number of stack symbols is transformed to a deterministic automaton with 2^{n^2} states, 2^{n^2+k} stack symbols and the same clock constraints as in the original automaton. Furthermore, in Section 4, this construction is shown to be asymptotically optimal both with respect to the number of states and with respect to the number of stack symbols.

2 Definitions

Event-clock automata operate on *timed strings* over an alphabet Σ, that is, sequences of the form $w = (a_1, t_1) \dots (a_n, t_n)$, where $a_1 \dots a_n \in \Sigma^*$ is a string, and $t_1 < \dots < t_n$ are real numbers indicating the time of the symbols' appearance.

For input-driven pushdown automata, the alphabet Σ is split into three disjoint classes: $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$, where symbols in Σ_{+1} are called *left brackets*, symbols in Σ_{-1} are *right brackets*, and Σ_0 contains *neutral symbols*. An input-driven pushdown automaton always pushes one stack symbol upon reading a left bracket, pops one stack symbol upon reading a right bracket, and does not access the stack on neutral symbols. Typically, a string over such an alphabet is assumed to be *well-nested* with respect to its left and right brackets, but Alur and Madhusudan [5] adapt the definition to handle ill-nested inputs.

The proposed *event-clock input-driven pushdown automata* (ECIDPDA) operate on timed strings over an alphabet $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$. These automata operate like input-driven pushdown automata, and additionally can evaluate certain constraints upon reading each input symbol. These constraints refer to the following *clocks*, each evaluating to a real number:

- a symbol history clock $\overleftarrow{x_a}$, with $a \in \Sigma$, provides the time elapsed since the symbol a was last encountered;
- a symbol prediction clock $\overrightarrow{x_a}$, with $a \in \Sigma$, foretells the time remaining until the symbol a will be encountered next time;
- a stack history clock $\overleftarrow{x_{\text{push}}}$, defined on a right bracket, evaluates to the time elapsed since the matching left bracket;
- a stack prediction clock $\overrightarrow{x_{\text{pop}}}$, defined on a left bracket, foretells the time remaining until the matching right bracket.

These values are formally defined as follows.

Definition 1. *Let $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ be an alphabet. The set of clocks over Σ is $\mathcal{C}(\Sigma) = \{ \overleftarrow{x_a} \mid a \in \Sigma \} \cup \{ \overrightarrow{x_a} \mid a \in \Sigma \} \cup \{ \overleftarrow{x_{\text{push}}}, \overrightarrow{x_{\text{pop}}} \}$. Then the value of a clock from $\mathcal{C}(\Sigma)$ on a timed string $w = (a_1, t_1) \dots (a_n, t_n)$ at position $i \in \{1, \dots, n\}$ is defined as follows.*

$$
\begin{aligned}
\overleftarrow{x_a} &= t_i - t_j, && \text{for greatest } j < i \text{ with } a_j = a \\
\overrightarrow{x_a} &= t_j - t_i, && \text{for least } j > i \text{ with } a_j = a \\
\overleftarrow{x_{\text{push}}} &= t_i - t_j, && \text{if } a_j \in \Sigma_{+1} \text{ and } a_i \in \Sigma_{-1} \text{ match each other} \\
\overrightarrow{x_{\text{pop}}} &= t_j - t_i, && \text{if } a_i \in \Sigma_{+1} \text{ and } a_j \in \Sigma_{-1} \text{ match each other}
\end{aligned}
$$

In each case, if no such j exists, then the value of the clock is undefined.

The original model by Nguyen and Ogawa [19] used only symbol history clocks $\overleftarrow{x_a}$ and symbol prediction clocks $\overrightarrow{x_a}$. Stack history clocks $\overleftarrow{x_{\text{push}}}$ were first introduced by Bhave et al. [8], who called them *the age of stack symbols*. As compared to the definition of Bhave et al. [8], another clock type, the stack prediction clock $\overrightarrow{x_{\text{pop}}}$, has been added to the model for symmetry.

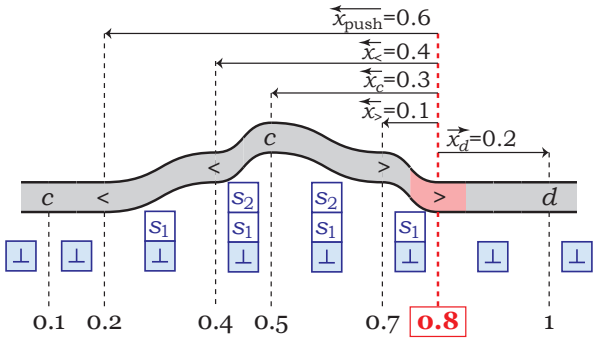

Fig. 1. Clock values for the string $w = (0.1, c)(0.2, <)(0.4, <)(0.5, c)(0.7, >)\mathbf{(0.8, >)}(1, d)$, at the last right bracket, as in Example 1.

