A Polynomial-Time Algorithm for Checking the Inclusion of Deterministic Restricted One-Counter Transducers Which Accept by Final State

Mitsuo Wakatsuki
Graduate School of Informatics and Engineering, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
wakatsuki.mitsuo@uec.ac.jp

Etsuji Tomita
The Advanced Algorithms Research Laboratory, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
tomita@ice.uec.ac.jp

Tetsuro Nishino
Graduate School of Informatics and Engineering, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
nishino@uec.ac.jp

Abstract

This paper is concerned with a subclass of deterministic pushdown transducers, called deterministic restricted one-counter transducers (droct’s), and studies the inclusion problem for droct’s which accept by final state. In the previous study, we presented a polynomial-time algorithm for checking the equivalence of these droct’s. By extending this technique, we present a polynomial-time algorithm for checking the inclusion of these droct’s.

keywords: inclusion problem, deterministic pushdown transducer, deterministic restricted one-counter transducer, polynomial-time algorithm, formal language theory

1 Introduction

One of the most interesting questions in formal language theory is the equivalence problem for deterministic pushdown automata (dpda’s) and the corresponding deterministic context-free grammars. The inclusion problem for dpda’s is also as important as the equivalence problem for dpda’s. Although Sénizergues [3] has proved that the equivalence problem for any pair of dpda’s is decidable, his algorithm is very complicated and is hard to be implemented. A checking the equivalence for some dpda’s can play an important role in the learning process for these dpda’s [4]. From a practical point of view, it is desirable to have a polynomial-time algorithm for checking the equivalence or the inclusion.

A deterministic one-counter automaton (doca) is a dpda having only one stack symbol, with the exception of a bottom-of-stack marker. A deterministic restricted one-counter automaton (droca) is a dpda which has only one stack symbol. The class of languages accepted by droca’s is a proper subclass of that of languages accepted by doca’s. Moreover, the class of languages accepted by droca’s which accept by final state properly contains the class of regular languages. Valiant has shown that the equivalence problem for doca’s is decidable in single exponential time [6][7] and the inclusion problem for doca’s is undecidable [6]. On the other hand, Higuchi et al. have presented polynomial-time algorithms for checking the inclusion (also the equivalence) for droca’s which accept by empty stack [1] and for real-time droca’s (i.e. droca’s without ε-moves) which accept by final state [2].

A deterministic pushdown transducer (dpdt) is a dpda provided with outputs. The inclusion problem (also the equivalence problem) for dpdt’s is essentially more difficult than for dpda’s. In this paper, we are concerned with a subclass of dpdt’s called deterministic restricted one-counter transducers (droct’s), which are droca’s provided with outputs, and study the inclusion problem for non-real-time droct’s (i.e. droct’s with possible ε-moves) which accept by final state. Since these droct’s may have infinite sequences of ε-moves, it is possible that their stack heights increase infinitely without reading inputs. In the previous study, we presented polynomial-time algorithms for checking the equivalence of a pair of real-time droct’s [8] and of a pair of non-real-time droct’s [9]. By extending the technique in Ref. [9], we present a new direct branching algorithm for checking the inclusion of non-real-time droct’s. The worst-case time complexity of our algorithm is polynomial in the description length of these droct’s.

The result can be accomplished by only slightly
modifying the algorithm of Ref. [9]. So, this paper is thoroughly based upon Ref. [9] and it only describes the new definitions with relevant basic properties and necessary changes.

2 Basic Properties and Propositions

A deterministic pushdown transducer (dpdt for short) which accepts by final state is denoted by $T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F)$, where $Q$, $\Gamma$, $\Sigma$, $\Delta$, $\mu$ are the finite sets of states, stack symbols, input symbols, output symbols, and transition-output rules respectively, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the initial stack symbol, and $F \subseteq Q$ is the set of final states. A dpdt $T$ is said to be a deterministic restricted one-counter transducer (droct for short) if $\Gamma = \{Z_0\}$.

A configuration $(p, \alpha)$ of the dpdt $T$ is said to be in $\varepsilon$-R/W mode if $\alpha = A\alpha'' \in \Gamma^+$ and $(p, A)^{\varepsilon/z}_{T} (q, \theta) \in \mu$.

