IPSJ Transactions on Programming Vol. 1 No. 1 36–46 (June 2008)

Regular Paper

Alternate Stacking Technique Revisited: Inclusion Problem of Superdeterministic Pushdown Automata

Nguyen Van Tang[†1] and Mizuhito Ogawa[†1]

This paper refines the alternate stacking technique used in Greibach-Friedman's proof of the language inclusion problem $L(A) \subseteq L(B)$, where A is a *pushdown automaton* (PDA) and B is a *superdeterministic pushdown automaton* (SPDA). In particular, we propose a product construction of a simulating PDA M, whereas the one given by the original proof encoded everything as a stack symbol. This construction avoids the need for the "liveness" condition in the alternate stacking technique, and the correctness proof becomes simpler.

1. Introduction

Recent interest in model checking makes us recall inclusion problems. Typically, the automata theoretic explanation of model checking on finite transition systems is the decidability of the inclusion problem $L(A) \subseteq L(B)$ among finite automata, where A and B describe a model and a specification, respectively. The standard methodology for the inclusion problem is to, (1) take the complement $L(B)^c$, (2) take the intersection between $L(A)$ and $L(B)^c$, and (3) check its emptiness. This also works when A is extended to a *pushdown automaton* (PDA), but fails when B is extended to a pushdown automaton. To our knowledge, for decidable inclusion with a general pushdown automaton A, the largest class of B is the *superdeterministic pushdown automata* (SPDAs), proposed by Greibach and Friedman [2)]. An SPDA is a DPDA satisfying:

(1) *finite delay* (i.e., a bounded number of ϵ-transitions in a row can be applied to any configuration), and
(2) for two configurations sharing the same control state, transitions with the same symbol lead to configurations sharing the same control state such that the length change of stacks is the same.

In Ref. 2), the authors used the alternate stacking technique [7)] to show that the inclusion problem $L(A) \subseteq L(B)$, where A is a PDA and B is an SPDA, is decidable. The key idea of the original proof [2)] is to construct a simulating pushdown automaton M such that $L(A) \subseteq L(B)$ iff $L(M) = \emptyset$. However, the original construction encodes everything as stack symbols (in an intricate way), and thus control states and transition rules of M could not be given in details. Furthermore, to decide the emptiness of M, one has to use an auxiliary procedure to check whether a configuration of the PDA A is *live* (i.e., whether a configuration reaches an accepting configuration) or not. These properties of their simulating PDA M lead to a complicated proof of soundness and completeness for the decision procedure [2)].

In this paper, we refine the alternate stacking technique [7)] used in Greibach-Friedman's proof [2)]. Basically, there are three main steps in the proof of the decidability of the inclusion problem $L(A) \subseteq L(B)$, where A is a PDA and B is an SPDA. First, establishing Key lemma (Lemma 3.3 [2)]) to find a bounded number k that is used for alternate stacking. Second, constructing a simulating PDA M by using the alternate stacking technique (Section 3). Third, based on the construction of M in the second step, proving soundness and completeness of the construction $L(A) \subseteq L(B)$ iff $L(M) = \emptyset$ (Section 4). Our refinement contributes to the last two steps. In particular, we give a more direct product construction of the simulating PDA M, which is different from the one given by the original proof, where everything is encoded as a stack symbol. This construction avoids the need for the "liveness" condition, and the correctness proof becomes simpler.

This paper is organized as follows. In Section 2, we recall the terminology, notations, and basic definitions of superdeterministic pushdown automata. Section 3 presents our refinement on the alternate stacking technique used in Ref. 2). We show the detailed construction of simulating PDA. This section also gives a simple example to illustrate our construction technique. Section 4 provides simple proof of soundness and completeness for the decision procedure, i.e., $L(A) \subseteq L(B)$ iff $L(M) = \emptyset$. We discuss some related works on decidable inclusion problems in Section 5. Section 6 concludes the paper.

†1 School of Information Science, Japan Advanced Institute of Science and Technology

2. Superdeterministic Pushdown Automata

2.1 Pushdown Automata

Let $\Sigma = \{a, b, c, ...\}$ be a finite set of letters. The set Σ^* denotes all finite words over Σ. The *empty word* is denoted by ε. A subset of Σ^* is called a *language*. Given a nonempty word $w \in \Sigma^*$ we write $w = a_1 a_2 \cdots a_n$ where $a_i \in \Sigma$ denotes the *i-th letter* of w for all $1 \leq i \leq n$. Let denote *head(w)* the first letter of w, i.e., *head(w)* $= a_1$. The *length* $|w|$ of w is n and $|\varepsilon| = 0$. The notation $|\cdot|$ also denotes the *cardinality* of a set, the *absolute value* of an integer, and the *size* of a pushdown automaton (see definition below).

Definition 1. *A pushdown automaton (PDA) A over an alphabet Σ is a tuple $A = (Q, \Sigma, \Gamma, Z_0, \Delta, q_0, F)$, where*

(1) *$Q = \{p, q, r, ...\}$ is a finite set of control states,*

(2) *$\Gamma = \{X, Y, Z, ...\}$ is a finite set of stack symbols such that $Q \cap \Gamma = \emptyset$, $Z_0 \in \Gamma$ is the initial stack symbol,*

(3) *Δ is a finite set of transition rules of the form $(p, X) \xrightarrow{a} (q, \alpha)$ where $p, q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$, and $\alpha \in \Gamma^*$, and $\epsilon \notin \Sigma$ (empty input word) is a special symbol,*

(4) *q_0 is the initial control state,*

(5) *and $F \subseteq Q$ is a set of final control states.*

For a rule $(p, X) \xrightarrow{a} (q, \alpha) \in \Delta$, we call (p, X) the *mode* of the rule with input a; if $a = \epsilon$, this is an *ϵ-rule*. If no rule is defined for (p, X) in $Q \times \Gamma$, (p, X) is a *blocking mode*. If no *ϵ-rule* is defined for mode (p, X) and (p, X) is not a blocking mode, we call it a *reading mode*. We say that a rule $(p, X) \xrightarrow{a} (q, \alpha)$ is a *push, internal*, or *pop* rule if $|\alpha| =$ 2,1, or 0, respectively. A PDA is called *real-time* (RPDA) if $(p, X) \xrightarrow{a} (q, \alpha) \in \Delta$ implies that $a \neq \epsilon$. A PDA is called *deterministic* (DPDA) if for every $p \in Q$, $X \in \Gamma$ and $a \in \Sigma \cup \{\epsilon\}$ we have: (1) $|\{(q, \alpha) \mid (p, X) \xrightarrow{a} (q, \alpha)\}| \leq 1$, and (2) if $(p, X) \xrightarrow{\epsilon} (q, \alpha)$ and $(p, X) \xrightarrow{a} (q', \alpha')$ then $a = \epsilon$.