A clock constraint is a logical formula that restricts the values of clocks at the current position: clock values can be compared with constants, and any Boolean combination of such conditions can be expressed.

Definition 2. *Let $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ be an alphabet. The set of clock constraints over Σ, denoted by $\Phi(\Sigma)$, consists of the following formulae.*

- *For every clock $C \in \mathcal{C}(\Sigma)$ and for every non-negative constant $\tau \in \mathbb{R}$, the following are atomic clock constraints: $C \leqslant \tau$; $C \geqslant \tau$.*
- *If φ and ψ are clock constraints, then so are $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$ and $\neg\varphi$.*

Let $w = (a_1, t_1) \dots (a_n, t_n)$ be a timed string, let $i \in \{1, \dots, n\}$ be a position therein. Each clock constraint can be either true or false on w at position i.

- *$C \leqslant \tau$ is true if the value of C on w at position i is defined and is at most τ.*
- *$C \geqslant \tau$ is true if the value of C on w at i is defined and is at least τ.*
- *$(\varphi \vee \psi)$ is true on w at i, if so is φ or ψ;*
- *$(\varphi \wedge \psi)$ is true on w at i, if so are both φ and ψ;*
- *$\neg\varphi$ is true on w at i, if φ is not.*

The following abbreviations are used: $C = \tau$ stands for $(C \leqslant \tau \wedge C \geqslant \tau)$; $C < \tau$ stands for $(C \leqslant \tau \wedge \neg(C \geqslant \tau))$; $C > \tau$ stands for $(C \geqslant \tau \wedge \neg(C \leqslant \tau))$.

Example 1. Let $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$, with $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$ and $\Sigma_0 = \{c, d\}$. Let $w = (0.1, c)(0.2, <)(0.4, <)(0.5, c)(0.7, >)\mathbf{(0.8, >)}(1, d)$ be a well-nested timed string over Σ, illustrated in Figure 1. Then, the values of the clocks at position 6 (the last right bracket) are: $\overleftarrow{x_{\text{push}}} = 0.8 - 0.2 = 0.6$, $\overleftarrow{x_<} = 0.4$, $\overleftarrow{x_c} = 0.3$, $\overleftarrow{x_>} = 0.1$, $\overrightarrow{x_d} = 1 - 0.8 = 0.2$, and $\overleftarrow{x_d}$, $\overrightarrow{x_<}$, $\overrightarrow{x_c}$, $\overrightarrow{x_>}$, $\overrightarrow{x_{\text{pop}}}$ are undefined. Accordingly, the clock constraint $\overleftarrow{x_{\text{push}}} > 0.1 \vee \overrightarrow{x_c} \geqslant 0$ is true, whereas $\overleftarrow{x_c} > 0.1 \wedge \overrightarrow{x_d} < 0.2$ is false.

An event-clock automaton is equipped with a finite set of such clock constraints. At each step of its computation, it knows the truth value of each of them, and can use this information to determine its transition. The following definition is based on Nguyen and Ogawa [19] and on Bhave et al. [8].

Definition 3. *A nondeterministic event-clock input-driven pushdown automaton (ECIDPDA) is an octuple* $\mathcal{A} = (\Sigma_{+1}, \Sigma_0, \Sigma_{-1}, Q, Q_0, \Gamma, \langle \delta_a \rangle_{a \in \Sigma}, F)$*, where:*

- $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ *is an input alphabet split into three disjoint classes;*
- Q *is a finite set of states;*
- Γ *is the pushdown alphabet;*
- $Q_0 \subseteq Q$ *is the set of initial states;*
- *for each neutral symbol* $c \in \Sigma_0$*, the state change is described by a partial function* $\delta_c \colon Q \times \Phi(\Sigma) \to 2^Q$*;*
- *the transition function by each left bracket symbol* $< \in \Sigma_{+1}$ *is* $\delta_< \colon Q \times \Phi(\Sigma) \to 2^{Q \times \Gamma}$*, which, for a given current state and the truth value of clock constraints, provides zero or more transitions of the form (next state, symbol to be pushed);*
- *for every right bracket symbol* $> \in \Sigma_{-1}$*, there is a partial function* $\delta_> \colon Q \times (\Gamma \cup \{\bot\}) \times \Phi(\Sigma) \to 2^Q$ *specifying possible next states, assuming that the given stack symbol is popped from the stack, or the stack is empty* ($\bot$)*;*
- $F \subseteq Q$ *is the set of accepting states.*

The domain of the transition function by each symbol must be finite.