**Definition 1** Let $(p, \alpha)$ be a configuration of a dpdt $T_1$ which accepts by final state, $(\bar{p}, \beta)$ be that of a dpdt $T_2$ which accepts by final state, and $h \in \Delta^\pm$.

If $\text{TRANS}(p, \alpha) \subseteq h \text{TRANS}(\bar{p}, \beta) = \{x/\nu \mid x/\nu \in \text{TRANS}(\bar{p}, \beta)\}$, then it is written as $(p, \alpha) \subseteq h(\bar{p}, \beta)$.

Here, if $h \in \Delta^\pm$, then $(p, \alpha) \subseteq h(\bar{p}, \beta)$ is the same as $k(p, \alpha) \subseteq h(\bar{p}, \beta)$ with $k = h^{-1} \in \Delta^\ast$ which stands for $\{x/\nu \mid x/\nu \in \text{TRANS}(p, \alpha)\} \subseteq \text{TRANS}(\bar{p}, \beta)$. Such a formula as $(p, \alpha) \subseteq h(\bar{p}, \beta)$, $h \in \Delta^\pm$ or $k(p, \alpha) \subseteq h(\bar{p}, \beta)$, $k \in \Delta^\ast$, is called an inclusion equation.

If $\text{TRANS}(T_1) \subseteq \text{TRANS}(T_2)$, then it is written as $T_1 \subseteq T_2$. Otherwise, $T_1 \not\subseteq T_2$.

For a droct $T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F)$, define $\rho = \max\{|\theta| \mid (p, A)^{\varepsilon/z}_{T} (q, \theta) \in \mu\}$. Without loss of generality, we may assume that $\rho \leq 2$, i.e. stack height increases by at most one per move.

**Lemma 1** Let $T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F)$ be a droct which accepts by final state, and $(p, \alpha) \in Q \times \Gamma^\ast$ be a configuration of $T$. Then, the following (i) - (iii) hold.

(i) If $N(p, \alpha) \neq \emptyset$ with $|\alpha| \geq |Q|$, then $N(p, \beta) \neq \emptyset$ for any $\beta \in \Gamma^\ast$.

(ii) For any $\beta \in \Gamma^\ast$, it holds that (a) $L(p, \alpha) \subseteq L(p, \alpha \beta)$ and (b) $\text{TRANS}(p, \alpha) \subseteq \text{TRANS}(p, \alpha \beta)$.

(iii) If $N(p, \alpha) = \emptyset$, then for any $\beta \in \Gamma^\ast$, it holds that (a) $N(p, \alpha \beta) = \emptyset$, (b) $L(p, \alpha) = L(p, \alpha \beta)$ and (c) $\text{TRANS}(p, \alpha) = \text{TRANS}(p, \alpha \beta)$.

**Lemma 2** Let $T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F)$ be a droct which accepts by final state, and $(p, \alpha) \in Q \times \Gamma^\ast$ be a configuration of $T$. If there exists a derivation such that $(p, \alpha)^{x/\nu}_{T} (q, \beta)$ for some $x \in \Sigma^\ast$, $y \in \Delta^\ast$, $(q, \beta) \in Q \times \Gamma^\ast$, then there exists a derivation such that $(p, \alpha)^{x'/\nu'}_{T} (q, \beta')$ with $n \leq |Q|(|Q| - 1)$ for some $x' \in \Sigma^\ast$, $y' \in \Delta^\ast$, $\beta' \in \Gamma^\ast$.

From Lemma 1 (i) and (iii)(a), we can construct a polynomial-time checking procedure whether any configuration $(p, \alpha) \in Q \times \Gamma^\ast$ of a given droct $T$ is $N(p, \alpha) \neq \emptyset$ or not. From Lemma 2, we can construct a polynomial-time checking procedure whether any configuration $(p, \alpha) \in Q \times \Gamma^\ast$ of $T$ is live or not.

We shall check the inclusion of two droct’s $T_i = (Q_i, \Gamma_i, \Sigma_i, \Delta_i, \mu_i, q_{0i}, Z_{0i}, F_i) \ (i = 1, 2)$ which accept by final state, whose associated drocas’s are $M_i$ respectively. We are only concerned with the case where $L(T_i) \neq \emptyset \ (i = 1, 2)$, $\Sigma = \Sigma_1 \subseteq \Sigma_2$, and $\Delta = \Delta_1 \subseteq \Delta_2$.

We give the most elementary proposition concerning outputs.

