JOURNAL OF COMPUTER AND SYSTEM SCIENCES 19, 79–117 (1979)

Superdeterministic DPDAs: The Method of Accepting Does Affect Decision Problems*

E. P. FRIEDMAN

Department of Computer Science, University of California, Los Angeles, Los Angeles, California 90024

AND

S. A. GREIBACH

Department of System Science, University of California, Los Angeles, Los Angeles, California 90024

Received September 28, 1978; revised March 7, 1979

A deterministic pushdown store automaton is superdeterministic if it is finite delay and whenever two configurations c_1 and c_1' in the same state and in reading mode are taken by the same input into two configurations c_2 and c_2' in reading mode, then c_2 and c_2' are also in the same state and the change in stack height between c_1 and c_2 is the same as that between c_1' and c_2'. Although it is decidable whether an arbitrary context-free language is included in the language accepted by a superdeterministic pushdown store automaton by final state and empty store, inclusion is undecidable for languages accepted by final state or accept mode by superdeterministic pushdown store automata. Equivalence is decidable for superdeterministic pushdown store automata (for either method of acceptance) in time $O(2^{p(n)})$, p a polynomial.

I. INTRODUCTION

The equivalence problem for deterministic pushdown store automata remains open at the present moment, although many new partial results have been announced recently [10, 11, 14, 16, 17]. The most important breakthroughs are due to Valiant who showed equivalence decidable in three subcases: nonsingular, finite turn and one counter [10, 22, 22]. Inclusion is undecidable in all three cases [3, 20]. Further equivalence results have been obtained by elaborating and extending Valiant's techniques [1, 10, 14, 17, 19] and combining them with the ideas introduced by Korenjak and Hopcroft [11, 13, 15, 16, 17, 23]. Equivalence problems for deterministic pushdown store automata are closely related to questions regarding monadic recursion schemes [2, 3, 4, 5].

We introduced in [10] a new subclass of deterministic pushdown store automata, the class of superdeterministic automata accepting by *final state and empty store* and proved

* This research was supported in part by the National Science Foundation under Grant No. MCS-78-04725.

0022-0000/79/040079-39$02.00/0

that it has a decidable inclusion problem. Furthermore, "$L(M_1) \subseteq L(M_2)$" is decidable for M_2 superdeterministic and M_1 an arbitrary nondeterministic pushdown store automaton. In this paper, we call attention to superdeterministic automata accepting by *final state alone*, or by *accept mode* in the sense of Valiant. When we make this modest change, equivalence remains decidable (as one would expect) but, surprisingly, inclusion becomes undecidable.

What has happened? A deterministic pushdown store automaton is superdeterministic if it is finite delay and if the change in stack height and state between reading modes is determined only by the state and input and not by the actual pushdown store contents. In fact, as long as the change in stack height and state between reading modes depends on the state and input alone, we can eliminate the constraints "deterministic" and "finite delay" [6]; we shall not pursue the matter here. Other families of deterministic pushdown store automata and context-free grammars have been defined by similar length uniformity conditions; for example, the ultrarealtime languages [9], the left-structured grammars of Yaffe [24], the stack uniform machines of Linna [14], and the strict restricted machines of Igarashi [25]. They all yield proper subfamilies of the family of the superdeterministic languages [6].

A superdeterministic pushdown store automaton can accept by final state and empty store the following languages, where w^R denotes the reversal of w, $| w |$ denotes the length of w and c is a symbol not in vocabulary Σ,

$$\text{EQUAL} = \{wcw^R \mid w \text{ in } \Sigma^*\}$$
$$\text{NEQUAL} = \{wcy^R \mid w, y \text{ in } \Sigma^*, w \neq y, | w | = | y |\}.$$

However, it cannot accept in that manner the languages

$$\text{UNEQUAL} = \{wcy^R \mid w, y \text{ in } \Sigma^*, w \neq y\}$$
$$\text{LESSTHAN} = \{wcy^R \mid w, y \text{ in } \Sigma^*, w \neq y, | y | \leqslant | w |\}$$
$$\text{GREATERTHAN} = \{wcy^R \mid w, y \text{ in } \Sigma^*, w \neq y, | y | \geqslant | w |\}.$$

The latter three languages are accepted by final state by realtime deterministic pushdown store automata, while GREATERTHAN is nonsingular and LESSTHAN is accepted by final state by a superdeterministic pushdown store automaton. Standard proofs of the undecidability of inclusion consider the inclusion of a language akin to EQUAL in a language similar to UNEQUAL (both languages suitably encoding the Post Correspondence Problem). For nonsingular or simple (one state, acceptance by empty store) deterministic pushdown store automata, a language similar to GREATERTHAN is used instead of UNEQUAL [2, 3, 20]. In Section 3 of this paper, we show that a language "like" LESSTHAN also suffices to yield the undecidability of inclusion; however, EQUAL and NEQUAL alone do not. The least AFDL containing EQUAL and NEQUAL is properly contained in the family of languages accepted by final state and empty store by superdeterministic *pda*, and so has a decidable inclusion problem [10].

Thus, what changes a decidable to an undecidable problem in this case is, so to speak, the "power of inequality". Loosely speaking, as long as we can match two strings only

when they are of equal length, inclusion remains decidable. But if we can match them also when the second may be shorter ("$\leqslant$"), we obtain an undecidable problem and similarly if the second may be longer ("$\geqslant$") or may be either longer or shorter ("$\neq$").

As far as we know, this is the first problem for pushdown store automata in which the method of acceptance makes a difference between a decidable and an undecidable problem. We also show that equivalence remains decidable for superdeterministic pushdown store automata accepting by accept mode. This adds to the growing list of classes of deterministic pushdown store automata (first noticed by Friedman [2] and Valiant [20]) for which inclusion is undecidable and equivalence is decidable.

We give the essential notation in Section 2 and establish the undecidability of inclusion in Section 3. We use a variant of the Post Correspondence Problem [18], modifying the ideas of [3] to use "$\leqslant$" instead of "$\geqslant$". We observe that the same construction applies to stack uniform automata so inclusion is also undecidable for Linna's stack uniform deterministic pushdown store automata accepting by final state.

In Section 4, we establish the decidability of equivalence for superdeterministic pushdown store automata accepting by accept mode using variants of Valiant's "alternate stacking" or "parallel stacking" technique. "Alternate stacking" involves simulating two machines M_1 and M_2 with one machine M whose stack contents $u_1v_1 \cdots u_nv_n$ encode the stacks $u_1 \cdots u_n$ and $v_1 \cdots v_n$ of M_1 and M_2. In general, the simulating machine may not be implementable by a pushdown store automaton. Alternating stacking "succeeds" if the stacks can be interwoven in such a way that the top segment or segments remain(s) uniformly bounded. There are usually two steps in obtaining such a result for a class $\mathscr{C}$ of deterministic pushdown store automata, as is done in [10, 14, 17, 19, 20].

Call configurations c_1 and c_1' in M_1 and c_2 and c_2' in M_2 *paired* if, for some inputs x and y and $i = 1, 2$, c_i is reached from the initial configuration of M_i on input x, c_i' is reached from c_i on input y and some accepting configuration is reachable from c_i' for some input string. In particular, if M_1 and M_2 are equivalent, then c_1 is equivalent to c_2 and c_1' to c_2' and some input xyz is accepted by both machines.

The first step is to use special features of machines M_1 and M_2 in $\mathscr{C}$ to show that there exists a k, depending on M_1 and M_2 such that, if M_1 and M_2 are equivalent, then some condition $Q(c_1, c_1', c_2, c_2', k)$ holds for all paired configurations. Then one gives a construction of a simulating pushdown store automaton $M = M(M_1, M_2, k)$ from M_1, M_2 and k such that, if the desired segment bound is violated, then $Q(c_1, c_1', c_2, c_2', k)$ fails for some paired configurations. This second property generally depends on the construction and not on $\mathscr{C}$, but is only of interest when combined with the first step. One can then let M accept only when either a word is found to be accepted by one machine and not the other, or the segment bound is violated. Finally, Valiant's arguments show that M_1 and M_2 are equivalent if and only if there is a k such that the language accepted by $M(M_1, M_2, k)$ is empty, and hence equivalence is decidable for $\mathscr{C}$.

We define two properties to play the role of Q, namely, the matched pushing property and the matched popping property, and show that equivalence is decidable for any class of deterministic pushdown store automata with either the matched pushing property or the matched popping property. If a class of deterministic pushdown store automata has the matched pushing property, then stack increases (pushes) in a machine M_1 cannot

lag too far behind those in an equivalent machine M_2. This means that no input can take equivalent configurations in M_1 and M_2 to "live" configurations (from which some string may be accepted) such that the M_1 computation has no net stack increase while the computation in M_2 adds many "useful" symbols (whose presence on the stack affects acceptance). The matched popping property on the other hand says that stack decreases (pops) in M_1 cannot lag too far behind those in M_2; i.e., no input can take equivalent configurations in M_1 and M_2 to "live" configurations such that the computation in M_1 has no net stack decrease and ends in a configuration with many "useful" symbols on the pushdown store while the computation in M_2 pops many symbols from the pushdown store.

Valiant showed that the class of nonsingular deterministic pushdown store automata has the matched pushing property [20] and Linna showed this for the class of stack uniform deterministic pushdown store automata [14] while Oyamaguchi, Honda and Inagaki recently showed that the class of strict deterministic realtime pushdown store automata has the matched pushing property and a form of the matched popping property [17]. In the Appendix, we show that the class of superdeterministic pushdown store automata accepting by accept mode has both the matched pushing and the matched popping properties. In Section 4, we use the precise definition of "many useful symbols" found in the Appendix to establish an order of magnitude bound of $2^{p(n)}$ ($p(n)$ a polynomial) on the time complexity of deciding equivalence for superdeterministic pushdown store automata accepting by accept mode.

2. Notation

We use a variant of Valiant's notation. Let e denote the *empty word*, and $|x|$, the *length of a word* x (so $|e| = 0$).

A *pushdown store automaton* (*pda*) is denoted by $M = (K, \Sigma, \Gamma, H, q_0, Z_0, F)$ where K is a finite set of *states*, Σ is a finite set of *input* symbols, Γ is a finite set of *pushdown store* (*pds*) symbols, q_0 in K is the *initial state*, Z_0 in Γ is the *initial pushdown store* symbol, $F \subseteq K \times (\Gamma \cup \{e\})$, is the set of *accepting or final modes* and H, the set of *transitions or rules*, is a finite subset of $K \times \Gamma \times (\Sigma \cup \{e\}) \times K \times \Gamma^*$.

We write (q, A, a, p, y) in H as $(q, A) \xrightarrow{a} (p, y)$ and call (q, A) the *mode* of the rule with input a; if $a = e$, this is an *e-rule*. A pair (q, yA), q in K, A in Γ, y in Γ^* is a *configuration with mode* (q, A) while (q, e) is a *configuration with mode* (q, e). For a configuration $c = (q, y)$, the state of c is *state*$(c) = q$ and the *stack height of* c is $|c| = |y|$ and the mode is denoted *mode*(c). If no rules are defined for mode (q, A) then it is a *blocking mode*; if no e-rule is defined for mode (q, A) and it is not a blocking mode, then (q, A) is a *reading mode*.

If $(q, A) \xrightarrow{a} (p, y)$ is in H, then we write $(q, uA) \xrightarrow{a} (p, uy)$ for any u in Γ^* and call it a *1-step computation*. If $c_1 \xrightarrow{u} c_2$ and $c_2 \xrightarrow{v} c_3$, write $c_1 \xrightarrow{uv} c_3$ and call it a *computation*. For any configuration c, we write $c \xrightarrow{e} c$ and call it a *0-step computation*.

The language accepted from configuration c by final state and empty store is

$$L(c) = \{w \text{ in } \Sigma^* \mid \text{for some } (f, e) \text{ in } F,\ c \xrightarrow{w} (f, e)\}$$

and the language accepted by *accept mode* is

$$T(c) = \{w \text{ in } \Sigma^* \mid c \xrightarrow{w} c', \text{mode}(c') \text{ in } F\}.$$

The language M accepts by accept mode is $T(M) = T((q_0, Z_0))$ and, by final state and empty store, is $L(M) = L((q_0, Z_0))$.

Two configurations c_1 and c_2 are

$$\begin{aligned} &\textit{ℓ-equivalent}, && c_1 \overset{\ell}{=} c_2, && \text{if} \quad L(c_1) = L(c_2), \\ &\textit{t-equivalent}, && c_1 \overset{t}{=} c_2, && \text{if} \quad T(c_1) = T(c_2). \end{aligned}$$

Two machines are ℓ-equivalent (t-equivalent) if their initial configurations are. Normally, during this paper, we assume that "equivalence" without a modifier means "t-equivalence"; we use "ℓ-equivalence" or "t-equivalence" only for emphasis. Similarly, by the inclusion problem, we mean the t-inclusion problem: $T(M_1) \subseteq T(M_2)$.

We are primarily concerned with *pdas* which are either realtime or finite delay. A *pda* M is of *delay* d if, whenever there is a series of 1-step computations

$$c_1 \xrightarrow{e} c_2 \xrightarrow{e} c_3 \cdots \xrightarrow{e} c_n,$$

then $n - 1 \leqslant d$ (i.e., at most d e-rules can be applied in a row to any configuration). It is *finite delay* if it is of delay d for some $d \geqslant 0$. It is *realtime* if it is of delay 0; that is, if there are no e-rules defined.

A *pda* $M = (K, \Sigma, \Gamma, H, q_0, Z_0, F)$ is a *deterministic pushdown store automaton* (*dpda*) if, for each mode (q, A), either (1) there is no e-rule with mode (q, A) and for each a in Σ there is at most one rule with mode (q, A) and input a, or (2) there is exactly one rule with mode (q, A) and this is an e-rule.

Now we can define superdeterministic formally.

DEFINITION. A deterministic pushdown automaton $M = (K, \Sigma, \Gamma, H, q_0, Z_0, F)$ is *superdeterministic* if it is finite delay and, for all accessible configurations in reading mode c_1, c_2, c_1', c_2' and all a in Σ, if

$$\text{state}(c_1) = \text{state}(c_2)$$

and

$$c_1 \xrightarrow{a} c_1' \qquad \text{and} \qquad c_2 \xrightarrow{a} c_2',$$

then $\text{state}(c_1') = \text{state}(c_2')$, and $|c_1| - |c_1'| = |c_2| - |c_2'|$.

If M is superdeterministic and in a configuration c in reading mode, changes in the state and stack height as M passes to other configurations in reading mode depend only on state(c) and the input, not on the actual *pds* contents of M. However, the contents of the top of the stack may vary. If M "doesn't like" the top of its stack, it can block but, if it reads a new input, it does so in a state and with a stack change independent of the stack contents at the previous input. When accepting by accept mode, it can also, by varying the top stack symbol, accept or not accept ending in the same state. Thus, M is limited in the way it can pass along information as to its actual stack contents from one input to another. The limitation on information transfer and the uniformity condition

on stack height are the basis of our proof of the decidability of equivalence. But the ability to pass some information down the stack and accept or not accept along the way yields the undecidability of inclusion.