An accepting computation of $\mathcal{A}$ *on a timed string* $w = (a_1, t_1) \dots (a_n, t_n)$ *is any sequence* (q_0, α_0), (q_1, α_1), $\dots$, (q_n, α_n)*, with* $q_0, \dots, q_n \in Q$*, and* $\alpha_0, \dots, \alpha_n \in \Gamma^*$*, that satisfies the following conditions.*

- *It begins in an initial state* $q_0 \in Q_0$ *with the empty stack,* $\alpha_0 = \varepsilon$*.*
- *For each* $i \in \{1, \dots, n\}$*, with* $a_i = c \in \Sigma_0$*, there exists a clock constraint* φ_i *that is true on* w *at position* i*, with* $q_i \in \delta_c(q_{i-1}, \varphi_i)$ *and* $\alpha_i = \alpha_{i-1}$*.*
- *For each* $i \in \{1, \dots, n\}$*, with* $a_i = < \in \Sigma_{+1}$*, there exists a clock constraint* φ_i *that is true on* w *at position* i*, with* $(q_i, s) \in \delta_<(q_{i-1}, \varphi_i)$ *and* $\alpha_i = s\alpha_{i-1}$ *for some* $s \in \Gamma$*.*
- *For each* $i \in \{1, \dots, n\}$*, with* $a_i = > \in \Sigma_{-1}$*, if* $\alpha_{i-1} = s\beta$ *for some* $s \in \Gamma$ *and* $\beta \in \Gamma^*$*, then there exists a clock constraint* φ_i *that is true on* w *at position* i*, with* $q_i \in \delta_>(q_{i-1}, s, \varphi_i)$ *and* $\alpha_i = \beta$
- *For each* $i \in \{1, \dots, n\}$*, with* $a_i = > \in \Sigma_{-1}$*, if* $\alpha_{i-1} = \varepsilon$*, then there exists a clock constraint* φ_i *that is true on* w *at position* i*, with* $q_i \in \delta_>(q_{i-1}, \bot, \varphi_i)$ *and* $\alpha_i = \varepsilon$*.*
- *The computation ends in an accepting state* $q_n \in F$ *with any stack contents.*

The language recognized by $\mathcal{A}$*, denoted by* $L(\mathcal{A})$*, is the set of all timed strings, on which* $\mathcal{A}$ *has at least one accepting computation.*

Definition 4. *A nondeterministic event-clock input-driven pushdown automaton* $\mathcal{A} = (\Sigma_{+1}, \Sigma_0, \Sigma_{-1}, Q, Q_0, \Gamma, \langle \delta_a \rangle_{a \in \Sigma}, F)$ *is said to be deterministic if the following conditions hold.*

1. *There is a unique initial state:* $|Q_0| = 1$.
2. *Every transition function* δ_a*, with* $a \in \Sigma_0 \cup \Sigma_{+1}$*, satisfies* $|\delta_a(q, \varphi)| \leqslant 1$ *for all* $q \in Q$ *and* $\varphi \in \Phi(\Sigma)$*, and whenever* $\delta_a(q, \varphi)$ *and* $\delta_a(q, \varphi')$*, with* $\varphi \neq \varphi'$*, are both non-empty, the clock constraints* φ *and* φ' *cannot both be true at the same position of the same string.*
3. *Similarly, every transition function* $\delta_>$*, with* $> \in \Sigma_{-1}$*, satisfies* $|\delta_>(q, s, \varphi)| \leqslant 1$ *for all* $q \in Q$*,* $s \in \Gamma \cup \{\bot\}$ *and* $\varphi \in \Phi(\Sigma)$*, and whenever* $\delta_c(q, s, \varphi)$ *and* $\delta(q, s, \varphi')$*, with* $\varphi \neq \varphi'$*, are both non-empty, the clock constraints* φ *and* φ' *cannot both be true at the same position of the same string.*

The first result of this paper is that nondeterministic event-clock input-driven pushdown automata can be determinized. Determinization results for a very similar model were earlier given by Nguyen and Ogawa [19] and by Bhave et al. [8]. However, their constructions relied on the method of *region construction*, in which the space of clock values is discretized. On the other hand, the construction in the present paper has the benefit of being *direct*, in the sense that the transition function for a deterministic automaton directly simulates the transitions of a nondeterministic automaton. Later it will be proved that this easier construction is also optimal with respect to the number of states and stack symbols. The proposed construction is not much more difficult than the construction for standard input-driven pushdown automata, without time.

3 Direct determinization of event-clock IDPDA

The classical construction for determinizing a standard (untimed) input-driven pushdown automaton [10,6], is based upon considering a nondeterministic automaton's behaviour on a left bracket and on a matching right bracket at the same time, while reading the right bracket. This is achieved by computing a *behaviour relation* $R \subseteq Q \times Q$ of the original automaton inside brackets, and then using it to simulate these two moments in the computation at once. In this way, the stack symbol pushed while reading the left bracket is matched to the symbol popped while reading the right bracket, and all possible computations of this kind can be considered at once.