**Proposition 1** (i) Suppose $T_1 \subseteq T_2$ holds and

$$(q_{01}, Z_{01})^{w/w_1}_{T_1}F(p, \alpha) \text{ with } L(p, \alpha) \neq \emptyset, \quad (1)$$

$$(q_{02}, Z_{02})^{w/w_2}_{T_2}F(\bar{p}, \beta) \text{ with } L(\bar{p}, \beta) \neq \emptyset, \quad (2)$$

for some $w \in \Sigma^\ast$, $w_1, w_2 \in \Delta^\ast$, $p \in Q_1$, $\bar{p} \in Q_2$, $\alpha \in \Gamma_1^\ast$ and $\beta \in \Gamma_2^\ast$, then it holds that

$w_1h = w_2 \text{ for some } h \in \Delta^\pm$ \quad (3)

and

$$(p, \alpha) \subseteq h(\bar{p}, \beta). \quad (4)$$

(ii) If we have, in addition to (i), another pair of derivations $(q_{01}, Z_{01})^{w'/w'_1}_{T_1}F(p, \alpha')$ with $L(p, \alpha') \neq \emptyset$ and $(q_{02}, Z_{02})^{w'/w'_2}_{T_2}F(\bar{p}, \beta')$ with $L(\bar{p}, \beta') \neq \emptyset$ for some $w' \in \Sigma^\ast$, $w'_1, w'_2 \in \Delta^\ast$ such that $w'_1h' = w'_2$ for some $h' \in \Delta^\pm$, $\alpha' \in \Gamma_1^\ast$, $\beta' \in \Gamma_2^\ast$, and hence $(p, \alpha') \subseteq h'(\bar{p}, \beta')$, then $h = h'$.

**Proof:** These properties can be easily derived from Definitions 5, 6 in Ref. [9], Definition 1, and Lemma 1 (ii).

The above $h \in \Delta^\pm$ in eq.(3) and eq.(4) is to compensate the difference between the two outputs, and is called an output compensating part. From Proposition 1 (ii), the output compensating part is unique for each inclusion equation as eq.(4) when $T_1 \subseteq T_2$.
3 The Inclusion Checking Algorithm

The inclusion checking is carried out by developing step by step a so-called comparison tree as in Ref. [5].

At the initial stage, the comparison tree contains only the root labeled \((q_{01}, Z_{01}) \subseteq (q_{02}, Z_{02})\) which is said to be in unchecked status. In each step, the algorithm considers a node labeled \((p, \alpha) \subseteq h(\bar{p}, \beta)\) such that eq.(1) and eq.(2) with eq.(3) for some \(w \in \Sigma^*, w_1, w_2 \in \Delta^*\), and tries to prove or disprove this inclusion. In the case where \(\text{FIRST}_{\text{live}}(p, \alpha) = \{\varepsilon\} \subseteq \text{FIRST}_{\text{live}}(\bar{p}, \beta)\) and \((p, \alpha)\) is not in \(\varepsilon\)-mode, if \(h = \varepsilon\) then we turn the above node to be in checked status. Otherwise, we expand it by branching or stack reduction (See Ref. [9] for stack reduction).

3.1 Branching

Branching is a basic step of developing the comparison tree.

Lemma 3  The inclusion equation eq.(4) holds iff the following conditions (i), (ii) and (iii) hold.

(i) \(\text{FIRST}_{\text{live}}(p, \alpha) \subseteq \text{FIRST}_{\text{live}}(\bar{p}, \beta)\).
(ii) \((p, \alpha)\) and \((\bar{p}, \beta)\) are in reading mode, for each \(a_i \in \text{FIRST}_{\text{live}}(p, \alpha) - \{\varepsilon\} = \{a_1, a_2, \ldots, a_l\} \subseteq \Sigma\) \((i = 1, 2, \ldots, l)\) let

\[
(p, \alpha) \xrightarrow{a_i/z_i} (p_1, \alpha_i) \quad \text{and} \quad (\bar{p}, \beta) \xrightarrow{a_i/z_i} (\bar{p}_1, \beta_i)
\]

for some \((p_i, \alpha_i) \in Q_1 \times \Gamma_i^*\), \((\bar{p}_i, \beta_i) \in Q_2 \times \Gamma_i^*\), \(z_i, \bar{z}_i \in \Delta^*\). Then it holds that \(z_i h_1 = h \bar{z}_i\) for some \(h_i \in \Delta^+, i = 1, 2, \ldots, l\).