DEFINITION. A language L is *ℓ-superdeterministic* if there is a superdeterministic *dpda* M such that either $L = L(M)$ or $L\$ = L(M)$ for some symbol \$ and *t-superdeterministic* if there is a superdeterministic *dpda* M such that $L = T(M)$ or $L\$ = T(M)$ for some symbol \$.

In defining the classes of ℓ- and t-superdeterministic languages, we allow M, in effect, to have an endmarker where useful; this does not affect the decidability or undecidability of inclusion or equivalence.

We need some further special notation regarding configurations and computations. If we have a series of 1-step computations

$$c_1 \xrightarrow{a_1} c_2 \xrightarrow{a_2} c_3 \xrightarrow{a_3} \cdots \xrightarrow{a_n} c_{n+1},$$

with $|c_1| \leqslant |c_i|$, $1 \leqslant i \leqslant n+1$, we write $c_1 \uparrow (a_1 \cdots a_n)\, c_{n+1}$. This is called a *stacking computation*. On the other hand, if $|c_i| \geqslant |c_{n+1}|$, $1 \leqslant i \leqslant n$, we write $c_1 \downarrow (a_1 \cdots a_n)\, c_{n+1}$ and call it a *popping configuration*. Notice that, if $|c_1| = |c_{n+1}|$, the computation could be both stacking and popping.

The *initial configuration* is denoted by $c_0 = (q_0, Z_0)$. A configuration c is *accessible from a configuration* c' if $c' \xrightarrow{w} c$ for some w in Σ^*, *accessible* if it is accessible from c_0, *accepting* if mode(c) is in F, *live* if $c \xrightarrow{w} \bar{c}$ for some w in Σ^* and accepting configuration $\bar{c}$ and *blocked* if mode(c) is blocking. For a in $\Sigma \cup \{e\}$, we say that c is *a-live* if $c \xrightarrow{aw} \bar{c}$ for some w in Σ^* and accepting configuration $\bar{c}$, and *a-blocked* if there is no rule with mode(c) and input a. A configuration which is not live is *dead*.

Let c be a live configuration in a *dpda* M accepting by accept mode. It is possible to accept without reading (i.e., popping) all the symbols of the pushdown store of c. In fact, we can have $c = (p, y_1 y_2)$ such that no *accepting* computation from c reads any symbol in y_1 and so $T(c) = T((p, y_2))$. In such a situation, the machine may as well put a "barrier" below y_2 and quit if this barrier symbol is ever read (since then no accepting configuration can be reached). This concept is crucial to the constructions in Section 4. We need to measure how far down the stack M can pop and still be in a live configuration.

For c live, let

$$\text{LIVE}(c) = \text{Max}\{|c| - |c'| \mid c' \text{ is live and accessible from } c\}.$$

For completeness, if c is dead, let $\text{LIVE}(c) = -1$. Thus, if $c = (p, y_1 y_2)$, $|y_2| > \text{LIVE}(c)$ and $c \xrightarrow{w} (p', y_1)$, (p', y_1) must be dead.

If in each rule (q, A, a, p, y) we have $|y| \leqslant 2$, then M is called *1-increasing*; thus a step of a 1-increasing *pda* increase the stack height by at most 1.

3. THE INCLUSION PROBLEM

In this section, we concentrate on showing that "$T(M_1) \subseteq T(M_2)$" is undecidable for M_1 and M_2 arbitrary superdeterministic *dpdas*. Our strategy is first to define a variant of

the Post Correspondence Problem [18] and then to show its undecidability for any arbitrary pair of lists of strings, say (X, Y), satisfying certain requirements. Next, given any such pair, we show how to construct two superdeterministic *dpdas* $M_1(X)$ and $M_2(Y)$ such that $T(M_1(X)) \subseteq T(M_2(Y))$ if and only if there is no solution to our variant of the Post Correspondence Problem for pair (X, Y). Hence, the inclusion problem for superdeterministic *dpdas* accepting by accept mode is shown to be undecidable.

We define a *Variant Correspondence Problem* (*VCP*) as follows: Let X and Y be two lists of $n > 1$ nonempty strings over finite alphabet Σ, denoted by $X = \langle x_1, ..., x_n \rangle$ and $Y = \langle y_1, ..., y_n \rangle$, where $|x_i| \leqslant |y_i|$ for each $2 \leqslant i \leqslant n$, and $|x_1| < |y_1|$. Call a pair of such lists (X, Y) a *VCP pair*. We say that the *VCP* for *VCP* pair (X, Y) has a *partial solution* $\langle i_1, ..., i_t \rangle$ if $2 \leqslant i_1, ..., i_t \leqslant n$ and $x_1 x_{i_1} \cdots x_{i_t}$ is a prefix[1] of $y_1 y_{i_1} \cdots y_{i_t}$. For symbol a in Σ, we say that the *VCP* for *VCP* pair (X, Y) has an *a-marked solution* $\langle i_1, ..., i_t \rangle$ if $x_1 x_{i_1} \cdots x_{i_t} a$ is a prefix of $y_1 y_{i_1} \cdots y_{i_t}$, and $2 \leqslant i_1, ..., i_t \leqslant n$.

In Lemma 3.1, we establish that it is undecidable whether there is an a-marked solution to the *VCP* for an arbitrary *VCP* pair of lists (X, Y) and marked symbol a. To such an end we provide a mechanism for building two lists of strings $X(M, w)$ and $Y(M, w)$ from any given Turing Machine M and input tape w, and show that M halts and accepts input w if and only if there is a q_f-marked solution to the *VCP* for the *VCP* pair of lists $(X(M, w), Y(M, w))$ and designated marked symbol q_f.

We use the following definitions and notation for a Turing machine.

A *Turing machine* is denoted by a 6-tuple $M = (K, \Sigma, \Gamma, \delta, q_0, q_f)$, where K is a finite set of *states*, Γ is a finite set of *tape symbols* with symbol $\not b$ in Γ designated as the *blank*, $\Sigma \subseteq \Gamma$ is a finite set of *input symbols* with $\not b$ not in Σ, q_0 in K is the *start state*, q_f in K is the *final state* with $q_0 \neq q_f$, and $\delta: (K - \{q_f\}) \times \Gamma \to K \times (\Gamma - \{\not b\}) \times \{L, R\}$ is the *transition function*. We assume that $K \cap \Gamma = \varnothing$.

A *configuration* of M is denoted by the string $\alpha q \beta$, where q in K is the *current state* of M and $\alpha\beta$ in $(\Gamma - \{\not b\})^*$ is the *nonblank portion of the tape*. The tape head is situated so that the next tape symbol that will be read is the leftmost symbol of β if $\beta \neq e$, and the next symbol is $\not b$ if $\beta = e$. The tape is infinite only to the right, and the tape head is not allowed to "fall off" the left end of the tape.

We define the relation $\vdash_M$ (or $\vdash$ when M is clear from the context) on configurations of M as follows. For all A, B, C in $(\Gamma - \{\not b\})$, α, β in $(\Gamma - \{\not b\})^*$, p, q in K,

(1) if $\delta(q, A) = (p, B, R)$, then $\alpha q A \beta \vdash_M \alpha B p \beta$
(2) if $\delta(q, A) = (p, B, L)$, then $\alpha C q a \beta \vdash_M \alpha p C B \beta$
(3) if $\delta(q, \not b) = (p, B, R)$, then $\alpha q \vdash_M \alpha B p$;
(4) if $\delta(q, \not b) = (p, B, L)$, then $\alpha C q \vdash_M \alpha p C B$.

We let $\overset{*}{\vdash}_M$ (or $\overset{*}{\vdash}$ when M is understood) denote the transitive reflexive closure of $\vdash_M$.

Turing machine M is said to *halt and accept* input tape w if it ever gets into some configuration of the form $\alpha q_f \beta$. No next move is possible from $\alpha q_f \beta$, but a next move is always possible from $\alpha q \beta$ when $q \neq q_f$, unless M tries to move off the left end of the tape.

[1] For any finite alphabet Σ, any strings u, v in Σ^*, we say that u is a *prefix* of v if there is some string z in Σ^* such that $v = uz$. String u is a *proper prefix* of v if $u \neq z$. It is a *suffix* of v if $v = zu$ for some z in Σ^*.

The *language accepted by Turing machine* M is defined to be

$$L(M) = \{w \text{ in } \Sigma^* \mid q_0 w \overset{*}{\underset{M}{\vdash}} \alpha q_f \beta \text{ for some } \alpha, \beta \text{ in } (\Gamma - \{\not b\})^*\}.$$

The following question is called the *halting problem for Turing machines*: Given any Turing machine M and input tape w, does M halt and accept w? It is well-known that the halting problem for Turing machines is undecidable, even in this formulation.

We now show that it is undecidable whether there is an a-marked solution to the *VCP* for two arbitrary lists of strings and marked symbol a.

LEMMA 3.1. *It is undecidable whether there is an a-marked solution to the VCP for an arbitrary VCP pair of lists* (X, Y) *and marked symbol a.*

Proof. Let $M = (K, \Sigma, \Gamma, \delta, q_0, q_f)$ be any Turing machine and w in Σ^* be any input tape. We shall build two finite lists $X(M, w)$ and $Y(M, w)$ of strings such that M halts and accepts w if and only if there is a q_f-marked slution to the *VCP* of *VCP* pair $(X(M, w), Y(M, w))$. Only the first string from each list is numbered (number 1), as the ordering of the other strings is irrelevant. Our construction is based on one used by Hopcroft and Ullman [12] to show the undecidability of the Modified Post Correspondence Problem, although ours differs from theirs at one crucial point. Hopcroft and Ullman's lists, say $\bar{X}(M, w) = \langle \bar{x}_1, ..., \bar{x}_n \rangle$ and $\bar{Y}(M, w) = \langle \bar{y}_1, ..., \bar{y}_n \rangle$, are constructed so the Turing machine M halts and accepts input w if and only if there is a sequence of integers, $2 \leqslant i_1, ..., i_t \leqslant n$, such that $\bar{x}_1 \bar{x}_{i_1} \cdots \bar{x}_{i_t} = \bar{y}_1 \bar{y}_{i_1} \cdots \bar{y}_{i_t}$. There is never any sequence of integers for our lists, $X(M, w) = \langle x_1, ..., x_n \rangle$ and $Y(M, w) = \langle y_1, ..., y_n \rangle$, such that $x_1 x_{i_1} \cdots x_{i_t} = y_1 y_{i_1} \cdots y_{i_t}$. Nevertheless, we shall see that M halts and accepts w if and only if there is a sequence such that $x_1 x_{i_1} \cdots x_{i_t} q_f$ is a prefix of $y_1 y_{i_1} \cdots y_{i_t}$. In other words, M halts and accepts w if and only if there is a q_f-marked solution to the *VCP* for *VCP* pair $(X(M, w), Y(M, w))$.

Let $\#$ be a new symbol not in the alphabet of M. The two lists $X(M, w)$ and $Y(M, w)$ are constructed as shown below.

	LIST $X(M, w)$	LIST $Y(M, w)$	
I. INITIALIZATION PAIR (pair 1)	$\#$	$\#q_0 w \#$	
II. COPY PAIRS			
for each A in $(\Gamma - \{\not b\})$	A	A	
	$\#$	$\#$	
III. NEXT MOVE PAIRS			
For each p in K, q in $K - \{q_f\}$,			
A, B, C in $(\Gamma - \{\not b\})$	qA	Bp,	if $\delta(q, A) = (p, B, R)$
	CqA	pCB,	if $\delta(q, A) = (p, B, L)$
	$q\#$	$Bp\#$	if $\delta(q, \not b) = (p, B, R)$
	$Cq\#$	$pCB\#$,	if $\delta(q, \not b) = (p, B, L)$

We see that lists $X(M, w)$ and $Y(M, w)$ have the same length. Let n denote the number of strings in each list; we let $\langle x_1, ..., x_n \rangle$ and $\langle y_1, ..., y_n \rangle$ denote lists $X(M, w)$ and $Y(M, w)$, respectively. Clearly, $x_1 = \#$ and $y_1 = \#q_0w\#$. Each string x_i, $1 \leqslant i \leqslant n$, is composed of symbols from the alphabet $(K - \{q_f\}) \cup (\Gamma - \{\not b\}) \cup \{\#\}$, and each string y_i is composed from symbols in $K \cup (\Gamma - \{\not b\}) \cup \{\#\}$. It is important to remember that q_f is not used in list $X(M, w)$, and that each string x_i is no longer than the corresponding string y_i, and $| x_1 | < | y_1 |$.

We must now prove that Turing machine M halts and accepts w if and only if there is a q_f-marked solution for lists $X(M, w)$ and $Y(M, w)$.

First, suppose that M halts and accepts string w. Then the computation of M on input w must have the form

$$q_0w \vdash \alpha_1q_1\beta_1 \vdash \cdots \vdash \alpha_{k-1}q_{k-1}\beta_{k-1} \vdash \alpha_kq_f\beta_k$$

for some $k \geqslant 1$ and $q_0, ..., q_{k-1} \neq q_f$. The same justification used in [12] establishes the existence of integers $2 \leqslant i_1, ..., i_s \leqslant n$ such that

$$x_1x_{i_1} \cdots x_{i_s} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-1}\#$$
$$y_1y_{i_1} \cdots y_{i_s} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-1}\#\alpha_kq_f\beta_k\#$$

We can then choose a sequence of copy pairs indicated by integers $j_1, ..., j_t$ to get

$$x_1x_{i_1} \cdots x_{i_s}x_{j_1} \cdots x_{j_t} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-1}\#\alpha_k$$
$$y_1y_{i_1} \cdots y_{i_s}y_{j_1} \cdots y_{j_t} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-1}\#\alpha_kq_f\beta_k\#\alpha_k$$

Since $x_1x_{i_1} \cdots x_{i_s}x_{j_1} \cdots x_{j_t}q_f$ is a proper prefix of $y_1y_{i_1} \cdots y_{i_s}y_{i_1} \cdots y_{j_t}$, the sequence of integers $\langle i_1, ..., i_s, j_1, ..., j_t \rangle$ is a q_f-marked solution to the *VCP* for *VCP* pair $(X(M, w), Y(M, w))$.

On the other hand, if M does not halt and accept input w, then the only partial solutions $\langle i_1, ..., i_t \rangle$ of the *VCP* for *VCP* pair $(X(M, w), Y(M, w))$ must have the form

$$x_1x_{i_1} \cdots x_{i_t} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-1}\#u$$
$$y_1y_{i_1} \cdots y_{i_t} = \#q_0w\#\alpha_1q_1\beta_1\# \cdots \#\alpha_{k-1}q_{k-1}\beta_{k-} \#\alpha_kq_k\beta_k\#v$$

where

$$q_0w \vdash \alpha_1q_1\beta_1 \vdash \cdots \vdash \alpha_{k-1}q_{k-1}\beta_{k-1} \vdash \alpha_kq_k\beta_k$$

for some $k \geqslant 0$, $q_0, ..., q_{k-1} \neq q_f$, and u is a prefix of $\alpha_kq_k\beta_k$.