In the event-clock case, the nondeterministic decisions made on a left bracket are based upon the clock values at that time, and if the simulation of these decisions were deferred until reading the matching right bracket, then those clock values would no longer be available. Since event-clock automata cannot manipulate clock values explicitly, they, in particular, cannot push the clock values onto the stack for later use. What can be done is to *test all elementary clock constraints while reading the left bracket*, store their truth values in the stack, and later, upon reading the right bracket, use this information to simulate the behaviour of the original automaton on the left bracket. This idea is implemented in the following construction, which uses the same set of states as the classical construction [10,6], but requires more complicated stack symbols.

Theorem 1. *Let $\mathcal{A} = (\Sigma_{+1}, \Sigma_0, \Sigma_{-1}, Q, Q_0, \Gamma, \langle\delta_a\rangle_{a\in\Sigma}, F)$ be a nondeterministic event-clock input-driven pushdown automaton. Let Ψ be the set of atomic constraints used in its transitions. Then there exists a deterministic event-clock input-driven pushdown automaton with the set of states $Q' = 2^{Q\times Q}$, and with the pushdown alphabet $\Gamma' = 2^{Q\times Q} \times \Sigma_{+1} \times 2^{\Psi}$, which recognizes the same set of timed strings as $\mathcal{A}$.*

Proof. States of the deterministic automaton $\mathcal{B}$ are sets of pairs $(p, q) \in Q \times Q$, with each pair meaning that there is a computation of the original automaton $\mathcal{A}$ on the longest well-nested suffix of the input that begins in the state p and ends in the state q. The initial state of $\mathcal{B}$ is $q'_0 = \{\, (q_0, q_0) \mid q_0 \in Q_0 \,\}$.

For a **neutral symbol** $c \in \Sigma_0$ and a state $P \in Q'$, the transition $\delta'_c(P)$ advances all current computations traced in P by the next symbol c. Each computation continues by its own transition, which requires a certain clock constraint to be true. Whether each clock constraint $\varphi \in \Phi(\Sigma)$ is true or false, can be deduced from the truth assignment to the atomic constraints. For every set of atomic constraints $S \subseteq \Psi$, let $\xi_S = \bigwedge_{C\in S} C \wedge \bigwedge_{C\in\Psi\setminus S} \neg C$ be a clock constraint asserting that among all atomic constraints, exactly those belonging to S are true. Then, for every set S, the new automaton has the following transition.

$$\delta_c(P, \xi_S) = \{\, (p, q') \mid \exists (p, q) \in P,\ \exists \varphi \in \Phi(\Sigma) : q' \in \delta_c(q, \varphi),\ \varphi \text{ is true under } S \,\}$$

On a **left bracket** $< \in \Sigma_{+1}$, the transition of $\mathcal{B}$ in a state $P \in Q'$ pushes the current context of the simulation onto the stack, and starts the simulation afresh at the next level of brackets, where it will trace the computations beginning in different states $p' \in Q$. A computation in a state p' is started only if any computations of $\mathcal{A}$ actually reach that state. In addition, $\mathcal{B}$ pushes the current left bracket ($<$), as well as the truth value of all atomic constraints at the present moment, $S \subseteq \Psi$. This is done in the following transitions, defined for all $S \subseteq \Psi$.

$$\delta'_<(P, \xi_S) = \big(\big\{\, (p', p') \,\big|\, \exists (p, q) \in P,\ \exists \varphi \in \Phi(\Sigma) : \varphi \text{ is true under } S,\ p' \in \delta_<(q, \varphi)\big\},\ (P, <, S)\big)$$

If a matching right bracket ($>$) is eventually read, then $\mathcal{B}$ shall pop $(P, <, S)$ from the stack and reconstruct what has happened to each of the computations of $\mathcal{A}$ in P at this point and further on. On the other hand, if this left bracket ($<$) is unmatched, then the acceptance shall be determined on the basis of the computations traced on the inner level of brackets.

When $\mathcal{B}$ encounters a **matched right bracket** $> \in \Sigma_{-1}$ in a state $P' \subseteq Q \times Q$, it pops a stack symbol $(P, <, S) \in \Gamma'$ containing the matching left bracket ($< \in \Sigma_{+1}$), the data on all computations on the current level of brackets simulated up to that bracket ($P \subseteq Q \times Q$), and the truth value of all atomic clock constraints at the moment of reading that bracket ($S \subseteq \Psi$).

Then, each computation in P is continued by simulating the transition by the left bracket ($<$), the behaviour inside the brackets stored in P', and the transition by the right bracket ($>$), all at once. Let $u\langle v\rangle$ be the longest well-nested suffix of the string read so far. Every computation of $\mathcal{A}$ on u, which begins