Let us denote $St = \Gamma^*$. The set $Q \times St$ is the set of *configurations* of a PDA. A pair $(p, \beta X)$ is a configuration *with mode* (p, X), written *mode*$((p, \beta X)) = (p, X)$. The configuration (q_0, Z_0) is called *initial*. For a configuration $c = (p, y)$, the control state of c is *state*$(c) = p$, and the *stack height* of c is $|c| = |y|$.

The transition relation between configurations is defined by: if $(p, X) \xrightarrow{a} (q, \alpha)$, then $(p, \beta X) \xrightarrow{a} (q, \beta\alpha)$ for any $\beta \in \Gamma^*$, and we call it *one-step computation*. A transition $(p, \beta X) \xrightarrow{\epsilon} (q, \beta\alpha)$ is an *ϵ-transition*. If $c_1 \xrightarrow{u} c_2$ and $c_2 \xrightarrow{v} c_3$, we write $c_1 \xrightarrow{uv} c_3$ and call it a computation from c_1 to c_3 on the input uv. For any configuration c, we write $c \xrightarrow{\tau} c$, and we call it a *zero-step* computation, where $\tau a = a\tau = a$ for all $a \in \Sigma$. A sequence $c_1 \xrightarrow{a_1} c_2 \cdots \xrightarrow{a_n} c_{n+1}$ of one-step computations is an *n-step* computation. If we have an n-step computation $c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_3 \cdots \xrightarrow{a_n} c_{n+1}$ with $|c_1| \leq |c_i|$, $1 \leq i \leq n+1$, we write $c_1 \uparrow (a_1 \cdots a_n) c_{n+1}$. This is a *stacking computation*.

A PDA A is of *delay d* if, whenever there is a sequence of one-step computations: $c_1 \xrightarrow{\epsilon} c_2 \xrightarrow{\epsilon} c_3 \cdots \xrightarrow{\epsilon} c_n$, then $n - 1 \leq d$ (i.e., at most d ϵ-rules in a row can be applied to any configuration). A PDA A is d *finite delay* if it is of delay d for some $d \geq 0$. It is easy to see that if a PDA is of delay 0, then it is real-time.

Languages. We consider PDAs accepting by a final state and an empty stack. A language accepted from a configuration c is $L(c) = \{w \in \Sigma^* \mid c \xrightarrow{w} (q, \varepsilon), q \in F\}$. The language accepted by a PDA A is $L(A) = L((q_0, Z_0))$. The PDAs M_1 and M_2 are equivalent, denoted as $M_1 \equiv M_2$, if they accept the same language, i.e., $L(M_1) = L(M_2)$. Configurations c_1 in M_1 and c_2 in M_2 are equivalent, denoted as $c_1 \equiv c_2$, if $L(c_1) = L(c_2)$. For a configuration c, c is *accessible* if $(q_0, Z_0) \xrightarrow{w} c$ for some $w \in \Sigma^*$. c is *live* if $c \xrightarrow{w} (q, \varepsilon)$ for some $q \in F$ and some $w \in \Sigma^*$.

2.2 Normalized Pushdown Automata

For the purpose of our work, it is convenient to use a normal form of pushdown automata.

Definition 2. *A pushdown automaton $A = (Q, \Sigma, \Gamma, Z_0, \Delta, q_0, F)$ is normalized if*

(1) *for all $p \in Q$ and $X \in \Gamma$, (p,X) is not a blocking mode;*

(2) *for all $p \in Q$, all rules in δ of the form $(p, X) \xrightarrow{a} (q, \alpha)$ either satisfy $a \in \Sigma$ or all of them satisfy $a = \epsilon$, but not both;*

(3) *every rule in δ is of the form $(p, X) \xrightarrow{a} (q, \varepsilon)$, $(p, X) \xrightarrow{a} (q, X)$, or $(p, X) \xrightarrow{a} (q, XY)$ where $a \in \Sigma \cup \{\epsilon\}$.*

The next lemma enables us to convert an arbitrary PDA into an equivalent normalized PDA.

Lemma 1 (Nowotka-Srba[3]). *For every PDA (DPDA) there is a normalized PDA (DPDA) that recognizes the same language.*

2.3 Superdeterministic Pushdown Automata

Superdeterministic pushdown automata (SPDAs) were first introduced by Greibach and Friedman[2]. In this section, we briefly recall the standard notion and key properties of SPDAs. Readers are referred to the seminal paper[2] for more details.

Definition 3. *A PDA $A = (Q, \Sigma, \Gamma, Z_0, \Delta, q, F)$ is superdeterministic if it satisfies the following conditions.*

(1) *A is deterministic and of finite delay,*

(2) *for all accessible configurations in reading mode c_1, c_2, c_1', c_2' and $w \in \Sigma^*$, if both of the following are satisfied:*

- *$state(c_1) = state(c_2)$,*
- *$c_1 \xrightarrow{w} c_1'$ and $c_2 \xrightarrow{w} c_2'$,*

then, $state(c_1') = state(c_2')$ and $|c_1| - |c_1'| = |c_2| - |c_2'|$.

Remark 1. *In[2], Greibach and Friedman considered the blocking condition on PDAs (middle, pp.677): "Unlike Valiant, we do not allow the pda to operate with empty stack (no rules (q, ϵ, a, p, y)). This avoids some complications in notation but does not affect the classes of languages involved because we allow endmarkers". In particular, the blocking condition is not an essential restriction if we use two special symbols # (start-maker) and \$ (end-marker), where # pushes a special stack symbol, and \$ pops it.*

This assumption was used to prove Key lemma (Lemma 2). More precisely, it was used to show the claim (middle, pp.684[2]) that: "Hidden in many of our arguments is the following consequence of determinism and acceptance by empty store. Suppose $L(c_1) \subseteq L(\bar{c}_1)$ with $\bar{c}_1$ (but not necessarily c_1) a configuration in a deterministic pda, $c_1 \xrightarrow{w} c_2$, and $\bar{c}_1 \xrightarrow{w} \bar{c}_2$. Then $L(c_2) \subseteq L(\bar{c}_2)$".