Since the symbol q_f does not appear in string $\alpha_kq_k\beta_k\#v$, $\langle i_1, ..., i_t \rangle$ is not q_f-marked solution to the *VCP* for *VCP* pair $(X(M, w), Y(M, w))$. Thus, there can be no q_f-marked solution to the *VCP* pair $(X(M, w), Y(M, w))$. ∎

We are now ready for the main theorem of this section which establishes the undecidability of the inclusion problem for superdeterministic *dpdas* accepting by accept mode.

THEOREM 3.2. *It is undecidable whether "$T(M_1) \subseteq T(M_2)$" for arbitrary superdeterministic dpdas M_1 and M_2.*

Proof. Let Σ any finite alphabet, a any symbol in Σ, and $X = \langle x_1, \ldots, x_n \rangle$, $Y = \langle y_1, \ldots, y_n \rangle$ any two lists of $n > 1$ strings in Σ^+ with $|x_i| \leqslant |y_i|$ for $1 \leqslant i \leqslant n$, and $|x_1| < |y_1|$. We shall build two superdeterministic *dpdas* $M_1(X)$ and $M_2(Y)$ such that $T(M_1(X)) \subseteq T(M_2(Y))$ if and only if there is no a-marked solution to the *VCP* for *VCP* pair (X, Y).

For each integer i, $1 \leqslant i \leqslant n$, let f_i be a new symbol. Symbol f_i should be regarded as an encoding for integer i.

We first construct the superdeterministic *dpda* $M_1(X)$ and make the claim that $T(M_1(X)) = \{f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a \mid t \geqslant 1, 2 \leqslant i_1, \ldots, i_t \leqslant n\}$.

Informally, $M_1(X)$ works by reading the encodings of integers (i.e., the f_i) and pushing the associated strings from list X (i.e., the x_i) onto the pushdown store. Then, as $M_1(X)$ reads symbols from Σ, it pops when the symbol at the top of the store matches the input symbol, but blocks otherwise. $M_1(X)$ accepts the tape if when the final symbol a is read, it matches with the only symbol remaining on the store (also a), thereby emptying the store to accept by accept mode (q_1, e).

A formal definition of $M_1(X)$ follows. Let $M_1(X) = (\{q_0, q_1\}, \Sigma \cup \{f_1, \ldots, f_n\}, \Sigma \cup \{Z_0\}, H_1, q_0, Z_0, \{(q_1, e)\})$, for a new *pds* symbol Z_0, with the transition set H_1 defined below.

I. Insure that the encoding of the integers has the proper form $f_{i_t} \cdots f_{i_1} f_1$ for $t \geqslant 1$ and $2 \leqslant i_1, \ldots, i_t \leqslant n$; push the associated x_i's onto the pushdown store. Initially, place symbol a on the bottom of the store.

For all σ in Σ, $2 \leqslant i \leqslant n$, include in H_1 the rules

$$(q_0, Z_0, f_i, q_0, a x_i^R),$$
$$(q_0, \sigma, f_i, q_0, \sigma x_i^R),$$
$$(q_0, \sigma, f_1, q_1, \sigma x_1^R).$$

II. The next portion of an accepted tape must be a string of symbols over Σ matching the part pushed onto the store.

For every σ in Σ, include in H_1 the rule

$$(q_1, \sigma, \sigma, q_1, e).$$

$M_1(X)$ is clearly superdeterministic. Moreover its construction is so straightforward that we make the claim without proof that

$$T(M_1(X)) = \{f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a \mid t \geqslant 1 \text{ and } 2 \leqslant i_1, \ldots, i_t \leqslant n\}.$$

We now construct the superdeterministic *dpda* $M_2(Y)$ and make the claim that for any $z = f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a$ accepted by $M_1(X)$, input z is accepted by $M_2(Y)$ if and only if $x_1 x_{i_1} \cdots x_{i_t} a$ is not a prefix of $y_1 y_{i_1} \cdots y_{i_t}$.

We want $M_2(Y)$ to start off in a manner similar to $M_1(X)$ by reading the encodings of

integers (the f_i), but now pushing the associated y_i's onto the pushdown store. Next, as $M_2(Y)$ reads symbols from Σ, it pops when the symbol at the top of the pushdown store matches the input symbol. Unlike $M_1(X)$, it does not block when a mismatch occurs, since we want $M_2(Y)$ eventually to accept such strings where a mismatch occurs. We record the fact that a mismatch has occurred by popping the symbol and then using a single e-rule to mark the next topmost symbol on the pushdown as a new symbol C by changing whatever symbol was there before to C. This marked pushdown symbol C keeps percolating down the stack as we pop.

$M_2(Y)$ rejects in mode (q_1, σ) for σ in Σ whenever we have successfully matched all the input symbols and the pushdown store is not empty. On the other hand, $M_2(Y)$ accepts as soon as a mismatch is found and continues to accept thereafter. Since the marked symbol C gets carried down through the pushdown store while the input symbols after the mismatch are processed, and (q_1, C) is an accept mode, we can easily accept strings in which the length of the input tape remaining to be read after the mismatch is no longer than the length of the pushdown store at the time of mismatch. Inputs that are "too long" would have to cause a block, due to the superdeterministic requirement. Luckily, our lists are forced to have $|x_j| \leqslant |y_j|$ for each $1 \leqslant j \leqslant n$ so that, whenever a mismatch occurs while $M_2(Y)$ is processing a string of the form $f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a$, we are ensured that there are more symbols left on the pushdown store than there are input symbols remaining to be read.

A formal definition of $M_2(Y)$ follows. Let $M_2(Y) = (\{q_0, q_1, q_2\}, \Sigma \cup \{f_1, \ldots, f_n\}, \Sigma \cup \{Z_0, C\}, H_2, q_0, Z_0, \{(q_1, C)\})$, for new *pds* symbols Z_0, C, with the transition set H_2 defined below.

I. Insure that the encoding of the integers has the proper form $f_{i_t} \cdots f_{i_1} f_1$ while pushing the associated y_i's onto the store.

For all i with $2 \leqslant i \leqslant n$, and σ in Σ, include in H_2 the rules

$$(q_0, Z_0, f_i, q_0, ay_i{}^R),$$
$$(q_0, \sigma, f_i, q_0, \sigma y_i{}^R),$$
$$(q_0, \sigma, f_1, q_1, \sigma y_1{}^R).$$

II. Match input symbols in Σ against the *pds* symbols. Pop and remain in state q_1 if a match occurs. If a mismatch occurs, pop into state q_2 and then use an e-rule to return to state q_1 while changing the new *pds* symbol to a C. "Percolate" this C down the store so that we can remember to accept the input tape. The accept mode is (q_1, C), which occurs only after a mismatch has been detected.

For all σ, γ in Σ, include in H_2 the rules

$$(q_1, \sigma, \sigma, q_1, e),$$
$$(q_1, \gamma, \sigma, q_2, e), \qquad \text{if} \quad \sigma \neq \gamma,$$
$$(q_2, \sigma, e, q_1, C),$$
$$(q_1, C, \sigma, q_2, e).$$

Rules in section I certainly preserve superdeterminism. We can also verify that rules in II are superdeterministic, since if M reads a symbol from Σ while in state q_1 it either (1) pops the topmost symbol and remains in state q_1 or (2) pops the topmost symbol and changes to state q_2. In state q_2, if the store is not empty, a single e-move is allowed, which changes whatever symbol is on top to a C and goes into reading mode in state q_1; if the store is empty, it is by definition not in reading mode. Therefore, the net effect of reading a symbol when in state q_1 is to decrease the height of the pushdown store by 1 and to return to state q_1 unless the machine blocks. Thus, $M_2(Y)$ satisfies the requirement of superdeterminism, since we only check stack height differences and state changes for configurations in reading mode.

We do not need to specify precisely the format for words accepted by $M_2(Y)$. It is sufficient to prove that $M_2(Y)$ accepts all of the words accepted by $M_1(X)$ except any prefixes of $f_{i_t} \cdots f_{i_1} f_1 y_1 y_{i_1} \cdots y_{i_t}$. $M_2(Y)$ also accepts words that are not accepted by $M_1(X)$, but these words are not relevant to the arguments that follow.

Claim. $T(M_2(Y))$ includes all the words in

$$T(M_1(X)) = \{f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a \mid t \geqslant 1 \text{ and } 2 \leqslant i_1, \ldots, i_t \leqslant n\}$$

except those (if any) where $x_1 x_{i_1} \cdots x_{i_t} a$ is a prefix of $y_1 y_{i_1} \cdots y_{i_t}$.

Consider any string z in $T(M_1(X))$. Then z is of the form $f_{i_t} \cdots f_{i_1} f_1 x_1 x_{i_1} \cdots x_{i_t} a$ where $t \geqslant 1$ and $2 \leqslant i_1, \ldots, i_t \leqslant n$.

Recall that lists X and Y were required to have $|x_j| \leqslant |y_j|$ for each $2 \leqslant j \leqslant n$, and $|x_1| < |y_1|$. Therefore, $|x_1 x_{i_1} \cdots x_{i_t} a| \leqslant |y_1 y_{i_1} \cdots y_{i_t}|$. We have two cases to consider.

(i) $x_1 x_{i_1} \cdots x_{i_t} a$ is a prefix of $y_1 y_{i_1} \cdots y_{i_t}$. That is, $y_1 y_{i_1} \cdots y_{i_t} = x_1 x_{i_1} \cdots x_{i_t} au$ for some string u in Σ^*.

(ii) The strings only match partway through. That is, for some σ, γ in Σ with $\sigma \neq \gamma$, and some u, v, w in Σ^*, with $|v| \leqslant |w|$ we have

$$x_1 x_{i_1} \cdots x_{i_t} a = u\sigma v \qquad \text{and}$$

$$y_1 y_{i_1} \cdots y_{i_t} = u\gamma w$$

Case 1. We first consider the case where z satisfies condition (i). Then

$$(q_0, Z_0) \xrightarrow{f_{i_t} \cdots f_{i_1} f_1} (q_1, a y_{i_t}^R \cdots y_{i_1}^R y_1^R) = (q_1, a u^R a x_{i_t}^R \cdots x_{i_1}^R x_1^R)$$

$$\xrightarrow{x_1 x_{i_1} \cdots x_{i_t} a} (q_1, au^R)$$

Thus, z is not in $T(M_2(Y))$, since the accept mode (q_1, C) is not the mode of configuration (q_1, au^R).

Case 2. Now consider the case where z satisfies condition (ii). Then $x_1x_{i_1} \cdots x_{i_t}a = u\sigma v$ and

$$(q_0, Z_0) \xrightarrow{f_{i_t} \cdots f_{i_1} f_1} (q_1, ay_{i_t}^R \cdots y_{i_1}^R y_1^R) = (q_1, awv^R \gamma u^R)$$

$$\xrightarrow{u} (q_1, awv^R\gamma)$$

$$\xrightarrow{\sigma} (q_2, awv^R)$$

We have four subcases to consider.

(a) If $v = w = e$, then $(q_2, awv^R) = (q_2, a) \xrightarrow{e} (q_1, C)$

(b) If $v = e \neq w$, then $w = Zw_1$ for some Z in Σ, w_1 in Σ^*.

$$(q_2, awv^R) = (q_2, aw_1{}^RZ) \xrightarrow{e} (q_1, aw_1{}^RC)$$

(c) If $e \neq v$ and $|v| = |w|$, then $w = Zw_1$ for some Z in Σ, w_1 in Σ^*,

$$(q_2, awv^R) = (q_2, aw_1{}^RZ) \xrightarrow{e} (q_1, aw_1{}^RC) \xrightarrow{v} (q_2, a) \xrightarrow{e} (q_1, C)$$

(d) If $v \neq e$ and $|v| < |w|$, then $w = Zw_1Yw_2$, for some Y, Z in Σ, w_1, w_2 in Σ^*, with $|Zw_1| = |v|$. In this case,

$$(q_2, awv^R) = (q_2, aw_2{}^RYw_1{}^RZ) \xrightarrow{e} (q_1, aw_2{}^RYw_1{}^RC)$$

$$\xrightarrow{v} (q_2, aw_2{}^RY) \xrightarrow{e} (q_1, aw_2{}^RC)$$

In all four subcases shown above, z is in $T(M_2(Y))$, since the mode of the configuration resulting after z has been completely read is the accept mode (q_1, C).

We summarize the preceding statements about languages accepted by accept mode by $M_1(X)$ and $M_2(Y)$ as follows: $T(M_1(X)) \subseteq T(M_2(Y))$ if and only if $M_1(X)$ does not accept any string of the form $f_{i_t} \cdots f_{i_1}f_1x_1x_{i_1} \cdots x_{i_t}a$ where $x_1x_{i_1} \cdots x_{i_t}a$ is a prefix of $y_1y_{i_1} \cdots y_{i_t}$ for some $t \geqslant 1$ and $2 \leqslant i_1, \ldots, i_t \leqslant n$. Thus, $T(M_1(X)) \subseteq T(M_2(Y))$ if and only if there is no a-marked solution for *VCP* pair (X, Y), the undecidability of which follows directly from Lemma 3.1. ▌

A *dpda* $M = (K, \Sigma, \Gamma, H, q_0, Z_0, F)$ is *stack uniform* in the sense of Linna [14] if it is realtime and whenever H contains transitions (q, A, a, q', u) and (p, A', a, p', u'), then $|u| = |u'|$. If M is stack uniform, then $L(M)$ is ℓ-stack uniform and $T(M)$ is t-stack uniform.

Thus, in the stack uniform condition, changes in stack height depend only on the input and not on the state or stack contents. It is not difficult to see that the class of ℓ-stack uniform languages (t-stack uniform languages) is properly contained in the class of ℓ-superdeterministic (t-superdeterminitic) languages and the transformations from stack uniform to superdeterministic *dpdas* are effective [6]. The classes of t-stack uniform and of ℓ-superdeterministic languages are incomparable [6].

Linna [14] showed that inclusion is decidable for ℓ-stack uniform languages and equi-

valence is decidable for t-stack uniform languages. We now observe that inclusion is undecidable for t-stack uniform languages. The machine $M_1(X)$ is obviously stack uniform. The machine $M_2(Y)$ is stack uniform except for the e-rules. Create $M_2'(Y)$ by substituting for the rule set

$$\{(q_2, \sigma, e, q_1, C), (q_1, C, \sigma, q_2, e) \mid \sigma \text{ in } \Sigma\}$$

the rule set

$$\{(q_2, \sigma, \gamma, q_2, e) \mid \sigma, \gamma \text{ in } \Sigma\}$$

and let the set of accepting modes be $\{(q_2, \sigma) \mid \sigma \text{ in } \Sigma\}$; eliminate the new symbol C. Then $M_2'(Y)$ is obviously stack uniform and $T(M_2(Y)) = T(M_2'(Y))$. Hence, the arguments of this section apply to stack uniform languages.

COROLLARY. *It is undecidable whether* "$T(M_1) \subseteq T(M)$" *for stack uniform dpdas* M_1 *and* M_2.

4. THE EQUIVALENCE PROBLEM

In this section, we show that equivalence is decidable for superdeterministic *dpdas* accepting by accept mode. We use variants of Valiant's "alternate stacking" construction combined with the notion of "placing a barrier" in the stack under certain circumstances when popping below the barrier cannot result in a live configuration.