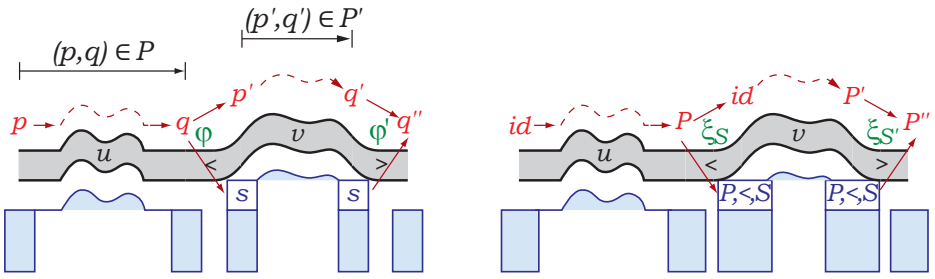

Fig. 2. (left) A computation of a nondeterministic event-clock IDPDA; (right) Its simulation by a deterministic event-clock IDPDA.

in a state p and ends in a state q, is represented by a pair (p, q). Upon reading the left bracket ($<$), the automaton $\mathcal{A}$ makes a transition to a state p', pushing a stack symbol s, along with checking a clock constraint φ. The automaton $\mathcal{B}$ can now check the same clock constraint by using the set S of atomic clock constraints that held true at the earlier left bracket ($<$). For every set of atomic constraints $S' \subseteq \Psi'$, the following transition is defined.

$$\begin{aligned}\delta'_{>}(P', (P, <, S), \xi_{S'}) = \{\, (p, q'') \mid{}& \\ \exists (p, q) \in P, \exists (p', q') \in P', \exists s \in \Gamma, \exists \varphi, \varphi' \in \Phi(\Sigma) : \varphi \text{ is true under } S,& \\ (p', s) \in \delta_{<}(q, \varphi),\ \varphi' \text{ is true under } S',\ q'' \in \delta_{>}(q', s, \varphi') \}&\end{aligned}$$

When $\mathcal{B}$ reads an **unmatched right bracket** $> \in \Sigma_{-1}$ while in a state $P \subseteq Q \times Q$, it continues the existing computations on the new bottom level of brackets.

$$\delta_{>}(P, \bot, \xi_S) = \{\, (p', p') \mid \exists (p, q) \in P,\ \exists \varphi : p' \in \delta_{>}(q, \bot, \varphi),\ \varphi \text{ is true under } S \,\}$$

The set of **accepting states** reflects all computations of $\mathcal{A}$ ending in an accepting state.

$$F' = \{\, P \subseteq Q \times Q \mid \text{there is a pair } (p, q) \text{ in } P, \text{ with } q \in F \,\}$$

A formal correctness claim for this construction reads as follows.

Claim. Let uvw be a timed string, where v is the longest well-nested suffix of uv, and let $P \subseteq Q \times Q$ be the state reached by $\mathcal{B}$ on uvw after reading uv. Then a pair (p, q) is in P if and only if there is a computation of $\mathcal{A}$ on uvw that passes through the state p right after reading u, and later, after reading the following v, enters the state q.

The claim can be proved by induction on the bracket structure of an input string.

Applying the claim to the whole input string shows that $\mathcal{B}$ accepts this string if and only if one of the computations of $\mathcal{A}$ on the same input string is accepting. □

It is interesting to note that the above determinization construction does not rely on the exact form of clock constraints: the resulting deterministic automaton checks only the constraints used by the original nondeterministic automaton, and only communicates the results through the stack in the form of Boolean values. Therefore, the same construction would apply verbatim for any kind of constraints on the pair (input string, current position) expressible in the model. In particular, the extended model of Bozzelli et al. [9] can be determinized in the same way.

Another thing worth mentioning is that for the particular set of clock constaints assumed in this paper, the determinization construction in Theorem 1 can be improved to eliminate all references to the stack prediction clock ($\overrightarrow{x_{\mathrm{pop}}}$), at the expense of using more states. This construction shall be presented in the upcoming full version of this paper.

4 A lower bound on the determinization complexity

The timed determinization construction in Theorem 1 produces 2^{n^2} states and 2^{n^2+k} stack symbols, where n is the number of states in the nondeterministic automaton and k is the number of atomic clock constraints. It shall now be proved that this construction is asymptotically optimal. The following theorem, proved in the rest of this section, is a timed extension of a result by Okhotin, Piao and Salomaa [15, Thm. 3.2].

Theorem 2. *For every n and for every k, there is an $O(n)$-state nondeterministic ECIDPDA over an alphabet of size $k + O(1)$, with nk stack symbols and k atomic constraints referring only to symbol history clocks, such that every deterministic ECIDPDA recognizing the same timed language must have at least 2^{n^2} states and at least $2^{n^2-O(n)+k}$ stack symbols.*

The automaton is defined over the following alphabet: $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$, $\Sigma_0 = \{a, b, c, \#\} \cup \{\, e_i \mid 1 \leqslant i \leqslant k \,\}$. For a set of pairs $R = \{(i_1, j_1), \ldots, (i_\ell, j_\ell)\} \subseteq \{1, \ldots, n\}^2$, let $u_R \in \{a, b, \#\}^*$ be the string that lists all pairs in R in the lexicographical order, under the following encoding.

$$u_R = \#ab\#a^{i_1}b^{j_1}\,\#a^{i_2}b^{j_2} \ldots \#a^{i_\ell}b^{j_\ell}\#ab$$

For every set of symbols $X = \{e_{i_1}, \ldots, e_{i_\ell}\} \subseteq \{e_1, \ldots, e_k\}$, let $v_X = e_1 \ldots e_k e_{i_1} \ldots e_{i_\ell}$ be the string that first lists all the symbols in $\{e_1, \ldots, e_k\}$, and then only the symbols in X.