Definition 4. *A language L is superdeterministic if there is an SPDA M such that either $L = L(M)$ or $L\$ = L(M)$ for an end-marker \$.*

Note that the language $\{a^n b^n \mid n \geq 0\}$ is superdeterministic. However, due to Condition 2 in Definition 3, the language $L = \{a^n b^m \mid m \geq n\}$ is not accepted by any SPDA (pp.678[2]). Suppose on the contrary that there is an SPDA A accepting L. While reading a, A pushes a symbol, and while reading b, A pops the same symbol. Thus, for instance, after reading a^5 and a^{10}, A will be in two configurations, c_1 and c_2, such that $state(c_1) = state(c_2)$. Now concatenating b^{10}, A will lead to configurations c_1' (for a^5b^{10}) and c_2' (for $a^{10}b^{10}$), respectively. However, $|c_1'| - |c_1| = 0 - 5 \neq |c_2'| - |c_2| = 0 - 10$. This violates the definition of SDPAs. Moreover, as shown in[2], the class of *superdeterministic languages* (languages accepted by SPDAs) contains the generalized parenthesis languages, which is a superclass of both parenthesis languages[6] and Dyck sets.

Remark 2. *It is undecidable whether a given context-free language is superdeterministic. However, it is decidable whether a given PDA M is an SPDA (pp.678[2]): "It is decidable whether a dpda M is finite delay (using the decidability of emptiness and finiteness for context-free grammars and the standard construction of grammars from machines), and if M is of finite delay, an upper bound d on the delay can be computed from a description of M. Knowing that M is of delay d, it can be determined whether or not M is superdeterministic by examining only computations $c \xrightarrow{a} c'$ for a symbol a with c and c' in reading mode. Since it is decidable for q in Q, y in Γ^* whether there is a u in Γ^* with (q, uy) accessible, it is decidable whether a dpda is superdeterministic. It is not known if it is decidable whether a deterministic context-free language is superdeterministic, just as it is not known whether it is decidable whether a deterministic context-free language is finite-turn or one-counter[7]. Standard arguments show that it is undecidable whether an arbitrary context-free language is superdeterministic".*

A PDA is called *one-increasing* if the stack height increases by at most one per move. As is well known, each PDA can be transformed into an equivalent one-increasing PDA.

Lemma 2 (Key Lemma 3.3[2]). *Let A be a normalized PDA, and B be a one-increasing SPDA of delay d. Let c_1 be a configuration in A and c_1' be an accessible configuration in B with $L(c_1) \subseteq L(c_1')$. Suppose we have in A a computation $c_1 \uparrow (w) c_2$, with c_2 live, and in B a computation $c_1' \xrightarrow{w} c_2'$. Then,*

(1) *$|c_1'| - |c_2'| \leq k$,*

(2) *and if $|c_1| = |c_2|$ then $|c_2'| - |c_1'| \leq k$.*

where,

- $k = (d+1)(k_1+1)n(m+1)^{2k_2} + 2d$,
- $k_1 = n + 3$, $k_2 = 1 + 2n^2m^2(n^2+4)$,

- $n = |Q_A| + |Q_B|$, $m = |\Gamma_A| + |\Gamma_B|$.

Based on this property, in the next section, we show that the inclusion problem $L(A) \subseteq L(B)$ is decidable for a PDA A and an SPDA B.

3. Alternate Stacking Technique

The alternate stacking technique, proposed by Valiant [7)], involves a simulation of two PDAs A and B using a single stack machine M whose stack contents $u_1v_1 \cdots u_tv_t$ encode the stack $u_1 \cdots u_t$ of A and $v_1 \cdots v_t$ of B; the machine M uses u_i to simulate one step of A and v_r for one step of B. In the general case, the simulating machine M is not a PDA. Alternate stacking "succeeds" when the stacks can be interwoven in such a way that M can be implemented as a PDA. Valiant [7)] showed that if A and B are *nonsingular DPDAs*[⋆1] and $L(A) = L(B)$, then the interweaving can indeed be done so that a uniform bound can be placed on the length of segments u_i and v_i so long as the configurations of A and B are live. Then the PDA M can be built so that if the stack segments exceed the bound, M accepts, knowing that $L(A) \neq L(B)$. Hence $L(A) = L(B)$ iff $L(M) = \emptyset$.

3.1 Simulating Pushdown Automata

In this subsection, we construct a simulating PDA M such that M will search for possible members of $L(A) \setminus L(B)$. In principle, similar to[2)], the key is to use the alternate stacking technique to construct M. In our approach, however, the control states, stack symbols, and transition rules of M are defined in the form of pairs of states, stack content, and transition rules of two PDAs, respectively.

We assume that $A = (Q_A, \Sigma, \Gamma_A, Z_A, \Delta_A, q_A^0, F_A)$ is a normalized PDA, and $B = (Q_B, \Sigma, \Gamma_B, Z_B, \Delta_B, q_B^0, F_B)$ is a normalized SPDA of delay d with an assumption that $0 \notin Q_B$. Let $\$_1$ and $\$_2$ be fresh symbols to mark the bottom of the stack of A and B, respectively. Let $f : \Gamma_B^* \cup \$_2^*\Gamma_B^* \to \Gamma_B^*$ be a function such that $f(y) = f(\$_2^* y) = y$ for all $y \in \Gamma_B^*$. Let $r > 0$ be an integer and let us take $2r$ as the segment bound for simulating the stack content of B. Denote $\Gamma'_B = \{[y], [\$_2 y] \mid y \in \Gamma_B^*, 0 \leq |y| \leq 2r\}$. A simulating PDA $M = M(A, B, r)$ can be constructed for any choice of r, and the next theorem, Theorem 5, will show that if the bound r is appropriately selected ($r = k+1$, where k was computed from A and B as in Theorem 2), then we can conclude that $L(A) \subseteq L(B)$ iff $L(M(A, B, k+1)) = \emptyset$. Formally, the simulating PDA $M = M(A, B, r)$ is constructed as follows:

Definition 5. *A simulating PDA of A and B is a tuple $M = M(A, B, r) = \langle Q_M, \Sigma, \Gamma_M, Z_M, \Delta_M, p_M^0, F_M \rangle$, where:*

- $Q_M = \{p_M^0\} \cup (Q_A \times Q_B) \cup (Q_A \times \{0\}) \cup (Q_A \times Q_B \times \Gamma'_B)$ *is the set of finite states,*
- p_M^0 *is the initial state,*
- $F_M = (F_A \times (Q_B \backslash F_B)) \cup (F_A \times \{0\})$,
- $\Gamma_M = (\Gamma_A \cup \{\$_1\}) \times \Gamma'_B$, $Z_M = (\$_1, [\$_2])$,
- *The transition relation $\Delta_M \subseteq Q_M \times \Gamma_M \times \Sigma \times (Q_M \times \Gamma_M^*)$ is defined as follows:*