"Alternate stacking" means simulating two machines M_1 and M_2 with one machine M whose stack contents $u_1v_1 \cdots u_nv_n$ encode the stacks $u_1 \cdots u_n$ of M_1 and $v_1 \cdots v_n$ of M_2; machine M uses u_n to simulate one step of M_1 and v_n for one step of M_2. In the general case the simulating machine M may not be implementable by a *pda*. Alternate stacking "succeeds" if the stacks can be interwoven in such a way that M can be constructed as a *pda*; this happens if, for example, the segments (u_i and v_i) are always nonempty and their size is uniformly bounded. Valiant [20] showed that if M_1 and M_2 are nonsingular (M is nonsingular if there is an integer m such that $(s, y'y) \overset{\ell}{=} (s', y')$ implies $|y| \leqslant m$ or $L((s, y'y)) = \varnothing$) *and* $L(M_1) = L(M_2)$, then the interweaving can indeed be done so that a uniform bound can be placed on the length of all segments as long as the configurations of M_1 and M_2 are live. For each integer r representing a "guess" as to the uniform bound, one can build a *pda* $M(M_1, M_2, r)$ so that, if any stack segment exceeds the bound, $M(M_1, M_2, r)$ accepts and otherwise $M(M_1, M_2, r)$ accepts only when one machine accepts and the other does not. Thus, if $L(M_1) \neq L(M_2)$, each $M(M_1, M_2, r)$ must accept some input, either because some segment exceeds r or because a word is found which is in one language but not in the other. If $L(M_1) = L(M_2)$, then for the "correct" bound r, alternate stacking succeeds and $L(M(M_1, M_2, r)) = \varnothing$. Since emptiness is decidable for *pdas*, equivalence is partially decidable for nonsingular *dpdas*; since inequivalence is obviously partially decidable, equivalence is in fact decidable for *dpdas* known to be nonsingular.

There are many ways to build an "alternate stacking" machine M to simulate two machines M_1 and M_2. The constructions in Valiant [20], in Tanaguchi and Kasami [19], in Linna [14], in Greibach and Friedman [10] and in the present paper all differ in the circumstances under which new levels are created and those under which alternate stacking is said to "fail".

Call configurations c_1 and c_1' in M_1, and c_2 and c_2' in M_2, *paired* if we have

$$c_0 \xrightarrow{u} c_1 \xrightarrow{v} c_1'$$

$$\bar{c}_0 \xrightarrow{u} c_2 \xrightarrow{v} c_2'$$

for c_0 initial in M_1, $\bar{c}_0$ initial in M_2 and c_1' and c_2' live. To demonstrate that a particular alternate stacking construction "works", one finds a relationship $Q(c_1, c_1', c_2, c_2', k)$ for paired configurations and a positive integer k and defines another integer $r(k)$, the segment bound, and a simulating machine $M(M_1, M_2, r(k))$ with the following properties.

(*1) If M_1 is equivalent to M_2, then there is a k such that $Q(c_1, c_1', c_2, c_2', k)$ holds for all paired configurations.

(*2) If alternate stacking fails for some input to $M(M_1, M_2, r(k))$, then $Q(c_1, c_1', c_2, c_2', k)$ fails for some paired configurations.

These two properties show that, if M_1 is equivalent to M_2 then, for some k, alternate stacking succeeds in $M(M_1, M_2, r(k))$ as long as the configurations simulated are live. Characteristically, (*2) is a property of the construction of $M(M_1, M_2, r(k))$ and may not depend on special properties of M_1 and M_2 (e.g., nonsingular or superdeterministic), while (*1) is proven from the special properties of M_1 and M_2. This is only a rough outline; each construction has some special features not easily described in this fashion.

In Valiant's construction [20], before a simulation step, the stack is either

$$u_1v_1 \cdots u_nv_n$$

or

$$u_1v_1 \cdots v_{n-1}u_n,$$

all segments (u_i and v_i) nonempty. Then a simulation step changes the top two segments. In the first case, say, a new segment is created if $|\, u_n \,|$ becomes greater than 1 and the top symbol of u_n is brought up to the new level (thus, u_nv_n becomes $u_n'v_nu_{n+1}$); if $u_n = e$, the top level is closed and v_{n-1} and v_n are merged while, if $v_n = e$, u_n becomes the top segment. Alternate stacking fails if the top segment exceeds r in length. Then, one takes $r = k$ and the relationship Q becomes the following condition, Q_1.

Q_1: if $c_1 \downarrow (w)\, c_1'$, $c_2 \uparrow (w)\, c_2'$, then $|\, c_2' \,| - |\, c_2 \,| \leqslant k$

(and symmetrically, exchanging the roles of M_1 and M_2).

For superdeterministic machines and $\mathscr{E}$-acceptance, one could modify this procedure to handle the delay d of M_1 and M_2; the actual construction in [10] was a quite different

and asymmetric one, to establish the stronger result that "$L(M_1) \subseteq L(M_2)$" is decidable for M_1 an arbitrary *pda* (i.e., $L(M_1)$ an arbitrary context-free language) and M_2 superdeterministic.

For t-acceptance by superdeterministic machines, property (*1) for this Q (or an obvious modification to handle the delay d) does not always hold. For example, M_1 could accept a^+ by keeping the stack height at 1 while M_2 steadily increases the stack using *pds* symbols which keep it in accept mode.

We can describe a version of alternate stacking for 1-increasing machines which keeps intermediate segments (u_i, v_i, $i \neq 1$) bounded by k. A new level is created when u_n or v_n has length $2k$. Then the top k symbols of the "large" segment are taken into new level while, if either $| u_n |$ or $| v_n |$ is less than $2k$, it is brought up entirely and its place in level n becomes empty. If u_n becomes empty but the whole stack of M_1 is nonempty, the levels are coalesced down to and including the first level i with u_i nonempty, and similarly for v_n. If the top segment ever exceeds $2k$, alternate stacking fails. Now the relationship Q can be taken as the following condition Q_2.

$$Q_2\text{: if } c_1 \xrightarrow{w} c_1',\ c_2 \downarrow (w)\ c_2',\ \text{and } |c_1'| \geqslant |c_1|, \text{ then } |c_2| - |c_2'| \leqslant k$$

(and the symmetric relationship exchanging the roles of M_1 and M_2).

Here too, (*1) holds for ℓ-acceptance by superdeterministic machines, but not for t-acceptance. For example, M_1 could accept $\{a^n b^n c^m \mid n, m \geqslant 1\}$ by first inserting an endmarker \$ on the stack then adding to the stack one A for each input a, then popping one A for each input b and finally, when \$ is reached, holding the stack constant at \$ while reading c's in accepting mode. Machine M_2 could do the same thing except that, while reading c's, it pushes C's, always in accepting mode. So for input $w = b^{k+1}c^{k+2}$, after input a^{k+1}, M_1 has a net stack drop of $k+1$ while M_2 has a net stack increase of 1.

Our solution is to put down a "barrier" in u_n or v_n in some (but not necessarily all) circumstances in which M_1 or M_2 cannot accept if it reads further down the stack. Then we add to Q_1 or to Q_2 a condition on LIVE(c_1') or LIVE(c_2').

The first (Valiant) construction can be modified so that alternate stacking fails when, say, $| v_n |$ exceeds the segment bound r AND, for the corresponding configuration c_2' in M_2, LIVE(c_2') $\geqslant r$. If LIVE(c_2') $< r$, the simulating *pda* M places a barrier E below the rth symbol in v_n; symbols below E can be removed when necessary to maintain segment bounds. The condition Q_1 becomes the "matched pushing property" defined below. The second construction sketched above can be modified in a similar fashion an the condition Q_2 becomes the "matched popping property" defined below.

In the Appendix, we prove that the class of superdeterministic *dpdas* has both the matched pushing and the matched popping property. We sketch the two corresponding constructions in this section. We give both approaches to illustrate the variety of alternate stacking techniques available. The segment bound we obtain in the matched pushing case is exponentially better than the bound in the matched popping case and so the corresponding algorithm is exponentially faster.

We give the necessary definitions.

DEFINITION. Let M_1 and M_2 be *dpdas*. An integer $r \geqslant 3$ is a *matched pushing number* for (M_1, M_2) if there are no accessible configurations c_1 in M_1, c_2 in M_2 and input w such that

$$c_1 \equiv c_2, \qquad c_1 \xrightarrow{w} c_1', \qquad c_2 \uparrow (w)\, c_2',$$

$$c_1' \text{ and } c_2' \text{ are live,}$$

$$|c_2'| \geqslant |c_2| + r, \qquad \mathrm{LIVE}(c_2') \geqslant r \qquad \text{and} \qquad |c_1| \geqslant |c_1'|.$$

If r is a matched pushing number for (M_1, M_2), then stack increases (pushes) in M_1 cannot lag too far behind those in M_2. There cannot be an input w taking equivalent accessible configurations c_1 in M_1 and c_2 in M_2 to live configurations such that the computation from c_1 has no net stack increase while the computation in M_2 is a stacking computation increasing the stack by at least r "useful" symbols, i.e., some accepting computation will later pop these r new symbols.

DEFINITION. Let M_1 and M_2 be *dpdas*. An integer $r \geqslant 3$ is a *matched popping number* for (M_1, M_2) if there are no accessible configurations c_1 in M_1, c_2 in M_2 and input w such that

$$c_1 \equiv c_2, \qquad c_1 \xrightarrow{w} c_1', \qquad c_2 \downarrow (w)\, c_2',$$

$$c_1' \text{ and } c_2' \text{ are live,}$$

$$|c_1'| \geqslant |c_1|, \qquad \mathrm{LIVE}(c_1') \geqslant r, \qquad |c_2| \geqslant |c_2'| + r.$$

If r is a matched popping number for (M_1, M_2) then stack decreases (pops) in M_1 cannot lag too far behind those in M_2. There cannot be an input w taking equivalent accessible configurations c_1 in M_1 and c_2 in M_2 to live configurations such that the computation from c_1 has no net stack decrease and ends in a configuration with at least r "live" pushdown store symbols (some accepting computation will pop them), while the computation from c_2 is a popping computation popping at least r symbols.

DEFINITION. A class $\mathscr{C}$ of *dpdas* has the *matched pushing property* (the *matched popping property*) if, for each pair (M_1, M) of *dpdas* in $\mathscr{C}$, there is a matched pushing number (a matched popping number) for (M_1, M_2).

Notice that, if r is a matched pushing number (a matched popping number) for (M_1, M_2), so is any $r' > r$. Hence, if we are dealing with a class of *dpdas* with the matched pushing property (the matched popping property), we can assume that a matched pushing number (a matched popping number) for (M_1, M_2) is also a matched pushing number (a matched popping number) for the other three possible pairs: (M_2, M_1), (M_1, M_1) and (M_2, M_2); in this case, we call it a "matched pushing number for M_1 and M_2" (a "matched popping number for M_1 and M_2").

Lemma A.2 of the Appendix says that the class of superdeterministic *dpdas* has the matched pushing property, while Lemma A.3 yields the matched popping property.

In both cases, we give an explicit expression for the corresponding number in terms of the size of the state set, the pushdown store set and the delay. Lemma 8 of [14] essentially shows that the class of stack uniform *dpdas* has the matched pushing property. The proof for the larger class of superdeterministic *dpdas* is similar, with a few extra complications caused by the need to focus on configurations in reading mode. We give it in full only to get the explicit bound, which is polynomial in the size of the machines, while the bound in [14] appears to grow as $2^{p(n)}$.

Before discussing the equivalence problem further, we place an additional condition on our *dpdas* which allows the simulating machine to decide "easily" whether a configuration is live or a-live, and also avoids certain pathologies due to e-rules. We call *dpdas* with this property "normalized".

Let $M = (K, \Sigma, \Gamma, H, q_0, Z_0, F)$ be a *dpda*. We call M *seminormalized* if

(1) M is 1-increasing, and

(2) all accepting modes are reading modes.

We call M *normalized* if it is seminormalized and

(3) there exists a function μ on Γ such that, whenever yA is a stack of an accessible configuration with A in Γ, then $\mu(A)$ lists all pairs (p, b), $p \in K$, $b \in \Sigma \cup \{e\}$ such that (p, yA) is b-live and all states p such that e is in $T((p, yA))$.

Condition (3) says that the stack symbols encode information about the possible future behavior of the machine. In particular, if (q, yA) is accessible, then A encodes information on whether (p, yA) is b-live and on whether e is in (p, yA); the function μ is the "decoder".

Standard arguments show that an equivalent normalized *dpda* can be effectively constructed from an arbitrary *dpda* (cf. [7] for *dpdas* and [8] for similar constructions for stack machines). Further, this construction can be made to preserve superdeterminism and to preserve the matched pushing (matched popping) property. The latter means that, if $\mathscr{C}$ is a class of *dpdas* with the matched pushing (popping) property, then there is an algorithm t to construct for each M in $\mathscr{C}$ an equivalent normalized *dpda* $t(M)$ such that $\mathscr{C}' = \{t(M) \mid M \text{ in } \mathscr{C}\}$ has the property.

The conversion to a seminormalized *dpda* increases machine size at most polynomially. The construction needed to make a *dpda* 1-increasing is obvious and clearly preserves superdeterminism and the matched pushing (popping) property. A little care must be exercised in enforcing (2): if we wish to preserve superdeterminism, the new machine must "hold off" entering an accepting mode until all e-rules are completed. On the other hand, the matched pushing (poping) property talks about live configurations so, if we wish to preserve just that property, the new machine should enter reading mode "prematurely" and then catch up on its e-moves.

The construction to enforce (3) does not affect stack height and so does not affect superdeterminism or the matched pushing (popping) property. Unfortunately, it does increase the machine size exponentially. Let us discuss this further, since we later try to minimize the effect of this exponenential increase.

The idea is to associate to a stack y a function TABLE_y from $K \times K \times (\Sigma \cup \{e\})$

into quadruples of 0's and 1's. We have $\text{TABLE}_y(p, q, a) = (i, j, k, t)$ where $i = 1$ if and only if there is a w such that $(p, y) \xrightarrow{aw} (q, y)$, $j = 1$ if and only if there exists an input w and accepting configuration c' such that $|c'| \neq 0$ and $(p, y) \xrightarrow{aw} c'$, $k = 1$ if and only if $(p, y) \xrightarrow{e} (q, e)$, and $t = 1$ if and only if there exists an accepting configuration c' such that $|c'| \neq 0$ and $(p, y) \xrightarrow{e} c'$.

The following facts are standard, for y in Γ^* and A in Γ. (1) TABLE_A can be computed from M in polynomial time. (2) If $\text{TABLE}_y = \text{TABLE}_{y'}$ then, for all x, $\text{TABLE}_{yx} = \text{TABLE}_{y'x}$. (3) TABLE_{yA} can be constructed from TABLE_y and TABLE_A alone. (4) If yA is the stack of an accessible configuration, then $\mu(A)$ can be computed from TABLE_{yA} and thus from TABLE_y and A.