Now, let $m \geqslant 1$ be the number of levels in the string to be constructed, let $s_1, \ldots, s_m, s_{m+1} \in \{1, \ldots, n\}$ be numbers, let $R_1, \ldots, R_m \subseteq \{1, \ldots, n\}^2$ be relations, and let $X_1, Y_1, \ldots, X_m, Y_m \subseteq \{e_1, \ldots, e_k\}$ be $2m$ sets of symbols. This information is encoded in the following string.

$$w = \underbrace{v_{X_1}{<}u_{R_1}v_{X_2}{<}u_{R_2} \ldots v_{X_m}{<}u_{R_m}}_{w_1} \underbrace{c^{s_{m+1}}v_{Y_m}{>}c^{s_m} \ldots v_{Y_2}{>}c^{s_2}v_{Y_1}{>}c^{s_1}}_{w_2}$$

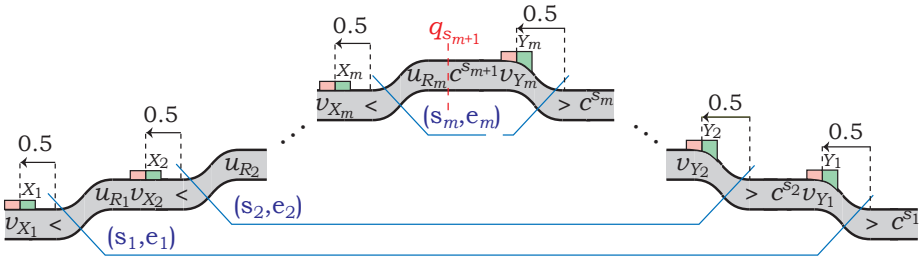

Fig. 3. A nondeterministic event-clock IDPDA checking the validity of a well-formed string.

The string is made timed by saying that the duration of each named substring is 1 time unit, and in each substring v_{X_i}, its first k symbols occur more than 0.5 time units earlier than the subsequent left bracket ($<$), whereas its remaining symbols representing the elements of X_i occur less than 0.5 time units earlier than the left bracket. Similarly, in each string v_{Y_i}, its first k symbols occur more than 0.5 time units earlier than the next right bracket ($>$), while its remaining symbols occur less than 0.5 time units earlier than the bracket. This allows an event-clock automaton to *see* the set X_i using clock constraints while reading the left bracket ($<$), and to see Y_i while at the right bracket ($>$). For the clock constraints not to see anything else, the first k symbols of v_X, and the first and the last three symbols u_R, occur at predefined time independent of X and R.

A timed string is said to be *well-formed* if it is defined as above, for some m, s_i, R_i, X_i and Y_i. A well-formed string is *valid*, if $(s_i, s_{i+1}) \in R_i$ and $X_i \cap Y_i \neq \varnothing$ for each i.

Lemma 1. *For every n and k, there exists a nondeterministic ECIDPDA using $O(n)$ states, nk stack symbols and k clock constraints, which accepts every valid well-formed string and does not accept any invalid well-formed string.*

Proof (a sketch). The automaton operates as in Figure 3. At the left bracket following each v_{X_i}, it guesses s_i and e_i, using a clock constraint $\overleftarrow{x_{e_i}} < 1$ to check that $e_i \in X_i$, pushes the pair (s_i, e_i) and remembers s_i in its state. While reading u_{R_i}, it guesses any s_{i+1} with $(s_i, s_{i+1}) \in R_i$ and remembers s_{i+1} in its state. On each $c^{s_{i+1}}$, the automaton checks that the current state is s_{i+1} and forgets its value. On the following right bracket, the automaton pops the pair (s_i, e_i), verifies that $e_i \in Y_i$ using a constraint $\overleftarrow{x_{e_i}} < 1$, and keeps s_i in its current state. □

Lemma 2. *For every n and k, every deterministic ECIDPDA that accepts every valid well-formed string and does not accept any invalid well-formed string must have at least 2^{n^2} states.*

Although the bound is the same as in the untimed case [15], an event-clock automaton could potentially use its clocks to reduce the number of states. Still,

it is proved that on a string $v_{\{e_1\}}{<}u_R c^t v_{\{e_1\}}{>}c^s$, after reading u_R, a deterministic automaton must remember the entire relation R in its internal state, for otherwise it would not be able to check whether the pair (s,t) is in R, as no information on R could be obtained using any clock constraints.