Case I: *Simulating an internal-transition of A with a transition of B:*

(1) $\langle (p_1, p_2), (X, [vZ]) \rangle$[⋆2] $\xrightarrow{a} \langle (p'_1, p'_2), (X, [vy]) \rangle$ *if:*

$$\begin{cases} (p_1, X) \xrightarrow{a} (p'_1, X) \in \Delta_A \\ (p_2, Z) \xrightarrow{a} (p'_2, y) \in \Delta_B \\ vy \neq \epsilon, \text{ and } |f(vy)| \leq 2r \end{cases}$$

(2) $\langle (p_1, p_2), (X, [vZ]) \rangle \xrightarrow{a} \langle (p'_1, 0), (X, [vy]) \rangle$ *if:*

$$\begin{cases} (p_1, X) \xrightarrow{a} (p'_1, X) \in \Delta_A \\ (p_2, Z) \xrightarrow{a} (p'_2, y) \in \Delta_B \\ vy = \varepsilon \text{ or } |f(vy)| = 2r+1 \end{cases}$$

(3) $\langle (p_1, p_2), (X, [vZ]) \rangle \xrightarrow{a} \langle (p'_1, 0), (X, [vZ]) \rangle$ *if:*

$$\begin{cases} (p_1, X) \xrightarrow{a} (p'_1, X) \in \Delta_A \\ (p_2, Z) \text{ has no rules with input } a \end{cases}$$

⋆1 A DPDA M is *nonsingular* if and only if there exists $m \geq 0$ such that for any two accessible configurations $(q, w'w)$ and (q', w') where $|w| > m$, if $L((q, w'w)) = L((q', w'))$ then $L((q', w')) = \emptyset$.

⋆2 For readability, we use $\langle ., . \rangle$ to denote a configuration of the simulating PDA M.

(4) $\langle (p_1,0),(X,[v])\rangle \xrightarrow{a} \langle (p_1',0),(X,[v])\rangle$ *for all* $[v]\in\Gamma_B'$ *if:*

$$\left\{\ (p_1,X)\xrightarrow{a}(p_1',X)\in\Delta_A\right.$$

(5) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{\epsilon} \langle (p_1',p_2),(X,[vZ])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{\epsilon}(p_1',X)\in\Delta_A\\ (p_2,Z)\ \textit{has no}\ \epsilon\textit{-rules}\end{cases}$$

(6) $\langle (p_1,p_2),(X,[\$_2])\rangle \xrightarrow{a} \langle (p_1',0),(X,[\$_2])\rangle$ *if* $(p_1,X)\xrightarrow{a}(p_1',X)\in\Delta_A$

Case II: *Simulating a push-transition of* A *with a transition of* B.

(1) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{a} \langle (p_1',p_2'),(X,[\$_2])(X',[vy])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\\ (p_2,Z)\xrightarrow{a}(p_2',y)\in\Delta_B\\ head(vy)=\$_2,|f(vy)|\le r\end{cases}$$

(2) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{a} \langle (p_1',p_2'),(X,[\varepsilon])(X',[vy])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\\ (p_2,Z)\xrightarrow{a}(p_2',y)\in\Delta_B\\ head(vy)\ne\$_2,|f(vy)|\le r\end{cases}$$

(3) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{a} \langle (p_1',p_2'),(X,[v'])(X',[v''])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\\ (p_2,Z)\xrightarrow{a}(p_2',y)\in\Delta_B\\ r<|f(vy)|\le 2r,\\ vy=v'v'',|v''|=r\end{cases}$$

(4) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{a} \langle (p_1',p_2'),(X,[v'])(X',[v''])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\\ (p_2,Z)\xrightarrow{a}(p_2',y)\in\Delta_B\\ |f(vy)|=2r+1,\\ vy=v'v'',|v''|=r+1\end{cases}$$

(5) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{a} \langle (p_1',0),(X,[\varepsilon])(X',[vZ])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\\ (p_2,Z)\ \textit{has no rules with input}\ a\end{cases}$$

(6) $\langle (p_1,0),(X,[v])\rangle \xrightarrow{a} \langle (p_1',0),(X,[v])(X',[v])\rangle$ *for all* $[v]\in\Gamma_B'$ *if:*

$$\left\{\ (p_1,X)\xrightarrow{a}(p_1',XX')\in\Delta_A\right.$$

(7) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{\epsilon} \langle (p_1',p_2),(X,[\$_2])(X',[vZ])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{\epsilon}(p_1',XX')\in\Delta_A\\ head(vZ)=\$_2,|f(vZ)|\le r\\ (p_2,Z)\ \textit{has no}\ \epsilon\textit{-rules}\end{cases}$$

(8) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{\epsilon} \langle (p_1',p_2),(X,[\varepsilon])(X',[vZ])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{\epsilon}(p_1',XX')\in\Delta_A\\ head(vZ)\ne\$_2,|f(vZ)|\le r\\ (p_2,Z)\ \textit{has no}\ \epsilon\textit{-rules}\end{cases}$$

(9) $\langle (p_1,p_2),(X,[vZ])\rangle$
$\xrightarrow{\epsilon} \langle (p_1',p_2),(X,[v'])(X',[v"])\rangle$ *if:*

$$\begin{cases}(p_1,X)\xrightarrow{\epsilon}(p_1',XX')\in\Delta_A\\ r<|f(vZ)|\le 2r, vZ=v'v'',|v''|=r\\ (p_2,Z)\ \textit{has no}\ \epsilon\textit{-rules}\end{cases}$$

(10) $\langle (p_1,p_2),(X,[\$_2])\rangle \xrightarrow{a} \langle (p'_1,0),(X,[\$_2])(X',[\$_2])\rangle$ *if:*

$$\left\{\ (p_1,X) \xrightarrow{a} (p'_1,XX') \in \Delta_A\right.$$

Case III: *Simulating a pop-transition of A with a transition of B:*

(1) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{a} \langle (p'_1,p'_2,[f(v)y]),\varepsilon\rangle$, $\langle (p'_1,p'_2,[f(vy)]),(X',[v'])\rangle \xrightarrow{\epsilon} \langle (p'_1,p'_2),(X',[v'f(vy)])\rangle$ *if:*

$$\begin{cases} (p_1,X) \xrightarrow{a} (p'_1,\varepsilon) \in \Delta_A \\ (p_2,Z) \xrightarrow{a} (p'_2,y) \in \Delta_B \\ |f(v'f(vy))| \le 2r \end{cases}$$