The normalized *dpda* M' has as stack symbols pairs (A, T) where A is a stack symbol of M and T encodes a table. A stack $A_1 \cdots A_n$ of M, each A_i a *pds* symbol, becomes in M' a stack $(A_1, T_0)\ (A_2, T_1) \cdots (A_n, T_{n-1})$ where, for $1 \leqslant i \leqslant n - 1$, T_i encodes $\text{TABLE}_{A_1 \cdots A_i}$ and T_0 encodes T_e (actually the bottom table is irrelevant, since we do not allow moves on empty store).

So we can assume that our *dpdas* are normalized. This will allow the simulating *dpda* to decide whether a simulated configuration is live or a-live by examining only the top *pds* symbol. For fixed r, the simulating *pda* will be able to determine for a configuration c whether $\text{LIVE}(c) \geqslant r$ by examining only the top r symbols of c and state(c). This is critical to the construction since the simulating machine must either accept if $\text{LIVE}(c) \geqslant r$ or else place a barrier r symbols down.

We next describe the construction from normalized *dpdas* M_1 and M_2 and an integer r (representing a "guess" at the matched pushing number) of a simulating *pda* $M = M(M_1, M_2, r)$ which keeps its segments bounded by r and has two accept modes ACCEPT_1 and ACCEPT_2 with the following properties. If $T(M) = \varnothing$, then $T(M_1) = T(M_2)$. If M ever reaches ACCEPT_1, then $T(M_1) \neq T(M_2)$. If M ever reaches ACCEPT_2, either $T(M_1) \neq T(M_2)$ or else r is not a matched pushing number for one of (M_1, M_2) and (M_2, M_1). This construction works for any pair of normalized *dpdas* M_1 and M_2. However, it is only when M_1 and M_2 belong to a class with the matched pushing property that the construction is useful. For if r is a matched pushing number for (M_1, M_2) and (M_2, M_1), then $T(M(M_2, M_2, r)) = \varnothing$ if and only if $T(M_1) = T(M_2)$. Thus, following Valiant [20], we can construct a countable series of machines $M(M_1, M_2, r)$ such that $T(M_1) = T(M_2)$ if and only if there is an r such that $T(M(M_1, M_2, r)) = \varnothing$, so equivalence is decidable. So equivalence is decidable for any class $\mathscr{C}$ of normalized *dpdas* with the matched pushing property.

To avoid certain complications, we assume that there are no blocking modes in M_1 and M_2. We do so by adding a new state DEAD and, for each input b and blocking mode (q, A) (including (q, e)), a new transition $(q, A, b, \text{DEAD}, e)$ and, for all A in $\Gamma \cup \{e\}$ and input symbols b, new transitions $(\text{DEAD}, A, b, \text{DEAD}, e)$. Since configurations with state DEAD are of course dead, this does not affect the matched pushing property. It does mean adding previously forbidden transitions with empty store, which are assumed to have the effect of leaving the store empty; again, since applications of these transitions lead only to dead configurations, this causes no problem.

The construction of a simulating *pda* is obtained by an obvious modification of Valiant's

construction and hence naturally resembles the construction in Section 5 of [14]. So we give only a sketch and do not prove the supporting lemma (Lemma 4.1).

Let $M_i = (K_i, \Sigma, \Gamma_i, H_i, q_i, Z_i, F_i)$ be normalized *dpdas*, $i = 1, 2$. Let $r \geqslant 3$ be an integer. We construct $M = M(M_1, M_2, r)$ as follows.

We can assume that $\Gamma_1 \cap \Gamma_2 = \varnothing$. A pair (p_1, p_2) on the top of the stack of M, p_2 in K_1 and p_2 in K_2, will indicate the states of M_1 and M_2. Let \$ be a new symbol to indicate the bottom of the stack.

A new symbol E will be used to create a barrier whenever M determines that M_1 or M_2 cannot read any further down the store and still accept. Strictly speaking, M has *two* barrier symbols E_1 and E_2, one for use with segments of the stack of M_1 and one for M_2, since the segments must be kept separate. However, to avoid further notational unpleasantness, we will speak only of "a barrier symbol E". Any mode (q, E) results in transfer to state DEAD as discussed above. We define auxiliary functions f_E and g_E by

$$f_E(u) = g_E(u) = u = f_E(vEu) \qquad \text{and} \qquad g_E(vEu) = Eu,$$

for u in $(\Gamma_1 \cup \Gamma_2)^*$ and v in $(\Gamma_1 \cup \Gamma_2 \cup \{E\})^*$. Thus, f_E takes the largest possible suffix without E and g_E also takes the rightmost E, if any.

Machine M has three special modes which are all blocking modes: REJECT, which blocks without accepting, and ACCEPT_1 and ACCEPT_2, which accept and block. Machine M enters ACCEPT_1 when it knows that $T(M_1) \neq T(M_2)$; it enters ACCEPT_2 when alternate stacking fails. It enters REJECT when it knows that no further input can be accepted from either configuration and so the current computation is irrelevant to the equivalence problem. The machine has two special states: MAIN · SUBROUTINE, which starts the main simulation subroutine, and CLEANUP, which either leads to ACCEPT_1, ACCEPT_2 or REJECT, or else manipulates the store into proper shape for the return to MAIN · SUBROUTINE.

Initially, M places on its *pds*

$$\$Z_1Z_2(q_1, q_2)$$

and goes to state MAIN · SUBROUTINE.

In state MAIN · SUBROUTINE, the pushdown store of M is either

(i) $\$u_1v_1 \cdots u_nv_n(p_1, p_2), \quad n \geqslant 1$

or

(ii) $\$u_1v_1 \cdots u_n(p_1, p_2), \quad n \geqslant 2,$

where each segment u_i (v_i) contains from 1 to r symbols from Γ_1 (Γ_2), possibly preceded by the barrier E and, in Case (i), $|f_E(u_n)| \leqslant 1$ and, in Case (ii), $|f_E(v_{n-1})| \leqslant 1$; p_i is a state of M_i.

This store corresponds to some configurations c_1 in M_1 and c_2 in M_2. If the barrier does not occur, $c_1 = (p_1, u_1 \cdots u_n)$ and similarly for c_2. Let us discuss Case (i); the transitions for Case (ii) are similar.

If (p_1, u_n) is in accepting mode but e is not in $T(c_2)$, or if (p_2, v_n) is in accepting mode but e is not in $T(c_1)$, M goes to ACCEPT$_1$ since the input to date is in exactly one of $T(M_1)$, $T(M_2)$. If $v_n = E$, then by the construction c_2 is not live, so M goes to ACCEPT$_1$ if c_1 is live and otherwise to REJECT; similarly if $u_n = E$.

If none of the above occur, then M simulates a step of M_1 and M_2. Let a be in Σ if (p_1, u_n) and (p_2, v_n) are in reading mode and otherwise let $a = e$. Write

$$(p_1, u_n) \xrightarrow{a} (p_1', u_n')$$

$$(p_2, v_n) \xrightarrow{a} (p_2', v_n'),$$

where both computations are 1-step unless $a = e$ and one configuration is in reading mode, in which case one computation is 1-step and the other, 0-step.

Then M changes its *pds* to

$$\$u_1v_1 \cdots u_n'v_n'(p_1', p_2')$$

and goes to state CLEANUP. Notice that if $u_n' = e$ the segments $v_{n-1}v_n'$ are coalesced.

To avoid compounding primes, suppose M enters state CLEANUP with one of the following stacks:

$$\$(p_1, p_2)$$

$$\$u_1(p_1, p_2)$$

$$\$v_1(p_1, p_2)$$

$$\$u_1v_1 \cdots u_nv_n(p_1, p_2), \qquad n \geqslant 1$$

$$\$u_1v_1 \cdots u_n(p_1, p_2), \qquad n \geqslant 2.$$

where the same conditions hold as in MAIN · SUBROUTINE except for the top two segments. The top segment may contain up to $2r + 1$ *pds* symbols and up to two occurrences of E, and the second segment can contain up to r *pds* symbols with possibly an occurrence of E leftmost. Let c_1 and c_2 be the corresponding configurations of M_1 and M_2.

The first three cases mean that one or both of M_1 and M_2 have emptied the store. In the first case, M goes to ACCEPT$_1$ if exactly one of (p_1, e), (p_2, e) is accepting and otherwise to REJECT. In the second case, M goes to ACCEPT$_1$ if either e is in exactly one of $T((p_2, e))$ and $T(c_1)$ or c_1 is a-live for any a in Σ, and otherwise to REJECT. The third case is similar.

The fourth and fifth cases are similar, so we discuss only the fourth. Machine M returns to MAIN · SUBROUTINE if its *pds* is in the proper format. Otherwise, one of the following situations occurs. Machine M handles the cases in the order given. After each action, M returns to MAIN · SUBROUTINE if possible and otherwise to CLEANUP (except for 4(a), when it goes to ACCEPT$_2$).

(1) If a barrier E occurs "in the middle" (i.e., not leftmost) of v_n, then M removes all *pds* symbols of v_n below the topmost barrier; similarly if E occurs in the middle of u_n.

Thus, if $v_n = vEv'$, $v \neq e$, then M replaces v_n with $g_E(v_n)$ and similarly u_n is replaced by $g_E(u_n)$.

Thus, in the other three situations we can assume that any barrier E appearing in u_n or v_n does so only as the leftmost symbol in the segment.

(2) If both u_n and v_n contain at least 2 *pds* symbols other than a barrier but v_n has at most $r + 1$ such symbols (otherwise, see (4) below), then M creates two new top segments by pulling up the top symbol from each segment. Thus, if $u_n = uA$ and $v_n = vB$ with $f_E(u) \neq e \neq f_E(v)$, A, B *pds* symbols and $|f_E(v)| \leqslant r$, then M replaces $u_n v_n$ with $uvAB$, thus creating two new top segments.

(3) If u_n has at least two *pds* symbols other than a barrier and v_n has at most one *pds* symbol other than a barrier, then M creates one new top segment by pulling up the topmost symbol of u_n. That is, if $u_n = uA$ with $|f_E(u)| \geqslant 1$, $|f_E(v_n)| \leqslant 1$ and A a *pds* symbol, M replaces $u_n v_n$ with uv_nA, thus creating a new top segment.

(4) If either it is the case that v_n contains at least $r + 2$ *pds* symbols above any barrier or it is the case that v_n contains $r + 1$ such symbols *and* u_n contains at most one, then one of two things happens.

(a) If $\text{LIVE}(c_2) \geqslant r$, then M goes to ACCEPT_2 because either c_1 and c_2 are not equivalent or r is not a matched pushing number for M_1 and M_2.

(b) If $\text{LIVE}(c_2) < r$, then M removes all but the topmost r *pds* symbols from v_n and then places a barrier E as the leftmost symbol of this newly revised segment since, if M ever reads down to the barrier, the corresponding M_2 configuration must be dead.

That is, if $f_E(v_n) = v'v$ with $v' \neq e$ and $|v| = r$ and either $|f_E(u_n)| \leqslant 1$ or $|v'| \geqslant 2$, then either (a) M goes to ACCEPT_2 if $\text{LIVE}(c_2) \geqslant r$ or (b) M replaces v_n with Ev if $\text{LIVE}(c_2) < r$.

Observe that M always leaves CLEANUP either to go to ACCEPT_2 or to return to MAIN · SUBROUTINE. Each of cases (1)–(4) can occur only once. Cases (2) and (3) cause an immediate return to MAIN · SUBROUTINE, and Case (4a) leads to ACCEPT_2. Case (4b) could be followed by (2) (if $|f_E(u_n)|$, $|f_E(v')| \geqslant 2$) and (1) could be followed by (2), (3) or (4). So M can be in CLEANUP at most three times in a row. Hence, M will be able to simulate computations of M_1 and M_2 on all words w in $T(M_1) \cup T(M_2)$, unless an initial subword of w leads to ACCEPT_1 or ACCEPT_2.

Standard arguments (cf. Lemma 9 of [14]) yield the following result, which we state without proof.

LEMMA 4.1. *Let M_1 and M_2 be normalized dpdas, let $r \geqslant 3$ be an integer and let $M = M(M_1, M_2, r)$. Then M has the following properties.*

(1) *If $T(M_1) \neq T(M_2)$, then $T(M) \neq \varnothing$.*

(2) *If M ever reaches* ACCEPT_1, *then $T(M_1) \neq T(M_2)$.*

(3) *If M ever reaches* ACCEPT_2, *then either $T(M_1) \neq T(M_2)$ or r is not a matched pushing number for M_1 and M_2.*

Now one can use Valiant's argument that inequivalence is always partially decidable while, for a class $\mathscr{C}$ of *dpdas* with the matched pushing property, testing the series of *pdas* $M(M_1, M_2, r)$ for emptiness gives a partial decision procedure for equivalence. Hence, equivalence is decidable.

THEOREM 4.2. *If $\mathscr{C}$ is a class of dpdas and $\mathscr{C}$ has the matched pushing property, then equivalence is decidable for $\mathscr{C}$.*

For superdeterministic *dpdas*, Lemma A.2 (in the Appendix) and some further discussion of the normalization process yields something stronger. Given superdeterministic *dpdas* M_1 and M_2, we can calculate a matched pushing number r and hence a specific $M = M(M_1, M_2, r)$ such that $T(M) = \varnothing$ if and only if $T(M_1) = T(M_2)$. Bounding r bounds the size of $M(M_1, M_2, r)$ and "$T(M) = \varnothing$" can be tested in time polynomial in the size of M. Putting all this together gives an estimate for the time complexity of deciding equivalence for superdeterministic *dpdas*.

THEOREM 4.3. *Equivalence can be decided for superdeterministic dpdas in time proportional to*

$$2^{p(n)}$$

where n bounds the size of the machines and $p(n)$ is a polynomial.

Proof. Suppose the machines M_1 and M_2 are of delay d and have the following bounds on the set sizes: states, s; *pds* symbols, g; input symbols, I; $h = \text{Max}\{|\, u \,| \mid \text{some rule } (p, A, a, q, u) \text{ is in } M_1 \text{ or } M_2\}$. We take the size of the machines as $n = s + g + I + h + d$.

The *dpdas* can be converted to seminormalized superdeterministic *dpdas* M_1' and M_2' at the cost of an increase in the number of states to $s' = 2sgIh$ and the delay to $d + h$ and with no increase in the number of pushdown store symbols. From the seminormalized *dpdas*, we can calculate, using Lemma A.2, a matched pushing number r with

$$r = 2(d + h) + (d + h + 1)^2 ((s')^4 g^2 + 1) ((s')^4 g^2 + 2).$$

For an appropriate constant k_1, replacing s, d, h, g and I by n yields

$$r \leqslant k_1 n^{38}.$$

The straightforward approach is to convert M_1' and M_2' to normalized *dpdas* $\overline{M}_1$ and $\overline{M}_2$. As mentioned before, this can be done without changing the matched pushing number r or the number of states. The conversion does involve replacing a pushdown store symbol A with a pair (A, TABLE_y) and there are $k_2^{s^2(I+1)}$ possible tables for an appropriate constant k_2. So the number of *pds* symbols increases to $\bar{g} = gk_2^{s^2(I+1)}$.