(b) In case \((p, \alpha)\) is in \(\varepsilon\)-mode and \((\bar{p}, \beta)\) is not in \(\varepsilon\)-mode, let \(a_1 = \varepsilon\), \(l = 1\),

\[
(p, \alpha) \xrightarrow{\varepsilon/z_1} (p_1, \alpha_1) \quad \text{and} \quad (\bar{p}, \beta) = (\bar{p}_1, \beta_1).
\]

Then it holds that \(z_1 h_1 = h\).

(c) In case \((p, \alpha)\) is not in \(\varepsilon\)-mode and \((\bar{p}, \beta)\) is in \(\varepsilon\)-mode, let \(a_1 = \varepsilon\), \(l = 1\),

\[
(p, \alpha) = (p_1, \alpha_1) \quad \text{and} \quad (\bar{p}, \beta) \xrightarrow{\varepsilon/z_1} (\bar{p}_1, \beta_1).
\]

Then it holds that \(h_1 = h \bar{z}_1\).

(d) In case both \((p, \alpha)\) and \((\bar{p}, \beta)\) are in \(\varepsilon\)-mode, let \(a_1 = \varepsilon\), \(l = 1\). (d.1) In case \((p, \alpha)\) is in \(\varepsilon\)-R/W mode or \(p \notin F_1\), let

\[
(p, \alpha) \xrightarrow{\varepsilon/z_1} (p_1, \alpha_1) \quad \text{and} \quad (\bar{p}, \beta) = (\bar{p}_1, \beta_1).
\]

Then it holds that \(z_1 h_1 = h\).

(d.2) In case \((p, \alpha)\) is not in \(\varepsilon\)-R/W mode with \(p \in F_1\) and \((\bar{p}, \beta)\) is in \(\varepsilon\)-R/W mode or \(\bar{p} \notin F_2\), let

\[
(p, \alpha) = (p_1, \alpha_1) \quad \text{and} \quad (\bar{p}, \beta) \xrightarrow{\varepsilon/z_1} (\bar{p}_1, \beta_1).
\]

Then it holds that \(h_1 = h \bar{z}_1\).

(d.3) In case \((p, \alpha)\) is not in \(\varepsilon\)-R/W mode with \(p \in F_1\) and \((\bar{p}, \beta)\) is not in \(\varepsilon\)-R/W mode with \(\bar{p} \in F_2\), let

\[
(p, \alpha) \xrightarrow{\varepsilon/z_1} (p_1, \alpha_1) \quad \text{and} \quad (\bar{p}, \beta) \xrightarrow{\varepsilon/z_1} (\bar{p}_1, \beta_1)
\]

Then it holds that \(z_1 h_1 = h \bar{z}_1\).

(iii) Concerning the above condition (ii), it holds that \((p_i, \alpha_i) \subseteq h_i(\bar{p}_i, \beta_i), i = 1, 2, \ldots, l\).

Proof: It follows from Proposition 1 (i).

The checking whether conditions (i) and (ii) in Lemma 3 hold or not is named output branch checking to the node labeled eq.(4) in question. When it is verified to hold, the checking is said to be successful. Then we expand the above node to have \(l\) sons in unchecked status labeled by (iii), with edges labeled \(z_i \backslash a_i \bar{z}_i, i = 1, 2, \ldots, l\), and we turn the node in question to be checked. The step of developing the comparison tree in this way is named branching to the node in question. If condition (i) or (ii) does not hold, conclude that \(T_1 \not\subseteq T_2\).

When the branching has been applied to the node in question, the number of sons of it is at most \(|\Sigma|\).

3.2 Skipping

In order to prevent the comparison tree from growing larger and larger infinitely by successive application of branching or stack reduction steps, certain nodes are expanded by other steps of skipping. The following skipping step is almost the same as in Ref. [9]. So, see Ref. [9] for the details.