(2) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{a} \langle (p'_1,p'_2,[f(vy)]),\varepsilon\rangle$, *and* $\langle (p'_1,p'_2,[f(vy)]),(X',[v'])\rangle \xrightarrow{\epsilon} \langle (p'_1,0),(X',[\varepsilon])\rangle$ *if:*

$$\begin{cases} (p_1,X) \xrightarrow{a} (p'_1,\varepsilon) \in \Delta_A \\ (p_2,Z) \xrightarrow{a} (p'_2,y) \in \Delta_B \\ v'vy = \varepsilon \text{ or } |f(v'f(vy))| \ge 2r+1 \end{cases}$$

(3) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{a} \langle (p'_1,0),\varepsilon\rangle$ *if:*

$$\begin{cases} (p_1,X) \xrightarrow{a} (p'_1,\varepsilon) \in \Delta_A \\ (p_2,Z) \text{ has no rules with input } a \end{cases}$$

(4) $\langle (p_1,0),(X,[v])\rangle \xrightarrow{a} \langle (p'_1,0),\varepsilon\rangle$ *for all* $[v] \in \Gamma'_B$ *if* $(p_1,X) \xrightarrow{a} (p'_1,\varepsilon) \in \Delta_A$

(5) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{\epsilon} \langle (p'_1,p_2,[f(vZ)]),\varepsilon\rangle$, *and* $\langle (p'_1,p_2,[f(vZ)]),(X',[v'])\rangle \xrightarrow{\epsilon} \langle (p'_1,p_2),(X',[v'f(vZ)])\rangle$ *if:*

$$\begin{cases} (p_1,X) \xrightarrow{\epsilon} (p'_1,\varepsilon) \in \Delta_A \\ vZ = \$_2 \\ (p_2,Z) \text{ has no } \epsilon\text{-rules} \end{cases}$$

(6) $\langle (p_1,p_2),(X,[\$_2])\rangle \xrightarrow{a} \langle (p'_1,0),\varepsilon\rangle$ *if:*

$$\left\{\ (p_1,X) \xrightarrow{a} (p'_1,\varepsilon) \in \Delta_A\right.$$

Case IV: *When stack of A is empty.*

(1) $\langle (p_1,p_2),(\$_1,[vZ])\rangle \xrightarrow{\epsilon} \langle (p_1,p'_2),(\$_1,[vy])\rangle$ *if* $(p_2,Z) \xrightarrow{\epsilon} (p'_2,y) \in \Delta_B$

(2) $\langle (p_1,p_2),(\$_1,[vZ])\rangle \xrightarrow{\epsilon} \langle (p_1,0),\varepsilon\rangle$ *if:*

$$\begin{cases} (p_2,Z) \xrightarrow{a} (p'_2,y) \in \Delta_B \text{ with } a \ne \epsilon, \\ \text{or } (p_2,Z) \text{ is blocked} \end{cases}$$

Case V: *When configurations of A are in the reading modes, while states of B have ϵ-transitions.*

(1) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{\epsilon} \langle (p_1,p'_2),(X,[vy])\rangle$ *if:*

$$\begin{cases} (p_1,X) \text{ is in the reading mode} \\ (p_2,Z) \xrightarrow{\epsilon} (p'_2,y) \in \Delta_B, |f(vy)| \le 2r \end{cases}$$

(2) $\langle (p_1,p_2),(X,[vZ])\rangle \xrightarrow{\epsilon} \langle (p_1,0),(X,[\varepsilon])\rangle$ *if:*

$$\begin{cases} (p_1,X) \text{ is in the reading mode} \\ (p_2,Z) \xrightarrow{\epsilon} (p'_2,y) \in \Delta_B, |f(vy)| \ge 2r+1 \end{cases}$$

Case VI: *The starting transition:*

$\langle p^0_M,(\$_1,[\$_2])\rangle \xrightarrow{\epsilon} \langle (q^0_A,q^0_B),(\$_1,[\$_2])(Z_A,[\$_2 Z_B])\rangle$

Before defining configurations of M, let us briefly explain the intuition behind its transition rules.

- Rules I (2), III (2), and V (2) are called *stacking-fail* transitions. Taking a stacking-fail transition, M changes its control to states in the set $Q_A \times \{0\}$. After entering this set $Q_A \times \{0\}$ of states, M continues simulating transitions of A only by using rules I (4), II (6), or III (4).
- Rules I (3), II (5), and III (3) are used when B is blocked where reading an input. In this cases, M changes its control state to the set $Q_A \times \{0\}$. After entering a state in $Q_A \times \{0\}$, M only simulates transitions of A by using I (4), II (6), or III (4).
- Rules II (1), II (2), II (3), and II (4) are used to simulate a push-transition of A with a transition of B, which has the same label.
- Rules III (1), III (2) are used to simulate a pop-transition of A.
- Rules IV (1) and IV (2) are used when the stack of A is empty; in this case, M simulates ϵ-transitions of B using a zero-step computation of A. Recall that B is finite delay of d, and thus rules IV (1) and IV (2) can be applied at most d times in a sequence.

- Rules I (5), II (7)(8)(9), and III (5) are used to simulate an ϵ-transition of A with a non-ϵ transition of B.
- Rules I (6), II (8), and III (6) are used to simulate a non-ϵ transition of A when B's stack is empty (i.e., when B is blocked).

Definition 6 (Configuration). • *A configuration of M is of the form $c = \langle s, (\$_1, [\$_2])(X_1, [v_1]) \cdots (X_t, [v_t])\rangle$, where $s \in Q_M$ and $(X_i, [v_i]) \in \Gamma_M$ for $1 \leq i \leq t$.*

- *c is an accepting configuration if $c = \langle s, \varepsilon\rangle$, $s \in F_M$.*

For a given configuration $c = \langle s, (\$_1, [\$_2]) \cdots (X_t, [v_t])\rangle$, we say that c encodes $c_1 = (p_1, X_1 ... X_t)$ of A and $c_2 = (p_2, f(v_1)...f(v_t))$ of B, with t levels. Note that $f(v_1)...f(v_t) \in \Gamma_B^*$, and M can determine whether the stack of B is empty by examining if $v_t = \$_2$, i.e., $|c_2| = 0$ iff $v_t = \$_2$. This is because, based on the transition rules II (1) and II (7), if $v_t = \$_2$ then $v_i = \$_2$ for all $1 \leq i \leq t$ (for these rules, we need to check if $head(vy) = \$_2$).