The final decision procedure runs in time polynomial in the size of the simulating machine $M = (\overline{M}_1, \overline{M}_2, r)$ [20]. The finite state control of M must be able to store two states and the top two segments, with up to $3r + 4$ symbols (including E's). Thus, the

size of M is roughly proportional to $\bar{g}(s')^2\,\bar{g}^{3r+4}$. Unfortunately, $\bar{g}$ is exponential in the size of the original two machines.

We can avoid this extra exponential in the following way. First, observe that for superdeterministic machines we know the matched pushing number and construct only one machine $M(M_1, M_2, r)$. Hence, in CLEANUP rule (4), there is no need to test "LIVE$(c_2) \leqslant r$"; action (4b) is always performed. Then if, say, $v_n = E$ when M is in state MAIN · SUBROUTINE, either c_1 and c_2 are not equivalent or c_2 is dead, so M goes to state ACCEPT$_1$ unless c_1 and c_2 are both dead, in which case it goes to REJECT. However, M still has to be able to test c_1 and c_2 for "liveness". The solution is to carry out only part of the normalization procedure. Machine M simulates M_1' and M_2'. The various pushdown store segments (u_i and v_i) are stored with only the TABLES for the symbol directly below (e.g., u_i with the table for the symbol directly below u_i in c_i). For the top two segments, these TABLES are stored in the finite state control. Also M stores, for each *pds* symbol A, TABLE$_A$. The top two segments, *without tables*, are stored in the finite state control of M. Now TABLE$_{yA}$ can be computed from TABLE$_y$ and TABLE$_A$. So M has enough data to reconstruct the TABLES for symbols in u_n and v_n when needed, and does so every time the appropriate information is required, without ever storing more than 4 TABLES at a time. Hence, the number of pushdown store symbols of M is proportional to $\bar{g}^4(s')^2\,g^{3r+4}$.

This allows us to approximate the size of M by:

$$(nk_2^{n^3})^4(n^4)^2\,n^{3k_1n^{38}+4}.$$

The term $n^{3k_1n^{38}}$ eventually swamps the other terms. So the time complexity of the decision algorithm is polynomial in that term. Hence, we get a bound of

$$c_1n^{c_2n^{38}}.$$

Since $n^{n^{38}} = 2^{(\log_2 n)n^{38}} < 2^{n^{39}}$, we get an order of magnitude bound of $2^{p(n)}$, p a polynomial, as claimed. ∎

We next give the construction of the simulating *pda* $M = M(M_1, M_2, r)$ in the matched popping case.

Let M_1, M_2, r, E and \$ be given as before, with the same conditions on M_1 and M_2. We extend the definition of f_E and g_E so that $f_E(\$u) = f_E(u)$ and $g_E(\$u) = \$g_E(u)$. In this construction, we place segments of M_1 and M_2 in parallel, rather than alternating them, and allow one (but only one) segment in a level to be empty. (We sometimes call this *parallel stacking*.) We carry up the end-of-stack marker \$ when necessary.

The set of *pds* symbols of M is

$$\Gamma = \{(u, v) \mid u \text{ in } (\Gamma_1 \cup \{E, \$\})^*,\ v \text{ in } (\Gamma_2 \cup \{E, \$\})^*,\ 0 \leqslant |u|, |v| \leqslant 3r + 3\}$$
$$\cup\ \{(p, q) \mid p \text{ in } K_1,\ q \text{ in } K_2\}.$$

MAIN · SUBROUTINE, CLEANUP, ACCEPT$_1$, ACCEPT$_2$ and REJECT play the same roles as before.

I. Initially, M places on the store

$$(\$Z_1 , \$Z_2)(q_1 , q_2)$$

and goes to MAIN · SUBROUTINE.

In the state MAIN · SUBROUTINE, the pushdown store of M is

$$(u_1 , v_1)(u_2 , v_2) \cdots (u_n , v_n)(p_1 , p_2)$$

where p_i is a state in M_i, $u_1u_2 \cdots u_n$ and $v_1v_2 \cdots v_n$ contain \$ leftmost and no other occurrences of \$ and, for $1 \leqslant i \leqslant n - 1$, $u_iv_i \neq e$, u_i is either empty or contains r members of Γ_1 possibly preceded by E, \$ or $\$E$. Similarly, v_i is either empty or contains r members of Γ_2 possibly preceded by E, \$ or $\$E$, u_n contains at most $2r - 1$ members of Γ_1 possibly preceded by E, \$ or $\$E$ and v_n contains at most $2r - 1$ members of Γ_2 possibly preceded by E, \$ or $\$E$, and $u_nv_n \neq e$.

This store corresponds to configurations c_1 with state p_1 in M_1 and c_2 with state p_2 in M_2.

II. First, M determines whether it should go to ACCEPT_1 or REJECT.

A. If (p_1 , u_n) is in accepting mode but e is not in $T(c_2)$ or if (p_2 , v_n) is in accepting mode but e is not in $T(c_1)$, M goes to ACCEPT_1.

B. If $v_n = E$, then M goes to ACCEPT_1 if c_1 is live and otherwise to REJECT: if $u_n = E$, M goes to ACCEPT_1 if c_2 is live and otherwise to REJECT.

If none of the above occur, then M simulates a step of M_1 and M_2. Let a be in Σ if (p_1 , u_n) and (p_2 , v_n) are in reading mode and otherwise let $a = e$. Write

$$(p_1 , u_n) \xrightarrow{a} (p_1' , u_n')$$

$$(p_2 , v_n) \xrightarrow{a} (p_2' , v_n')$$

where both computations are 1-step unless $a = e$ and one configuration is in reading mode, in which case one computation is 1-step and the other 0-step.

Then M replaces (u_n , v_n) with (u_n' , v_n') and (p_1 , p_2) with (p_1' , p_2') and goes to state CLEANUP.

To avoid compounding primes, suppose M is in state CLEANUP with *pds*

$$(u_1 , v_1) \cdots (u_n , v_n)(p_1 , p_2).$$

The conditions on the u_i and v_i are the same as before for $i \neq n$. However, u_n could be empty or could have as many as $3r$ pushdown store symbols plus up to two E's inside, and similarly for v_n.

If the *pds* is suitable for return to MAIN · SUBROUTINE, M does this. Otherwise, it performs one of the five actions below. It searches for the proper action in the order indicated below and afterwards either goes to ACCEPT_1, ACCEPT_2 or REJECT as

indicated below or returns to CLEANUP. Let c_1 and c_2 be the corresponding configurations of M_1 and M_2.

III. One of the machines has emptied its store. This means that $u_n = \$$ or $v_n = \$$.

A. If $u_n = \$ = v_n$, both machines have emptied the store so M goes to ACCEPT$_1$ if exactly one of (p_1, e) and (p_2, e) is in accepting mode and otherwise to REJECT.

B. If $u_n = \$$ and $v_n \neq \$$, M goes to ACCEPT$_1$ if either e is in exactly one of $T((p_1, e))$ and $T(c_2)$ or c_2 is a-live for any a in Σ, and otherwise to REJECT.

C. Same as B exchanging the roles of M_1 and M_2.

IV. Either u_n or v_n has an E "in the middle" (i.e., not leftmost). Then M replaces (u_n, v_n) with $(g_E(u_n), g_E(v_n))$.

V. If $n \geqslant 1$, one segment is empty and the other is not "too large", M closes level n and consolidates levels $n-1$ and n. If $u_n = e$ and $|f_E(v_n)| \leqslant 2r$ or $v_n = e$ and $|f_E(u_n)| \leqslant 2r$, M closes level n by replacing $(u_{n-1}, v_{n-1})(u_n, v_n)$ with $(u_{n-1}u_n, v_{n-1}v_n)$.

VI. One of the top segments is "large" but not "too large" and the other is nonempty and not "too large", so M opens a new level and transfer up the r top symbols of "large" segments and all of "small" segments. If $|f_E(u_n)|, |f_E(v_n)| \leqslant 2r$, $u_n \neq e \neq v_n$ and $|f_E(u_n)| = 2r$ or $|f_E(v_n)| = 2r$, let $u_n = u'_n u_{n+1}$ and $v_n = v'_n v_{n+1}$, where $|u_{n+1}| = r$ if $|f_E(u_n)| = 2r$ and otherwise $u_{n+1} = u_n$ and similarly for v_n. Then M opens level $n+1$ by replacing (u_n, v_n) with $(u'_n, v'_n)(u_{n+1}, v_{n+1})$.

VII. One of the top segments is "too large". This can happen only after levels are consolidated by V and hence the other segment must be "small" (or empty). Then either M goes to ACCEPT$_2$ because parallel stacking fails or it places the barrier E r symbols down.

A. If $|f_E(u_n)| \geqslant 2r+1$, $|f_E(v_n)| \leqslant r$ and LIVE$(c_1) \geqslant r$, then either c_1 and c_2 are inequivalent or r is not a matched popping number, so M goes to ACCEPT$_2$.

B. If $|f_E(u_n)| \geqslant 2r+1$, $|f_E(v_n)| \leqslant r$ and LIVE$(c_1) < r$, then M places a barrier r symbols down by replacing (u_n, v_n) with (Eu, v_n), or $(\$Eu, v_n)$ if u_n contains \$, where $f_E(u_n) = u'u$, $|u| = r$.

C. Same as A, exchanging the roles of u_n and v_n and of c_1 and c_2.

D. Same as B, exchanging the roles of u_n and v_n and of c_1 and c_2.

Again, notice that M must always leave CLEANUP. Cases III and VIIA, C can occur only once. Case VI causes a return to MAIN · SUBROUTINE. Cases VIIB, D cause a return to MAIN · SUBROUTINE unless $v_n = e$ or $u_n = e$, when V occurs. Case IV can recur only if followed by V, while V can occur more than three times only if followed by IV. In the worst case, one could have $2n-2$ CLEANUP steps; e.g., $n-1$ occurrences of IV and V ($u_i = e$, $i \neq 1$, E in v_n) or one occurrence of V, one of VIIB, D, then $n-2$ of IV and V.

The next lemma is analogous to Lemma 4.1.

LEMMA 4.4. *Let M_1 and M_2 be normalized dpdas, let $r \geqslant 3$ be an integer and let $M = M(M_1, M_2, r)$. Then M has the following properties.*

(1) *If $T(M_1) \neq T(M_2)$, then $T(M) \neq \varnothing$.*

(2) *If M ever reaches* ACCEPT_1, *then $T(M_1) \neq T(M_2)$.*

(3) *If M ever reaches* ACCEPT_2, *then either $T(M_1) \neq T(M_2)$ or r is not a matched popping number for M_1 and M_2.*

Proof. The proof of (1) and (2) is standard, so let us consider (3). Suppose $T(M_1) = T(M_2)$ and M reaches ACCEPT_2. This happens only after VIIA or C. The arguments are symmetric so suppose it happens after VIIC.

Situation VIIC occurs only when the top M_2 segment has grown beyond $2r$. After a MAIN · SUBROUTINE simulation step, the top segment can have at most $2r$ *pds* symbols and, when that happens, rule VI causes M to open a new level. Hence, situation VIIC must be preceded by a level consolidation step V. So M goes from a *pds*

$$(u_1, v_1) \cdots (u_n, v_n)(e, v_{n+1})(p_1', p_2')$$

to a *pds*

$$(u_1, v_1) \cdots (u_n, v_n v_{n+1})(p_1', p_2')$$

with $|f_E(v_n v_{n+1})| \geqslant 2r + 1$ and $\text{LIVE}(c_2') \geqslant r$, where c_2' is the corresponding M_2 configuration. Also, $|f_E(v_n)| = r$, v_{n+1} has no E's and $|v_{n+1}| > r$ (otherwise, this would not be the step that causes VIIC to occur). Let c_1' be the corresponding configuration of M_1.

Now $u_n \neq \$$ or else situation III would take precedence. Rule VI ensures that "\$" is carried up until the store grows to $2r$. So there is a j, $1 \leqslant j \leqslant n$, such that $u_j \neq e$ and $u_i = e$ for $i > j$. Consider the last time M opened level $j + 1$ using VI. Since then, level j has not been read and some input w has been read. Thus, M has a *pds*

$$(u_1, v_1) \cdots (u_j, v_j)(u_{j+1}', v_{j+1}')(p_1, p_2)$$

with $|f_E(u_j)| = |u_{j+1}'| = r$ and no E's in u_{j+1}'. Either $v_j = e$ and $|f_E(v_{j+1}')| \leqslant 2r - 1$ or $|f_E(v_j)| = |v_{j+1}'| = r$ and v_{j+1}' contains no E's. So we can write the corresponding accessible configurations c_1 in M_1 and c_2 in M_2 as:

$$c_1 = (p_1, y_1 f_E(u_j)\, u_{j+1}') \qquad \text{and} \qquad c_2 = (p_2, y_2 v_j f_E(v_{j+1}')).$$

Since that point, all M_1 segments have become empty above level j. Hence

$$c_1 \downarrow (w)\, c_1' = (p_1', y_1 f_E(u_j)).$$

and $|c_1| - |c_1'| = |u_{j+1}'| = r$. We also have

$$c_2 \xrightarrow{w} c_2' = (p_2', y_2 v_j y_3 f_E(v_{j+1} \cdots v_{n+1}))$$

for appropriate y_3; the strings y_i represent the fact that applications of IV may cause M to erase some of the symbols of M_1 or M_2 below a barrier. Then $|c_2'| - |c_2| =$

$|f_E(v_{j+1} \cdots v_{n+1})y_3| - |f_E(v'_{j+1})|$. There are two cases. If $j = n$, then $v_j \neq e$, and $|f_E(v_{i+1})| = r$, so

$$|c'_2| - |c_2| \geqslant |f_E(v_{n+1}) - r| \geqslant 1.$$

If $j < n$, then

$$|c'_2| - |c_2| \geqslant |f_E(v_n v_{n+1})| - |f_E(v'_{j+1})| \geqslant (2r + 1) - (2r - 1) \geqslant 2.$$

Hence, either c_1 is not equivalent to c_2 and so $T(M_1) \neq T(M_2)$ or else r is not a matched popping number for M_1 and M_2. ∎

The usual arguments give us the next theorem immediately from Lemma 4.4.

THEOREM 4.5. *Let $\mathscr{C}$ be a class of dpdas and let $\mathscr{C}$ have the matched popping property. Then equivalence is decidable for $\mathscr{C}$.*

Remark. Let $\mathscr{R}$ be the class of realtime *dpdas* accepting by accept mode. It is easy to show that $\mathscr{R}$ has neither the matched pushing nor the matched popping property. We conjecture that there is an algorithm to convert a realtime *dpda* M into an equivalent realtime *dpda* M' such that $\{M'\}$ has the matched pushing property. If this were true, then equivalence would be decidable for $\mathscr{R}$. On the other hand, $\mathscr{R}$ is the largest class to which these ideas can apply in their present form. We can convert an arbitrary *dpda* M into an equivalent *dpda* M' which uses e-rules only to pop the store [7]. If there is a uniform bound on how many symbols can be popped from a live accessible configuration during e-rules, then M' can be converted to an equivalent realtime *dpda* M''. Otherwise, for each r there is an accessible configuration c and a computation c', $c \to^e c'$, $|c| - |c'| \geqslant r$ with c' live and, of course, $c \equiv c'$. So M' does not have the matched popping property and, if the language accepted by M is not realtime, M cannot be converted to any equivalent *dpda* with the matched popping property.