Definition 2  Suppose that a node in question is labeled

\[
(p, \omega_1 \alpha'') \subseteq h(\bar{p}, \omega_2 \beta''),
\]

where \(\alpha = \omega_1 \alpha''\) and \(\beta = \omega_2 \beta''\) with \(\alpha'' \neq \varepsilon\) or \(\beta'' \neq \varepsilon\). We say that the prerequisite for skipping to it is satisfied if the tree \(T(T_1 : T_2)\) contains a branching node labeled

\[
(p, \omega_1) \subseteq h(\bar{p}, \omega_2)
\]

where \(\omega_1 \in \Gamma_1^+\) and \(\omega_2 \in \Gamma_2^+(6)\).
such that \( N(p, \omega_1) \neq \emptyset \) with \( |\omega_1| \geq |Q_1| \), or \( N(p, \omega_2) \neq \emptyset \) with \( |\omega_2| \geq |Q_2| \).

**Definition 3** Suppose that the prerequisite for skipping to the node labeled eq.(5) in question is satisfied as in Definition 2, and that

\[
(p, \omega_1) \subseteq h(p, \omega_2)\frac{u|x/v}{T(T_1, T_2)}((q, \gamma) \subseteq (q', \gamma'))
\]

and \( \text{FIRST}_{\text{live}}(q, \gamma) = \{ \varepsilon \} \subseteq \text{FIRST}_{\text{live}}(q', \gamma') \), where \( \gamma = \varepsilon \) or \( N(q, \gamma) = \emptyset \) with \( 1 \leq |\gamma| \leq |Q_1| \), and \( \gamma = \varepsilon \) or \( N(q', \gamma) = \emptyset \) with \( 1 \leq |\gamma| \leq |Q_2| \), for some \( x \in (\Sigma \cup \{ \lambda \}^*) \), \( u, v \in \Delta^* \) with \( u = hv, q \in F_1, q' \in F_2 \).

Now find a shortest string \( \sigma(x_0) \in \Sigma^* \) such that

\[
(p, \omega_1) \frac{u|x_0/v_0}{T(T_1, T_2)}(q, \zeta) \quad \text{and} \quad (p, \omega_2) \frac{v|x_0/v_0}{T(T_1, T_2)}(q', \zeta')
\]

with \( |\zeta| \geq |\gamma| \) and \( |\zeta'| \geq |\gamma'| \), for some \( u_0, v_0 \in \Delta^* \), and check whether it is successful or not to have \( u_0 = hv_0 \). Then the skipping to the node in question is said to be applicable if the above checking is successful for every possible \( (q, \gamma) \subseteq (q', \gamma') \) as above. A node labeled \( (q, \gamma \alpha''') \subseteq (q', \gamma'') \) is defined to be a skipping-end from the node in question, and an edge label between them is defined to be \( u_0|x_0/v_0 \).

When skipping is applicable to the node in question, we expand it to have skipping-ends in unchecked status, then we turn the node in question to be skipping. The step of developing the comparison tree in this way is named skipping to the node labeled eq.(5) with respect to eq.(6). Whenever a new node is added to the comparison tree, we must apply skipping again to the node which has been already applied skipping, because it is possible that some new skipping-ends will be added to it afterward.

### 3.3 Halting

Consider a skipping node labeled \( (p, \alpha) \subseteq h(p, \beta) \), where \( \alpha = \omega_1 \alpha'' \) and \( \beta = \omega_2 \beta'' \) with \( \alpha'' \neq \varepsilon \) or \( \beta'' \neq \varepsilon \), which has a skipping-end labeled

\[
(q, \gamma) \subseteq (q', \gamma_0)
\]

such that

\[
(p, \alpha) \frac{u|x_0/v_0}{T(T_1, T_2)}((q, \gamma) \subseteq (q', \gamma_0))
\]

for some \( x_0 \in (\Sigma \cup \{ \lambda \}^*) \), \( u_0, v_0 \in \Delta^* \) with \( u_0 = hv_0, q \in F_1, q' \in F_2 \). Now suppose that it has been applied skipping steps sufficiently many times, and that another brother skipping-end labeled

\[
(q, \gamma) \subseteq (q', \gamma)
\]

such that

\[
(p, \alpha) \frac{u|x_0/v_0}{T(T_1, T_2)}((q, \gamma) \subseteq (q', \gamma))
\]

for some \( x \in (\Sigma \cup \{ \lambda \}^*) \), \( u, v \in \Delta^* \) with \( u = hv, \gamma \neq \gamma_0 \), has just been constructed in the comparison tree. In addition, assume that they satisfy

\[
|\gamma_0| < |\gamma|.
\]

Here, both inclusion equations eq.(7) and eq.(9) should be checked to hold for \( (p, \alpha) \subseteq h(p, \beta) \) to hold. However, \( (q', \gamma_0) \subseteq (q, \gamma) \) holds from Lemma 1 (ii). Hence, it suffices to check only the former eq.(7). Then the latter skipping-end labeled eq.(9) is turned to be in newly introduced halting status, and it will not be expanded any more. Nodes which are not in halting status are said to be nonhalting.