Remark 3. *There are three main steps in the proof of the decidability of the inclusion problem $L(A) \subseteq L(B)$, where A is a PDA and B is an SPDA. First, establish the Key lemma to find a bounded number k that is used for alternate stacking. Second, construct a simulating PDA M by using the alternate stacking technique. Third, based on the construction of M in the second step, prove the soundness and completeness of the construction $L(A) \subseteq L(B)$ iff $L(M) = \emptyset$. Our refinement contributes to the last two steps. In particular, in the original proof[2], the liveness condition is stated in the construction case II (pp. 693[2]) and it is used for searching words that are accepted by the PDA A, but rejected by the SPDA B. In our encoding, control states, stack symbols, and transition rules M are defined in the form of pairs of states, stack content, and transition rules of two PDAs A and B, respectively. Thus, we do not need to use the "liveness" condition, because such violation of the inclusion is represented by transition rules in our product construction of M. As we will see in Section 4, a proof of "liveness" is not needed and the whole correctness proof for the decision procedure becomes simpler.*

3.2 An Example

This subsection provides an example to illustrate our construction of the simulating pushdown automata. In the following figures, for simplicity, we describe

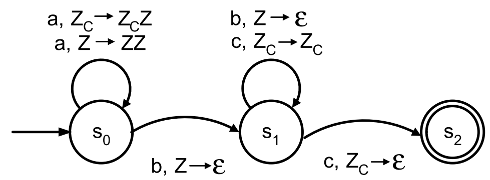

Fig. 1 Pushdown automaton C.

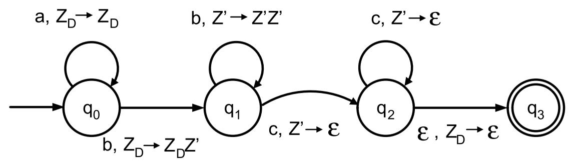

Fig. 2 Superdeterministic pushdown automaton D.

control states of each PDA as nodes of a graph. We adopt the following conventions to represent edges: for a transition rule $(p, X) \xrightarrow{a} (q, y)$, we label the edge from p to q as $a, X \to y$.

Example 1. *Consider two PDAs C (in* **Fig. 1***) and D (in* **Fig. 2***) over the input alphabet $\Sigma = \{a, b, c\}$,where D is an SPDA. The PDA $C = (\{s_0, s_1, s_2\}, \Sigma, \{Z, Z_C\}, Z_C, \Delta_C, \{s_0\}, \{s_2\})$, where Δ_C is defined as:*

- $(s_0, Z_C) \xrightarrow{a} (s_0, Z_C Z)$
- $(s_0, Z) \xrightarrow{a} (s_0, ZZ)$
- $(s_0, Z) \xrightarrow{b} (s_1, \varepsilon)$
- $(s_1, Z) \xrightarrow{b} (s_1, \varepsilon)$
- $(s_1, Z_C) \xrightarrow{c} (s_1, Z_C)$
- $(s_1, Z_C) \xrightarrow{c} (s_2, \varepsilon)$

$L(C) = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$.

The SPDA $D = (\{q_0, q_1, q_2, q_3\}, \Sigma, \{Z', Z_D\}, Z_D, \Delta_D, \{q_0\}, \{q_3\})$, where Δ_D is defined as:

- $(q_0, Z_D) \xrightarrow{a} (q_0, Z_D)$
- $(q_0, Z_D) \xrightarrow{b} (q_1, Z_D Z')$
- $(q_1, Z') \xrightarrow{b} (q_1, Z'Z')$

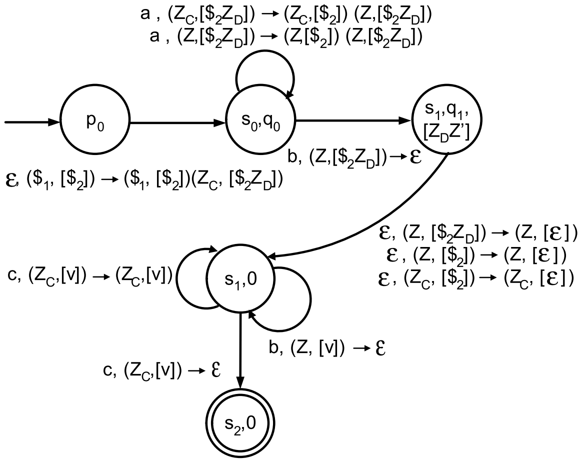

Fig. 3 The simulating PDA $M(C, D, 1)$.

- $(q_1, Z') \xrightarrow{c} (q_2, \varepsilon)$
- $(q_2, Z') \xrightarrow{c} (q_2, \varepsilon)$
- $(q_2, Z_D) \xrightarrow{\epsilon} (q_3, \varepsilon)$

$L(D) = \{a^m b^n c^n \mid m \geq 0, n \geq 1\}$.

The simulating pushdown automaton $M = M(C, D, 1)$ is illustrated in **Fig. 3**. In this case, $r = 1$, it is sufficient to consider stack symbols of the forms $(X, [v])$ with $|v| \leq 2$. The language of M is $L(M(C, D, 1)) = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$.

4. Soundness and Completeness

In this section, we show that the construction presented in the preceding section is sound and complete, i.e., $L(A) \subseteq L(B)$ if and only if $L(M(A, B, k+1)) = \emptyset$, where k was computed from A and B as in Lemma 2.

4.1 Soundness

Lemma 3. *$L(A) \not\subseteq L(B)$ implies $L(M(A, B, r))$ $\neq \emptyset$ for all $r \geq 1$.*

Proof. Let $w \in L(A) \backslash L(B)$. It is sufficient to show that $w \in L(M)$. Denote c_{in} as the initial configuration of M. Recall that A is normalized (by Lemma 1), there is a computation of A on every word. By the definition of transitions of M, there is a computation of M on w. There are three cases:

- **Case 1:** There are no computations of B on w, or there is a computation of B on w but after reading w, the stack of B is nonempty. By transitions of M, we have $c_{in} \xrightarrow{w} \langle (p, 0), \varepsilon \rangle$. Since $w \in L(A)$, $(p, 0) \in F_A \times \{0\}$. Thus, $w \in L(M)$.
- **Case 2:** There is a computation of B on w leading to a configuration (q, ε), where $q \notin F_B$. Because A accepts w, there is a computation of M on w leading to a configuration $\langle (p, q), \varepsilon \rangle$, $(p, q) \in F_A \times (Q_B \setminus F_B)$. Thus, $w \in L(M)$.
- **Case 3:** Where simulating w, the stacking fails. In this case, we have $c_{in} \xrightarrow{w} \langle (p, 0), \varepsilon \rangle$, $p \in F_A$.