Lemma 3. *For every n and k, every deterministic ECIDPDA that accepts every valid well-formed string and does not accept any invalid well-formed string must have at least $2^{n^2-o(1)+k}$ stack symbols.*

Proof. The proof is modelled on the proof by Okhotin, Piao and Salomaa [15, Lemma 3.4], with the clock constraints added. The argument uses binary relations that are both left-total and right-total: that is, relations $R \subseteq \{1,\ldots,n\}^2$ in which, for every $x \in \{1,\ldots,n\}$, there is an element y with $(x,y) \in R$, and, symmetrically, for every y, there is an element x with $(x,y) \in R$. There are at least $2^{n^2} - 2n \cdot 2^{n(n-1)} = 2^{n^2-O(n)}$ such relations,

Fix the number of levels $m \geqslant 1$, let $R_1,\ldots,R_m \subseteq \{1,\ldots,n\}^2$ be left- and right-total relations, and let $X_1,\ldots,X_m \subseteq \{e_1,\ldots,e_k\}$ be non-empty sets of symbols. These parameters define the first part w_1 of a well-formed string. It is claimed that, after reading w_1, a deterministic automaton somehow has to store all relations $R_1,\ldots,R_m$ and all sets $X_1,\ldots,X_m$ in the available memory: that is, in m stack symbols and in one internal state.

Suppose that, for some $R_1,\ldots,R_m,R'_1,\ldots,R'_m \subseteq \{1,\ldots,n\}^2$ and $X_1,\ldots,X_m,X'_1,\ldots,X'_m \subseteq \{e_1,\ldots,e_k\}$, with $(R_1,\ldots,R_m,X_1,\ldots,X_m) \neq (R'_1,\ldots,R'_m,X'_1,\ldots,X'_m)$, the automaton, after reading the corresponding first parts w_1 and w'_1, comes to the same state with the same stack contents.

$$w_1 = v_{X_1}{<}u_{R_1} v_{X_2}{<}u_{R_2} \ldots v_{X_m}{<}u_{R_m}$$
$$w'_1 = v_{X'_1}{<}u_{R'_1} v_{X'_2}{<}u_{R'_2} \ldots v_{X'_m}{<}u_{R'_m}$$

First, as in the argument by Okhotin, Piao and Salomaa [15, Lemma 3.4], assume that these parameters differ in an i-th relation, with $(s,t) \in R_i \setminus R'_i$. Let $s_i = s$. Since all relations $R_{i-1},\ldots,R_1$ are right-total, there exists a sequence of numbers $s_{i-1},\ldots,s_1$, with $(s_j,s_{j+1}) \in R_j$ for all $j \in \{1,\ldots,i-1\}$. Similarly, let $s_{i+1} = t$. Since the relations $R_{i+1},\ldots,R_m$ are left-total, there is a sequence $s_{i+2},\ldots,s_{m+1}$, with $(s_j,s_{j+1}) \in R_j$ for all $j \in \{i+1,\ldots,m\}$. Construct the following continuation for w_1 and w'_1.

$$w_2 = c^{s_{m+1}} v_{X_m}{>}c^{s_m} \ldots v_{X_2}{>}c^{s_2} v_{X_1}{>}c^{s_1}$$

The concatenation $w_1 w_2$ is then well-formed and valid, whereas the concatenation $w'_1 w_2$ is well-formed and invalid, because $(s_i,s_{i+1}) \notin R'_i$. But, while reading w_2, the automaton cannot tell w_1 from w'_1 using history clocks, and thus the automaton either accepts both concatenations or rejects both of them, which is a contradiction.

Now assume that the prefixes w_1 and w'_1 use the same relations $R_1,\ldots,R_m$ and differ in an i-th set, with $e \in X_i \setminus X'_i$. Since all relations are left-total, there exists a sequence of numbers $s_1,\ldots,s_m,s_{m+1}$, with $(s_j,s_{j+1}) \in R_j = R'_j$ for all

$j \in \{1, \ldots, m\}$. This time, the continuation includes the sequence of numbers and takes all sets X_j from w_1, except for X_i, which is replaced by $\{e\}$.

$$w_2 = c^{s_{m+1}} v_{X_m} > c^{s_m} \ldots v_{X_{i+1}} > c^{s_{i+1}} v_{\{e\}} > c^{s_i} v_{X_{i-1}} > c^{s_{i-1}} \ldots v_{X_1} > c^{s_1}$$

Then, both concatenations $w_1 w_2$ and $w_1' w_2$ are well-formed. However, the concatenation $w_1 w_2$ is valid, whereas $w_1' w_2$ is invalid, because $X_i' \cap \{e\} = \varnothing$. Hence the automaton again either accepts or rejects both strings, and a contradiction is obtained.