**Definition 4** A skipping-end labeled eq.(9) with eq.(10) is said to satisfy the halting condition if it can find a nonhalting brother skipping-end labeled eq.(7) with eq.(8) such that eq.(11) holds.

When the skipping has been applied to the node in question, the number of skipping-ends of it is at most \(|F_1||F_2|(|Q_1||Q_2| + 1)^2\).

### 3.4 The Whole Algorithm

The whole algorithm is shown in Figure 1. Here, the next node to be visited is chosen as the “smallest” of the unchecked or skipping nodes, where the size of a node labeled \( (p, \alpha) \subseteq h(p, \beta) \) is the pair \((\max\{|\alpha|, |\beta|\}, \min\{|\alpha|, |\beta|\})\), under lexicographic ordering.

**Example 1** Let us apply our algorithm to the following pair of droct’s: \( T_1 = (\{p_0, p_1, p_2, p_3\}, \{A\}, \{a, b, c\}, \{a, b, \mu_1, p_0, A, \{p_2, p_3\}\} \) and \( T_2 = (\{q_0, q_1, q_2, q_3\}, \{B\}, \{a, b, c\}, \{a, b, \mu_2, q_0, B, \{q_1, q_3\}\} \), where

\[
\begin{align*}
\mu_1 &= \{(p_0, A)^{a/ab}(p_0, A^2), (p_0, A)^{b/ab}(p_1, \varepsilon), (p_0, A)^{c/ab}(p_1, \varepsilon), (p_0, A)^{d/ab}(p_1, \varepsilon), (p_1, A)^{e/ab}(p_1, \varepsilon), (p_1, A)^{f/ab}(p_1, \varepsilon)\} \\
\mu_2 &= \{(q_0, B)^{a/\gamma}(q_1, B^2), (q_0, B)^{b/\gamma}(q_1, B), (q_0, B)^{c/\gamma}(q_1, B), (q_0, B)^{d/\gamma}(q_1, B), (q_0, B)^{e/\gamma}(q_1, B), (q_0, B)^{f/\gamma}(q_1, B)\}
\end{align*}
\]
(q_1, B)^a/b(q_1, B^2), (q_1, B)^b/a(q_2, B),
(q_1, B)^{c/\varepsilon}(q_1, B), (q_2, B)^{b/\varepsilon}(q_2, \varepsilon),
(q_2, B)^{c/\alpha}(q_3, \varepsilon), (q_3, B)^{c/\varepsilon}(q_3, \varepsilon)\}.

Successive application of branching steps yields an intermediate tree containing early nodes numbered \( \mathcal{G} \) in Figure 2 (a). When \( \mathcal{G}(p_3, A^2) \subseteq (q_1, B^3) \) is visited, stack reduction is applied to yield its son \( \mathcal{G}(p_3, A) \subseteq (q_1, B^3) \). And then, after successive application of branching steps, when \( \mathcal{G}(p_3, A^3) \subseteq (q_1, B^4) \) in Figure 2 (b) is visited, stack reduction is similarly applied to yield its son \( \mathcal{G}(p_3, A) \subseteq (q_1, B^4) \). When \( \mathcal{G}(p_3, A^4 \cdot A) \subseteq (q_1, B^4 \cdot B) \) is visited first, skipping is applied with respect to \( \mathcal{G}(p_3, A^4 \cdot A) \) to yield skipping-ends \( \mathcal{G}(p_3, A) \subseteq (q_3, B^3 \cdot B) \). For the node numbered \( \mathcal{G} \), it is confirmed to satisfy the halting condition, since it can find nonhalting brother skipping-ends \( \mathcal{G} \) and \( \mathcal{G} \). Therefore, the status of the node numbered \( \mathcal{G} \) is turned to halting. Furthermore, when the same node \( \mathcal{G}(p_3, A^4 \cdot A) \subseteq (q_1, B^4 \cdot B) \) is visited and applied skipping again, a new skipping-end \( \mathcal{G}(p_3, A \cdot A) \subseteq (q_1, B^5 \cdot B) \) is yielded to have the tree. For the node numbered \( \mathcal{G} \), since it can find a nonhalting brother skipping-end \( \mathcal{G} \), it is also confirmed to satisfy the halting condition and then its status is turned to halting. When the same node \( \mathcal{G}(p_3, A^4 \cdot A) \subseteq (q_1, B^4 \cdot B) \) is visited again, the tree is not changed any more by this step. Then the algorithm halts with the correct conclusion that “\( T_1 \subseteq T_2 \)”.