□

4.2 Completeness

Lemma 4. *Let k be the number computed in Lemma 2. $L(M(A, B, k+1)) \neq \emptyset$ implies $L(A) \not\subseteq L(B)$.*

Proof. Let $w \in L(M(A, B, k+1))$. Thus, there is an accepting computation of $M(A, B, k+1)$ on w. We consider two cases of accepting configurations of M.

1) **Case 1:** $c_{in} \xrightarrow{w} \langle (p, q), \varepsilon \rangle$ where $(p, q) \in F_A \times (Q_B \setminus F_B)$. In this case, there is a computation of B on w leading to the configuration (q, ε). Because $q \notin F_B$, we obtain $w \notin L(B)$. On the other hand, on reading w, A leads to the accepting configuration (p, ε), i.e., $w \in L(A)$. Thus, $w \in L(A) \setminus L(B)$.

2) **Case 2:** $c_{in} \xrightarrow{w} \langle (p, 0), \varepsilon \rangle$ where $p \in F_A$. Consider two subcases. First, if B is blocked at some point on reading w. In this case, there is not a computation of B on w, i.e., $w \notin L(B)$. On the other hand, on reading w, A leads to the configuration (p, ε), $p \in F_A$. Hence $w \in L(A)$. Since B is deterministic, we have $w \in L(A) \setminus L(B)$. The proof is completed. Second, if stacking fails at some point on simulating w. In this case, to prove $L(A) \not\subseteq L(B)$, we assume on the contrary that $L(A) \subseteq L(B)$. We will show a contradiction. Since the stacking fails on reading w, we suppose that $w = w_1 w_2$ such that, after reading w_1 the first time, stacking fail occurs and M is in the control $(p_1, 0)$ with the stack content $(\$_1, [\$_2])(X_1, [v_1]) \cdots (X_{t-1}, [v_{t-1}])(X_t, [v_t])$.

Whereas after reading w_1, A is in the configuration $c_2 = (p_1, X_1...X_t)$ (c_2 is live) and B is in the configuration $c'_2 = (p_2, f(v_1...v_{t-1}v_t))$. There are two subcases which lead to the stacking failure: either $[v_t] = [\varepsilon]$ or $|f(v_t)| \geq 2r+1$.

- **If** $[v_t] = [\varepsilon]$**:** we have $t \geq 2$ and $f(v_1...v_t) \neq \epsilon$ (because, if $t=1$ then $[v_t]$ must be $[\$_2]$, and if $f(v_1...v_t) = \epsilon$ then the stack of B is empty and $[v_t] = [\$_2]$). Since $f(v_1...v_t) \neq \epsilon$ there is at least one $f(v_i) \neq \epsilon$. Select the "nearest" v_j such that $f(v_j) \neq \epsilon$ and $f(v_i) = \epsilon$ for $j+1 \leq i \leq t$. Consider the time when the level $j+1$ of the stack is opened. Since $f(v_j) \neq \epsilon$, this means that the rule II (3) or II (4) was used, and the "new" top segment at that time was v'_{j+1} with $|v'_{j+1}| = r$ or $|v'_{j+1}| = r+1$. Since that time, M has not read below level $j+1$. Thus, we have $w_1 = w'w''$, and after reading w', M is in the configuration $\langle (p'_1, p'_2), (\$_1, [\$_2]) \cdots (X_j, [v_j])(X'_{j+1}, [v'_{j+1}]) \rangle$ encoding the configurations
  $c_1 = (p'_1, X_1...X_jX'_{j+1})$ of A, and $c'_1 = (p'_2, f(v_1...v_jv'_{j+1}))$ of B such that: $c_0 \xrightarrow{w'} c_1$ and $c'_0 \xrightarrow{w'} c'_1$ (c_0 and c'_0 are the initial configurations of A and B, respectively). Because $L(A) \subseteq L(B)$ (by assumption) and B is deterministic, $L(c_1) \subseteq L(c'_1)$. On the other hand, we have $c_1 \uparrow (w'')c_2$ and $c'_1 \xrightarrow{w''} c'_2$. Note that these conditions satisfy assumptions of the Key lemma (Lemma 2). However, we have $|c'_1| - |c'_2| = |v'_{j+1}| \geq r = k+1 > k$. This contradicts Lemma 2. Hence, the assumption $L(A) \subseteq L(B)$ is wrong. Thus, $L(A) \not\subseteq L(B)$.
- **If** $|f(v_t)| \geq 2r+1$**:** Consider the time when the level $t-1$ of the stack is opened. At that point, one of rules II (1), II (2), II (3), or II (4) was used and the "new" top segment was v'_{t-1} with $|v'_{t-1}| \leq r+1$. Since that time, M has not read below level $t-1$. Thus, we have $w_1 = w'w''$, and after reading w', M is in the configuration $\langle (p'_1, p'_2), (\$_1, [\$_2]) \cdots (X'_{t-1}, [v'_{t-1}]) \rangle$ encoding the configurations
  $c_1 = (p'_1, X_1...X_{t-2}X'_{t-1})$ of A, and $c'_1 = (p'_2, f(v_1...v_{t-2}v'_{t-1}))$ of B such that: $c_0 \xrightarrow{w'} c_1$ and $c'_0 \xrightarrow{w'} c'_1$. Because $L(A) \subseteq L(B)$ (by assumption) and B is deterministic, $L(c_1) \subseteq L(c'_1)$. In addition, we have $c_1 \uparrow (w'')c_2$ and $c'_1 \xrightarrow{w''} c'_2$. Note that these conditions satisfy assumptions of Lemma 2. Now, we can compute:
$$\begin{cases} |c_1| - |c_2| = |X_{t-1}| - |X'_{t-1}| = 1 - 1 = 0 \\ |c'_2| - |c'_1| = |v_t| - |v'_{t-1}| \geq k+1 > k. \end{cases}$$
  This contradicts Lemma 2. Hence $L(A) \not\subseteq L(B)$.

In both cases, we have, if $L(M(A, B, k+1)) \neq \emptyset$, then $L(A) \not\subseteq L(B)$. The lemma is proved. □

From Lemmas 3 and 4, we obtain:

Lemma 5. *$L(A) \subseteq L(B)$ if and only if $L(M(A, B, k+1)) = \emptyset$.*

4.3 The Inclusion Problem

Let $A = (Q, \Sigma, \Gamma, Z_0, \Delta, q_0, F)$ be a PDA. The *size* $|A|$ of a PDA A is defined as $|Q| + |\Sigma| + |\Gamma| + \{|pXq\alpha| \mid (p, X) \xrightarrow{a} (q, \alpha) \in \Delta\}$. We obtain the same complexity class as that of the original construction.