This shows that, for each $m \geqslant 1$, the automaton must be able to reach at least $(2^{n^2} - 2n \cdot 2^{n(n-1)})^m (2^k - 1)^m$ distinct configurations after reading different strings of the given form. Then, $|\Gamma|^m \cdot |Q|$ cannot be less than this number, and for m large enough this inequality holds only if $|\Gamma| \geqslant 2^{n^2 - o(1) + k}$. □

The proof of Theorem 2 follows from Lemmata 1–3. It implies that the determinization construction in Theorem 1 is asymptotically optimal both with respect to the number of states and to the number of stack symbols.

Acknowledgement The authors are grateful to the anonymous reviewers for pointing out numerous shortcomings of the original submission.

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012. pp. 35–44. IEEE Computer Society (2012). https://doi.org/10.1109/LICS.2012.15
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). https://doi.org/10.1016/0304-3975(94)90010-8
3. Alur, R., Fix, L., Henzinger, T.A.: Event-clock automata: A determinizable class of timed automata. Theor. Comput. Sci. **211**(1-2), 253–273 (1999). https://doi.org/10.1016/S0304-3975(97)00173-4
4. Alur, R., Madhusudan, P.: Decision problems for timed automata: A survey. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures. LNCS, vol. 3185, pp. 1–24. Springer (2004). https://doi.org/10.1007/978-3-540-30080-9_1
5. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Babai, L. (ed.) Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004. pp. 202–211. ACM (2004). https://doi.org/10.1145/1007352.1007390
6. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM **56**(3), 16:1–16:43 (2009). https://doi.org/10.1145/1516512.1516518
7. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a tool suite for automatic verification of real-time systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA. LNCS, vol. 1066, pp. 232–243. Springer (1995). https://doi.org/10.1007/BFb0020949

8. Bhave, D., Dave, V., Krishna, S.N., Phawade, R., Trivedi, A.: A logical characterization for dense-time visibly pushdown automata. In: Dediu, A., Janousek, J., Martín-Vide, C., Truthe, B. (eds.) Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings. LNCS, vol. 9618, pp. 89–101. Springer (2016). https://doi.org/10.1007/978-3-319-30000-9_7
9. Bozzelli, L., Murano, A., Peron, A.: Event-clock nested automata. In: Klein, S.T., Martín-Vide, C., Shapira, D. (eds.) Language and Automata Theory and Applications - 12th International Conference, LATA 2018, Ramat Gan, Israel, April 9-11, 2018, Proceedings. LNCS, vol. 10792, pp. 80–92. Springer (2018). https://doi.org/10.1007/978-3-319-77313-1_6
10. von Braunmühl, B., Verbeek, R.: Input driven languages are recognized in log n space. In: Karplnski, M., van Leeuwen, J. (eds.) Topics in the Theory of Computation, North-Holland Mathematics Studies, vol. 102, pp. 1–19. North-Holland (1985). https://doi.org/10.1016/S0304-0208(08)73072-X
11. Clemente, L., Lasota, S.: Timed pushdown automata revisited. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015. pp. 738–749. IEEE Computer Society (2015). https://doi.org/10.1109/LICS.2015.73
12. Geeraerts, G., Raskin, J., Sznajder, N.: On regions and zones for event-clock automata. Formal Methods Syst. Des. **45**(3), 330–380 (2014). https://doi.org/10.1007/s10703-014-0212-1
13. Löding, C., Madhusudan, P., Serre, O.: Visibly pushdown games. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings. LNCS, vol. 3328, pp. 408–420. Springer (2004). https://doi.org/10.1007/978-3-540-30538-5_34
14. Mehlhorn, K.: Pebbling moutain ranges and its application of dcfl-recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings. LNCS, vol. 85, pp. 422–435. Springer (1980). https://doi.org/10.1007/3-540-10003-2_89
15. Okhotin, A., Piao, X., Salomaa, K.: Descriptional complexity of input-driven pushdown automata. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) Languages Alive - Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday. LNCS, vol. 7300, pp. 186–206. Springer (2012). https://doi.org/10.1007/978-3-642-31644-9_13
16. Okhotin, A., Salomaa, K.: Complexity of input-driven pushdown automata. SIGACT News **45**(2), 47–67 (2014). https://doi.org/10.1145/2636805.2636821
17. Okhotin, A., Selivanov, V.L.: Input-driven pushdown automata on well-nested infinite strings. In: Santhanam, R., Musatov, D. (eds.) Computer Science - Theory and Applications - 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 - July 2, 2021, Proceedings. LNCS, vol. 12730, pp. 349–360. Springer (2021). https://doi.org/10.1007/978-3-030-79416-3_21
18. Ouaknine, J., Worrell, J.: On the language inclusion problem for timed automata: Closing a decidability gap. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings. pp. 54–63. IEEE Computer Society (2004). https://doi.org/10.1109/LICS.2004.1319600
19. Tang, N.V., Ogawa, M.: Event-clock visibly pushdown automata. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current

Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings. LNCS, vol. 5404, pp. 558–569. Springer (2009). https://doi.org/10.1007/978-3-540-95891-8_50