According to the proof similar to Refs. [8] and [9], we have the following theorem.

**Theorem 1** For the given two droct’s which accept by final state, our inclusion checking algorithm halts in a polynomial-time with respect to the cardinalities of states, input symbols, output symbols, and transition-output rules with the correct conclusion.

### 4 Conclusions

By extending the technique in Ref. [9], we have presented an algorithm for checking the inclusion of non-real-time droct’s (i.e. droct’s with possible \( \varepsilon \)-moves) which accept by final state. The worst-case time complexity of our algorithm is polynomial with respect to the description length of these droct’s.

**Acknowledgments** This work was supported in part by Grants-in-Aid for Scientific Research Nos.20500007 and 2350011 from the MEXT of Japan.

**References**


The Inclusion Checking Algorithm

Check whether $N(p, \alpha) \neq \emptyset$ or not for each configuration $(p, \alpha) \in Q_1 \times \Gamma_1^+$ with $1 \leq |\alpha| \leq |Q_1|$. Similarly, check whether $N(\bar{p}, \beta) \neq \emptyset$ or not for each configuration $(\bar{p}, \beta) \in Q_2 \times \Gamma_2^+$ with $1 \leq |\beta| \leq |Q_2|$. Let the comparison tree consist of only a root labeled $(q_{01}, Z_{01}) \subseteq (q_{02}, Z_{02})$ in unchecked status.

while the comparison tree contains an unchecked or a skipping node do
  if the comparison tree contains an unchecked node then
    let $P$ be the smallest unchecked node
  else let $P$ be the smallest skipping node fi
  suppose $P$ is labeled $(p, \alpha) \subseteq h(\bar{p}, \beta)$;
  if stack reduction is applicable to $P$ then
    apply the stack reduction to $P$;
    turn the status of $P$ to be checked, while its newly added son is in unchecked;
    turn the status of all s-checked nodes to be skipping
  else if $\text{FIRST}_{\text{live}}(p, \alpha) = \{\varepsilon\} \subseteq \text{FIRST}_{\text{live}}(\bar{p}, \beta)$, $(p, \alpha)$ is not in $\varepsilon$-mode and $h \in \Delta^*$ then
    if $h = \varepsilon$ then turn $P$ to checked
    else conclude that “$T_1 \not\subseteq T_2$”; halt fi
  else if $(p, \alpha') \subseteq h'(\bar{p}, \beta') (h' \neq h)$ appears as the label of another internal node then
    conclude that “$T_1 \not\subseteq T_2$”; halt
  else if $(p, \alpha) \subseteq h(\bar{p}, \beta)$ appears as the label of another internal node then
    turn $P$ to checked
  else if $P$ satisfies the halting condition then turn $P$ to halting
  else if the prerequisite for skipping to $P$ is satisfied then
    if skipping to $P$ is applicable then
      apply the skipping to $P$;
      if either any skipping-end has been newly added to $P$ as its son, or any label of an edge from $P$ has been changed by the above skipping then
        turn the status of $P$ to be skipping, while its newly added sons are in unchecked;
        turn the status of all s-checked nodes to be skipping
      else turn the status of $P$ to be s-checked fi
    else conclude that “$T_1 \not\subseteq T_2$”; halt fi
  else if output branch checking is successful for $P$ then
    apply the branching to $P$;
    turn the status of $P$ to be checked, while its newly added sons are in unchecked;
    turn the status of all s-checked nodes to be skipping
  else conclude that “$T_1 \not\subseteq T_2$”; halt fi fi fi fi fi fi fi fi fi od

Conclude that “$T_1 \subseteq T_2$”; halt

Figure 1: The Inclusion Checking Algorithm
Figure 2: The comparison tree for Example 1