Theorem 1. *The inclusion problem $L(A) \subseteq L(B)$, where A is a PDA and B is an SPDA, is decidable. Furthermore, the decision procedure has time complexity bounded by $2^{2^{p(h)}}$, where $p(h)$ is a polynomial time in the size of both automata, $h = |A| + |B|$.*

Proof. The decidability follows from Lemma 5. We now approximate the size of M. Recall that the emptiness problem can be decided in $O(n^3)$ for any PDA of size n. The stack of M is bounded by $|\Gamma_A| \cdot |\Gamma_B|^{2k+2}$, where k is the number given in Lemma 2. The maximum number of control states of M is $|Q_A| \cdot |Q_B| \cdot |\Gamma_B|^{2k+1}$. The number of transitions of M is bounded by $|Q_A|^2 \cdot |Q_B|^2 \cdot |\Sigma| \cdot |\Gamma_A|^3 \cdot |\Gamma_B|^{6k+6}$. Recall that $s = |Q_A| + |Q_B|$, $g = |\Gamma_A| + |\Gamma_B|$. The size of M is bounded by $|M| \leq s^4 g^{6k+6}$. Lemma 2 expresses that $k = (d+1)(s+4)g(g+1)^{2(1+2s^2g^2(s^2+4))} + 2d$, where d is the delay of B. Define $h = s + g$, and we see that $k \leq h^{c_1 h^{c_2}}$, for some constants c_1 and c_2. Thus, for some constant c_3, the size of M is bounded by $h^{c_3 h^{c_1 h^{c_2}}}$. Thus, the time complexity of the construction is bounded by $2^{2^{p(h)}}$ for a polynomial $p(h)$. □

5. Related Work

SPDAs were proposed by S. Greibach and E. Friedman in Refs. 2) and 4). It is shown that the acceptance condition of SPDAs does strictly affect decision problems. More precisely, for SPDAs accepting by final control state, the inclusion problem is undecidable[4]. If we consider SPDAs accepting by a final state and an empty stack, it is shown that the language inclusion problem $L(A) \subseteq L(B)$ is decidable for A is an arbitrary PDA, and B is an SPDA[2]. As far as we know, the class of SPDAs is the largest class which enjoys decidability for this inclusion problem. The main results of the inclusion problem $L(A) \subseteq L(B)$, in which A is an arbitrary PDA and B is an SPDA, can be summarized as follows:

- This inclusion problem is undecidable if B accepting by the final state[4].
- This inclusion problem is decidable if B accepting by the final state and the empty stack[2].

Some works related to the inclusion problem of context-free languages have been published recently by Minamide and Tozawa[1),5)]. In Ref. 1), Minamide and Tozawa developed two algorithms for deciding the inclusion $L(G_1) \subseteq L(G_2)$ where G_1 is a context-free grammar and G_2 is either an XML-grammar or a regular hedge grammar. Tozawa and Minamide[5] proved further that these algorithms for XML-grammars and regular hedge grammars are PTIME and 2EXPTIME, respectively. These algorithms were incorporated into the PHP string analyzer and validated several publicly available PHP programs against XHTML DTD. The languages of XML-grammars or regular hedge grammars are subclasses of generalized parenthesis languages. On the other hand, the class of languages of SPDAs contains the class of generalized parenthesis languages[2]. Thus, SPDAs are more expressive than XML-grammars and regular hedge grammars.

6. Conclusion

This paper refined the alternate stacking technique used in Greibach-Friedman's proof of the language inclusion problem $L(A) \subseteq L(B)$, where A is a PDA and B is an SPDA. The original construction encodes everything as stack symbols (in an intricate way), whereas our refinement gives a more direct product construction, and clarifies how alternate stacking works. For our construction, a proof of "liveness" is not needed, and the whole correctness proof for the decision procedure became simpler. As mentioned, the key lemma (Lemma 2) plays a crucial role in the decidable inclusion for SPDAs. However, the original proof of the key lemma[2] is indeed intricate. It would be interesting to improve the proof of this lemma.

Acknowledgments We would like to thank the anonymous referees and Dr. Yoshinao Isobe for careful reading and helpful comments in improving the paper. We are grateful to Prof. Jean Terrilon for helpful suggestions in refining the final version of the paper. Thanks also go to Dr. Nao Hirokawa for fruitful discussions on the first draft. This research is supported by the 21st Century COE program "Verifiable and Evolvable e-Society" of JAIST, funded by the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

1) Minamide, Y. and Tozawa, A.: XML Validation for Context-Free Grammars, *Proc. 4th ASIAN Symposium on Programming Languages and Systems* (*APLAS'06*), Lecture Notes in Computer Science, Vol.4279, pp.357–373, Springer-Verlag (2006).
2) Greibach, S. and Friedman, E.P.: Superdeterministic PDAs: A Subcase with a Decidable Inclusion Problem, *J. ACM*, vol.27, No.4, pp.675–700 (1980).
3) Nowotka, D. and Srba, J.: Height-Deterministic Pushdown Automata, *Proc. Mathematical Foundation of Computer Science* (*MFCS'07*), Lecture Notes in Computer Science, Vol.4708, pp.125–134, Springer-Verlag (2007).
4) Friedman, E.P. and Greibach, S.: Superdeterministic PDAs. The method of accepting does affect decision problems, *Journal of Systems and Computer Science*, vol.19, no.3, pp.79–117 (1979).
5) Tozawa, A. and Minamide, Y.: Complexity Results on Balanced Context-Free Languages, *Proc. 10th International Conference on Foundations of Software Science and Computational Structures*, Lecture Notes in Computer Science, Vol.4423, pp.346–360, Springer-Verlag (2007).
6) McNaughton, R.: Parenthesis grammars, *J. ACM*, Vol.14, No.3, pp.490–500 (1967).
7) Valiant, L.G.: Decision procedures for families of deterministic pushdown automata, Ph.D. thesis, Univeristy of Warwick, 1973.

(Received December 21, 2007)
(Accepted April 17, 2008)

Nguyen Van Tang received his M.S. in 2005 from Hanoi University of Science, Hanoi, Vietnam. His research interests include formal languages, real-time systems, and verification methodology, such as model checking and theorem proving.

Mizuhito Ogawa received his M.S. in 1985 and Ph.D degree in 2002, both from the University of Tokyo. He worked in NTT Basic Research Laboratories from 1985 until 2001, and in JST from 2002 to 2003. From 2003, he has been working at the Japan Advanced Institute of Science and Technology. His research interests include theory in rewriting, formal languages, combinatorics, and program verification methodology, such as theorem provers and model checkers.