A curious open problem is whether the t-inclusion problem is decidable for realtime (delay 0) superdeterministic *dpdas*; the family of languages accepted by accept mode by such machines is equivalent to the family of languages accepted by accept mode by the "strict restricted" *dpdas* of Igarashi [6, 25].

APPENDIX

In this appendix, we give the proof that the class of superdeterministic pushdown store automata has the matched pushing and the matched popping property. First, we establish a useful property of equivalent configurations in superdeterministic *dpdas*. Lemma A.1 says that, if c_1 and $\bar{c}_1$ are equivalent configurations in superdeterministic *dpdas* and for a particular input string there is a matched push-pop sequence starting from c_1, then there must also be a matched push-pop sequence starting from $\bar{c}_1$. Lemma A.1(1) is similar to Lemma 4′ of [14], while Lemma A.1(2) is stronger.

LEMMA A.1. *Let c_1 and $\bar{c}_1$ be configurations in superdeterministic 1-increasing dpdas M_1 and M_2 such that $c_1 \equiv \bar{c}_1$. Suppose M_1 has a computation*

$$c_1 \uparrow (v)\, c_2 \uparrow (x)\, c_3 \uparrow (y) c_4$$

with

$$v \neq e,\ c_4 \text{ live},\ \text{mode}(c_1) = \text{mode}(c_2),\ \text{mode}(c_3) = \text{mode}(c_4),$$
$$|c_2| = |c_3| \quad \text{and} \quad |c_1| = |c_4|,$$

while in M_2

$$\bar{c}_1 \xrightarrow{v} \bar{c}_2 \xrightarrow{x} \bar{c}_3 \xrightarrow{y} \bar{c}_4,$$
$$\text{state}(\bar{c}_1) = \text{state}(\bar{c}_2),\ \text{state}(\bar{c}_3) = \text{state}(\bar{c}_4),$$
$$\bar{c}_1, \bar{c}_2, \bar{c}_3, \bar{c}_4 \text{ in reading mode.}$$

(1) Then $|\bar{c}_2| \geqslant |\bar{c}_1|$ and $|\bar{c}_2| - |\bar{c}_1| \geqslant |\bar{c}_3| - |\bar{c}_4|$.

(2) Further, if $|c_2| > |c_1|$, then $|\bar{c}_2| > |\bar{c}_1|$, and $|\bar{c}_2| - |\bar{c}_1| = |\bar{c}_3| - |\bar{c}_4|$.

Proof. Clearly, for any $n \geqslant 0$,

$$c_1 \xrightarrow{v^n x y^n} c_4.$$

Hence, c_4, which is live, is equivalent to the configuration that M_2 enters after $v^n x y^n$ starting from $\bar{c}_1$. In particular, M_2 cannot block during $v^n x y^n$. Since $\text{state}(\bar{c}_1) = \text{state}(\bar{c}_2)$ and $\bar{c}_1$ and $\bar{c}_2$ are in reading mode, if $|\bar{c}_1| > |\bar{c}_2|$, each application of v would decrease the store. So, for $n > |\bar{c}_1|$, M_2 would block during v^n. Hence, $|\bar{c}_1| \leqslant |\bar{c}_2|$. Similarly, the net effect of v and y cannot be to decrease the store, so $|\bar{c}_2| - |\bar{c}_1| + |\bar{c}_4| - |\bar{c}_3| \geqslant 0$. This establishes (1).

Now, suppose that $|c_2| > |c_1|$. First, consider the possibility $|\bar{c}_1| = |\bar{c}_2|$. Write:

$$\bar{c}_1 \xrightarrow{v} \bar{c}_{2,1} = \bar{c}_2 \xrightarrow{v} \bar{c}_{2,2} \xrightarrow{v} \cdots \xrightarrow{v} \bar{c}_{2,j} \xrightarrow{v} \cdots$$

where $|\bar{c}_1| = |\bar{c}_{2,j}|$, $\text{state}(\bar{c}_1) = \text{state}(\bar{c}_{2,j})$ and $\bar{c}_{2,j}$ is in reading mode. Let g be the total number of *pds* symbols in M_1 and M_2 and let σ be the total number of states. After $\sigma g^{|\bar{c}_1|} + 1$ repetitions of v, there must be n, $k \geqslant 1$ with $\bar{c}_{2,n} = \bar{c}_{2,n+tk}$ for all $t \geqslant 0$. Since xy^{n+tk} must take $\bar{c}_{2,n+tk}$ to a live configuration, it must also do so for $\bar{c}_{2,n}$.

Since $c_1 \equiv \bar{c}_1$, $v^n x y^{n+tk}$ must take c_1 in M_1 to a live configuration for all $t \geqslant 0$. Since M_1 is finite delay, $y \neq e$ (or else M_1 would erase at least n symbols on e input after $v^n x$). We can write $y = y'b$, for a symbol b, $c_1 = (p, \alpha A)$, $c_2 = (p, \alpha\beta A)$, $c_3 = (q, \alpha\beta B)$, $c_4 = (q, \alpha B)$ for A, B *pds* symbols and $\beta \neq e$. Then we can write:

$$c_1 \xrightarrow{v^n x} (q, \alpha\beta^n B) \xrightarrow{y'} c'_n \xrightarrow{b} (q, \alpha\beta^{n-1}B) \xrightarrow{y'} c'_{n-1} \xrightarrow{b} (q, \alpha\beta^{n-2}B),$$

with c'_n and c'_{n-1} in reading mode. During the two y' computations, M_1 does not read below βB in the store and so these computations are identical. Thus, $\text{mode}(c'_n) = \text{mode}(c'_{n-1})$

and $| \alpha\beta^n B | - | c'_n | = | \alpha\beta^{n-1}B | - | c'_{n-1} |$. Hence, $| c'_{n-1} | < | c'_n |$. By the definition of superdeterminism, future inputs by' continue to erase the stack. If we select t so that $n + tk > | c'_n | + 1$, then input $v^n xy^{n+tk} = v^n xy'(by')^{n+tk-1}b$ will erase the store and cause a block before the last b is read, a contradiction.

Hence, $| \bar{c}_2 | > | \bar{c}_1 |$. A similar argument will show that $| \bar{c}_3 | \neq | \bar{c}_4 |$. For, suppose $| \bar{c}_3 | = | \bar{c}_4 |$. We can write:

$$\bar{c}_1 \xrightarrow{v^n} \bar{c}_{2,n} \xrightarrow{x} \bar{c}_{3,n} \xrightarrow{y^\ell} \bar{c}_{4,n,\ell} \xrightarrow{y^{n-\ell}} \bar{c}_{4,n} ,$$

where $\bar{c}_{3,n}$, $\bar{c}_{4,n,\ell}$ and $\bar{c}_{4,n}$ are in reading mode and $\bar{c}_{4,n}$ is live. For $n > g^{|\bar{c}_3|} + 2$, there must be ℓ, $k \geqslant 1$, $\ell + k < n - 2$ with $\bar{c}_{4,n,\ell} = \bar{c}_{4,n,\ell+k}$. Thus, $v^n xy^{n+tk}$ must lead M_2 (and so M_1) to a live configuration for all $t \geqslant 0$ but, as before, this is impossible for M_1.

Next, we consider the possibility $| \bar{c}_4 | > | \bar{c}_3 |$. Now we must use the "liveness" of c_4 more strongly. Let z be the smallest word such that z takes c_4 to an accepting configuration in M_1. Thus, $v^n xy^n z \in T(c_1) = T(\bar{c}_1)$ for all $n \geqslant 0$. Let M_1 and M_2 be of delay d. Let $s = (d + 1)(| yz | + 2)$. Define $\bar{c}_{2,n}$, $\bar{c}_{4,\ell,n}$ as before.

Now observe that, for $n > | z |$, M_2 does not "touch bottom" during $v^n xy^n z$ since z can erase at most $(| z | + 1)(d + 1)$ symbols and $| \bar{c}_{4,n,\ell} | = | \bar{c}_{3,n} | + \ell(\bar{c}_4 | - | \bar{c}_3 |)$. Also, during y, M_2 certainly never sees more than s *pds* symbols. Hence, if the top s *pds* symbols are the same in $\bar{c}_{4,n,\ell}$ and $\bar{c}_{4,n,\ell+k}$, the configurations are equivalent modulo acceptance of words in y^*z (but not necessarily for other inputs). But, for $n > \sigma g^s + 2$, we must have configurations $\bar{c}_{4,n,\ell}$ and $\bar{c}_{4,n,\ell+k}$, $\ell + k < n - 2$, whose top s symbols are the same. This pattern must then continue, since $| \bar{c}_{4,n,\ell+k} | > | \bar{c}_{4,n,\ell} |$, so $\bar{c}_{4,n,\ell}$ and $\bar{c}_{4,n,\ell+tk}$ have the same top s symbols for all $t \geqslant 0$. Thus, since M_2 reaches an accepting configuration from $\bar{c}_{4,n,\ell}$ with input $y^{n-\ell}z$, it does so after $\bar{c}_{4,n,\ell+tk}$. That is, for all $t \geqslant 0$, $v^n xy^{n+tk}z$ is in $T(c_1) = T(\bar{c}_1)$. Again, this is impossible for t large.

Now, we have $r_1 = | \bar{c}_2 | - | \bar{c}_1 | > 0$ and $r_2 = | \bar{c}_3 | - | \bar{c}_4 | > 0$ and must exclude $r_2 < r_1$. Suppose $r_2 < r_1$. Let z be as before, but let $s = (| xvyz | + 3)(d + 1)$.

We define $\bar{c}_{2,n}$ and $\bar{c}_{4,n}$ as before. Notice that $r_1 < s$, $r_2 < s$. There are n, $k_1 \geqslant 1$ such that the top s symbols of $\bar{c}_{2,n}$ and $\bar{c}_{2,n+k_1}$ are the same. Input v causes fewer than s *pds* symbols to be read and increases the store by r_1. Hence, repetitions of v^{k_1} have the same effect. We can write:

$$\bar{c}_{2,n} = (\bar{p}, \eta\bar{\gamma}) \qquad \text{and} \qquad \bar{c}_{2,n+tk_1} = (\bar{p}, \eta\gamma^t\bar{\gamma})$$

for all $t \geqslant 0$, for $| \gamma | = r_1 k_1$ and $| \bar{\gamma} | = s$. Further, we can assume that $k_1 > 2s$ (or we could take $\bar{k}_1 = (2s + 1)k_1$ instead of k_1).

Let $r_3 = | \bar{c}_3 | - | \bar{c}_2 |$. We have

$$\bar{c}_{3,n+tk_1} = (\bar{q}, \eta\gamma^t\gamma'),$$

where $1 \leqslant | \gamma' | = | \bar{\gamma} | + r_3 \leqslant 2s \leqslant r_1 k_1$. For each ℓ, $1 \leqslant \ell \leqslant n + tk_1$,

$$\bar{c}_{4,n+tk_1,\ell} = (\bar{q}, \eta\gamma^{f(\ell)}\gamma_\ell)$$

for some string γ_ℓ with $1 \leqslant |\gamma_\ell| \leqslant r_1 k_1$. Since input xy^ℓ erases $r_2\ell - r_3 < r_2\ell + s$ *pds* symbols, we have:

$$|\gamma_\ell| + |\gamma| f(\ell) = tk_1r_1 - r_2\ell + r_3 > tk_1r_1 - r_2\ell - s,$$

or

$$|\gamma| f(\ell) > (t-1)k_1r_1 - r_2\ell - s.$$

This relationship will also hold for $\ell > n + tk_1$, as long as $f(\ell) \geqslant 1$.

For $t > (g+1)^{k_1r_1}$, we must have $\gamma_\ell = \gamma_{\ell+k_2}$ with $\ell + k_2 < n + tk_1 - 2$. Hence, repetitions of y^{k_2} have the same effect as long as M_2 does not read η.

Since input $y^{n+tk_1-\ell}z$ leads M_2 to acceptance from $\bar{c}_{4,n+tk_1,\ell}$, so will $y^{n+tk_1-\ell-jk_2}z$, as long as the *pds* of M_2 is at least s symbols above η as it encounters z. Thus, $T(\bar{c}_1)$ will contain $v^{n+tk_1}xy^{n+tk_1+jk_2}z$ and so $T(c_1)$ must do so too. But, if we set $jk_2 > |c_4|$ and $t > \text{Max}(n + jk_2 + 2k_1 + 1, (g+1)^{k_1r_1} + 2)$, we see that M_1 is in configuration c_4 after reading $v^{n+tk_1}xy^{n+tk_1}$ from c_1 and then blocks while y^{jk_2} is read. However, since $r_2 < s < k_1$ and $r_1 < s$:

$$\begin{aligned}
|\gamma| f(n + tk_1 + jk_2) &> (t-1)k_1r_1 - r_2(n + tk_1 + jk_2) - s \\
&= tk_1(r_1 - r_2) - r_2(n + jk_2) - s - k_1r_1 \\
&> tk_1 - r_2(n + jk_2) - s - k_1r_1 \\
&> ts - s(n + jk_2) - s - sk_1 \\
&= s(t - n - jk_2 - 1 - k_1) > sk_1 > r_1k_1 = |\gamma|
\end{aligned}$$

so $f(n + tk_1 + jk_2) \geqslant 1$. So there is at least one γ left on the store when z is read. Hence, $T(\bar{c}_1)$ contains $v^{n+tk_1}xy^{n+tk_1+jk_2}z$, but $T(c_1)$ does not, a contradiction.

Hence,

$$r_1 = |\bar{c}_2| - |\bar{c}_1| = r_2 = |\bar{c}_3| - |\bar{c}_4|. \quad \blacksquare$$

Next, we show that the class of superdeterministic pushdown store automata has the matched pushing property. This strengthens Lemma 8 of [14]. We give the full proof in order to establish a better bound on the matched pushing number (and hence on the time complexity of the equivalence algorithm) and to present ideas re-used in the proof of Lemma A.3 (matched popping).

LEMMA A.2. *Let M_1 and M_2 be seminormalized superdeterministic dpdas of delay d with at most s states and g pds symbols. Let*

$$\begin{aligned}
k &= (s^4g^2 + 1)(d + 1) \\
r &= (k + d + 1)k + 2d.
\end{aligned}$$

Then r is a matched pushing number for M_1 and M_2.

Proof. Consider the following property for a word w and an integer t. $P(w, t)$: There are accessible configurations c_1 in M_1, c_2 in M_2 such that

$$c_1 \equiv c_2, \qquad c_1 \xrightarrow{w} c_1', \qquad c_2 \uparrow (w)\, c_2',$$

$$c_1' \text{ and } c_2' \text{ are live,}$$

$$|\, c_2' \,| \geqslant |\, c_2 \,| + t, \qquad \text{LIVE}(c_2') \geqslant t, \qquad \text{and} \qquad |\, c_1 \,| \geqslant |\, c_1' \,|.$$

Clearly, t is a matched pushing number for (M_1, M_2) if and only if $P(w, t)$ fails for all w. Thus, we must show that $P(w, t)$ fails for all w.

The fact that the configurations need not be in reading mode can cause some difficulties. Hence, we use a more convenient condition $Q(w, t)$. For a live configuration c, let

$$\text{RLIVE}(c) = \text{Max}\{|\, c \,| - |\, c' \,| \mid c' \text{ is accessible from } c \text{ and live and } \textit{in reading mode}\}.$$

If c is dead, let $\text{RLIVE}(c) = -1$. We are measuring how far down the stack M can pop from c and still be in a live configuration in reading mode.

Condition $Q(w, t)$ is defined as follows.

$Q(w, t)$: There are accessible configurations c_1 in M_1 and c_2 in M_2 such that

$$c_1 \equiv c_2, \qquad c_1 \xrightarrow{w} c_1', \qquad c_2 \uparrow (w)\, c_2',$$

$$c_1' \text{ and } c_2' \text{ are live and } \textit{in reading mode,}$$

$$|\, c_2' \,| \geqslant |\, c_2 \,| + t, \qquad \underline{\text{RLIVE}(c_2') \geqslant t} \qquad \text{and} \qquad |\, c_1 \,| \geqslant |\, c_1' \,| - d.$$

Suppose $P(w, t)$ holds for $t > 2d + 1$. Then we have equivalent accessible configurations c_1 and c_2 and computations

$$c_1 \xrightarrow{w} c_1', \qquad c_2 \uparrow (w)\, c_2',$$

with c_1' and c_2' live, $|\, c_2' \,| \geqslant |\, c_2 \,| + t$, $\text{LIVE}(c_2') \geqslant t$ and $|\, c_1 \,| \geqslant |\, c_1' \,|$. Now c_1' and c_2' may not be in reading mode. We can write

$$c_1 \xrightarrow{w} c_1' \xrightarrow{e} c_1'', \qquad c_2 \uparrow (w)\, c_2' \xrightarrow{e} c_2'',$$

with c_1'' and c_2'' in reading mode. Since M_1 and M_2 are of delay d, obviously

$$|\, c_1 \,| \geqslant |\, c_1' \,| \geqslant |\, c_1'' \,| - d, \qquad |\, c_2'' \,| \geqslant |\, c_2' \,| - d \geqslant |\, c_2 \,| + t - d.$$

Since $t > d$, $c_2 \uparrow (w) c_2''$. Now let c_3 be the live configuration of minimal length accessible from c_2'. It is also accessible from c_2''. Clearly, $|\, c_2'' \,| - |\, c_2 \,| \geqslant |\, c_1' \,| - |\, c_2 \,| - d$, so if c_3 is in reading mode, $\text{RLIVE}(c_2'') \geqslant \text{LIVE}(c_2') - d \geqslant t - d$. Suppose c_3 is not in reading

mode. Since $t > 2d + 1$, the input taking c_2' to c_3 can be written as zb, b an individual symbol. Thus, we have

$$c_2' \xrightarrow{e} c_2'' \xrightarrow{z} c_3' \xrightarrow{b} c_3 ,$$

with c_3' live and in reading mode. Since c_3 is not in reading mode, $| c_3' | \leqslant | c_3 | + d$. Also, c_3' is a candidate for the minimal live configuration in reading mode accessibe from c_2''. Hence, $\mathrm{RLIVE}(c_2'') \geqslant \mathrm{LIVE}(c_2') - 2d \geqslant t - 2d$. Comparing with the definition of Q, we see that $Q(w, t - 2d)$ holds.

Thus, it suffices to show that $Q(w, r - 2d)$ fails for all w. Suppose to the contrary that w is a word of minimal length for which $Q(w, r - 2d)$ holds. Then we have accessible configurations c_1 in M_1 and c_2 in M_2 and computations

$$C_1 \colon c_1 \xrightarrow{w} c_1' , \qquad C_2 \colon c_2 \uparrow (w)\, c_2' ,$$

with c_1' and c_2' live and in reading mode, $| c_2' | \geqslant | c_2 | + r - 2d$, $\mathrm{RLIVE}(c_2') \geqslant r - 2d$ and $| c_1 | \geqslant | c_1' | - d$. Since $\mathrm{RLIVE}(c_2') > r - 2d$, there is also a computation

$$C_3 \colon c_2' \downarrow (\bar{w})\, c_3 ,$$

with c_3 live and in reading mode and $| c_2' | \geqslant | c_3 | + r - 2d$.

Our strategy is to show that either c_1 and c_2 are not equivalent or we can find a smaller string w' for which $Q(w', r - 2d)$ holds, thus contradicting the minimality of w. To do so, we establish a series of CLAIMS. CLAIMS 1 and 2 say that C_1 cannot have a subcomputation which increases and then decreases the stack height of M_1 by more than k units. This in turn follows from Lemma A.1, the minimality of w and some elementary combinatorial reasoning. Next CLAIM 3 shows that, because the stack height of M_2 increases by more than $r - 2d$ during C_2 and decreases by that amount during C_3, some subcomputation of C_1 must increase stack height by more than $k + d$. But, since $| c_1 | \geqslant | c_1' | - d$, the stack height must also decrease by k, thus contradicting CLAIM 1. This establishes the lemma.

First we establish CLAIM 1.

CLAIM 1. *Computations C_1 and C_2 cannot have a matched push-pop sequence. That is, C_1 and C_2 cannot be divided*:

$$C_1 \colon c_1 \xrightarrow{u} c_u \uparrow (v)\, c_v \uparrow (x)\, c_x \downarrow (y)\, c_y \xrightarrow{z} c_1' ,$$
$$C_2 \colon c_2 \xrightarrow{u} \bar{c}_u \xrightarrow{v} \bar{c}_v \xrightarrow{x} c_x \xrightarrow{y} \bar{c}_y \xrightarrow{z} c_2' ,$$

with

$$\mathrm{mode}(c_u) = \mathrm{mode}(c_v),\ \mathrm{mode}(c_x) = \mathrm{mode}(c_y),$$
$$| c_v | = | c_x |,\ | c_u | = | c_y |,\ | c_v | > | c_u |,$$
$$\mathrm{state}(\bar{c}_u) = \mathrm{state}(\bar{c}_v),\ \mathrm{state}(\bar{c}_x) = \mathrm{state}(\bar{c}_v)$$
and $\bar{c}_u$, $\bar{c}_v$, $\bar{c}_x$ and $\bar{c}_y$ in reading mode.

Proof. Suppose that such computations exist. We want to obtain a contradiction by showing that $Q(uxz, r - 2d)$ holds. Since M_1 is finite delay, $v \neq e$. By the definitions of "$\uparrow$" and "$\downarrow$", we have

$$c_1 \xrightarrow{u} c_u \uparrow (x)\, c_y \xrightarrow{z} c_1' .$$

By Lemma A.1(2), $| \bar{c}_v | - | \bar{c}_u | = | \bar{c}_x | - | \bar{c}_y | > 0$.

Since M_2 is superdeterministic,

$$c_2 \xrightarrow{u} \bar{c}_u \xrightarrow{x} \bar{c}_x' \xrightarrow{z} c_2'' ,$$

with

$$\bar{c}_x' \text{ and } c_2'' \text{ in reading mode,}$$

$$| \bar{c}_x | - | \bar{c}_v | = | \bar{c}_x' | - | \bar{c}_u |,$$

$$\text{state}(\bar{c}_x') = \text{state}(\bar{c}_x) = \text{state}(\bar{c}_y), \qquad \text{state}(c_2'') = \text{state}(c_2'),$$

$$\text{and} \qquad | c_2'' | - | \bar{c}_x' | = | c_2' | - | \bar{c}_y |.$$

Thus, we have, since we are eliminating equal increases and decreases,

$$\begin{aligned} | c_2' | - | c_2 | &= (| c_2' | - | \bar{c}_y |) + (| \bar{c}_y | - | \bar{c}_x |) + (| \bar{c}_x | - | \bar{c}_v |) \\ &\quad + (| \bar{c}_v | - | \bar{c}_u |) + (| \bar{c}_u | - | c_2 |) \\ &= (| c_2' | - | \bar{c}_y |) + (| \bar{c}_x | - | \bar{c}_v |) + (| \bar{c}_u | - | c_2 |) \\ &= (| c_2'' | - | \bar{c}_x' |) + (| \bar{c}_x' | - | \bar{c}_u |) + (| \bar{c}_u | - | c_2 |) \\ &= | c_2'' | - | c_2 |. \end{aligned}$$

So $| c_2' | = | c_2'' |$ and the stack increase is "right".

It remains to show that $\text{RLIVE}(c_2'') \geqslant r - 2d$. We have $c_1 \equiv c_2$ and thus

$$c_2' \equiv c_1' \equiv c_2'' .$$

Hence, input $\overline{w}$ must take c_2'' to a live configuration equivalent to c_3. If $T(c_3)$ contains nonempty words, then certainly we can assume that $\overline{w}$ takes c_2'' to a live configuration c_3' in reading mode. If $T(c_3) = \{e\}$, then $\overline{w}$ takes c_2'' to an accepting configuration c_3' and, since M_2 is seminormalized, c_3' is in reading mode and live ($T(c_3') = \{e\}$). Thus, in all cases, we have $c_2'' \to^{\overline{w}} c_3'$ with c_3' live and in reading mode. Since $\text{state}(c_2') = \text{state}(c_2'')$, we have $| c_3' | = | c_3 |$. Thus $\text{RLIVE}(c_2'') = \text{RLIVE}(c_2') \geqslant r - 2d$. So $Q(uxz, r - 2d)$ holds with $| uxz | < w$, a contradiction. ∎

Now we show that, if stack height increases by k and then decreases by k during C_1, such a matched push-pop sequence must occur.

CLAIM 2. *Computation C_1 cannot have a subcomputation during which the stack height increases and then decreases by k. Hence, C_1 cannot have a subcomputation during which the stack height increases by $k + d$.*

Proof. Suppose such a subcomputation existed. Let $t = k/(d+1) = s^4g^2 + 1$. Since M_1 is of delay d and 1-increasing, any increase or decrease of $d+1$ units must involve reading at least one input symbol. We divide the "bad" subcomputation into parts which "push" or "pop" $d+1$ symbols on the store. We write $w = \bar{u}v_1 \cdots v_t\bar{x}y_t \cdots y_1\bar{z}$ and find configurations $c_{v,i}$ and $c_{y,i}$ with:

$$c_1 \xrightarrow{\bar{u}} c_{v,1}, \qquad c_{y,1} \xrightarrow{\bar{z}} c_1', \qquad c_{v,t+1} \uparrow (\bar{x})\, c_{y,t+1},$$

$$\left.\begin{array}{c} c_{v,i} \uparrow (v_i)\, c_{v,i+1}, \qquad c_{y,i+1} \downarrow (y_i)\, c_{y,i} \\ |c_{v,i+1}| - |c_{v,i}| = |c_{y,i+1}| - |c_{y,i}| = d+1 \end{array}\right\} \quad 1 \leqslant i \leqslant t,$$

$$c_{v,t+1} = c_{y,t+1}.$$

Notice that $v_i \neq e \neq y_i$. We can similarly divide the C_2 computation:

$$\bar{c}_2 \xrightarrow{\bar{u}} \bar{c}_{v,1} \xrightarrow{v_1} \bar{c}_{v,2} \longrightarrow \cdots \xrightarrow{v_t} \bar{c}_{v,t+1} \xrightarrow{\bar{x}} \bar{c}_{y,t+1} \xrightarrow{y_t} \bar{c}_{y,t} \longrightarrow \cdots \xrightarrow{y_1} \bar{c}_{y,1} \xrightarrow{\bar{z}} c_2'$$

with each $\bar{c}_{v,i}$ and $\bar{c}_{y,i}$ in reading mode.

There are only s^4g^2 possible quadruples:

$$(\text{mode}(c_{v,i}), \text{mode}(c_{y,i}), \text{state}(\bar{c}_{v,i}), \text{state}(\bar{c}_{y,i})),$$

and so two must reappear. That is, for some n, $m \geqslant 1$, we must have

$$\text{mode}(c_{v,n}) = \text{mode}(c_{v,n+m}),\ \text{mode}(c_{y,n}) = \text{mode}(c_{y,n+m}),$$
$$\text{state}(\bar{c}_{v,n}) = \text{state}(\bar{c}_{v,n+m}) \text{ and } \text{state}(\bar{c}_{y,n}) = \text{state}(\bar{c}_{y,n+m}).$$

Let $u = \bar{u}v_1 \cdots v_{n-1}$, $v = v_n \cdots v_{n+m-1}$, $x = v_{n+m} \cdots v_t\bar{x}y_t \cdots y_{n+m}$, $y = y_{n+m-1} \cdots y_n$ and $z = y_{n-1} \cdots y_1\bar{z}$.

Clearly, the computation

$$c_1 \xrightarrow{u} c_{v,n} \uparrow (v)\, c_{v,n+m} \uparrow (x)\, c_{y,n+m} \downarrow (y)\, c_{y,n} \xrightarrow{z} c_1$$

and the corresponding division of C_2 violate CLAIM 1.

Hence, C_1 cannot have a subcomputation during which the stack height increases and then decreases by k. Since $|c_1| \geqslant |c_1'| - d$, if stack height increased by $k+d$ during C_1, it must subsequently decrease by k. ∎

Finally, we contradict CLAIM 2 by showing that C_1 must have a subcomputation which increases the stack height by at least $k + d$.

CLAIM 3. *Computation C_1 must have a subcomputation which increases stack height by at least $k + d$.*

Proof. Consider the portion of C_2 which last increases the stack height by $r - 2d$, and the portion of C_3 which decreases it accordingly. Let $t = (r - 2d)/(d + 1) = (k + d + 1)(s^4g^2 + 1)$. This time, we find a push-pop sequence in C_2 and C_3 and the corresponding computation of M_1. In C_2, there are t "pushes" of $d + 1$ symbols from configuration $c_{v,i}$ to $c_{v,i+1}$ with the corresponding "pops" from $c_{y,i+1}$ to $c_{y,i}$ in C_3. This time, we find the corresponding configurations $\bar{c}_{v,i}$ and $\bar{c}_{y,i}$ in reading mode in C_1 and in C_4: $c_1' \xrightarrow{\bar{w}} c_3$. A quadruple (mode($c_{v,i}$), mode($c_{y,i}$), state($\bar{c}_{v,i}$) state($\bar{c}_{y,i}$)) must occur $k + d + 1$ times.

Let these repetitions occur at i_j, $1 \leqslant i_1 < \cdots < i_{k+d+1} \leqslant t$. The crucial point is that Lemma A.1(2) tells us that there must be a stack increase of at least 1 from $\bar{c}_{v,i_j}$ to $\bar{c}_{v,i_{j+1}}$ and hence, a total increase of at least $k + d$ from c_{v,i_1} to $\bar{c}_{v,i_{k+d+1}}$. ∎

CLAIM 3 contradicts CLAIM 2, which completes the proof of the Lemma. ∎

The final lemma (A.3) says that the class of superdeterministic *dpdas* has the matched popping property. The bound obtained is exponentially worse than the one for the matched pushing number, since the proof requires forcing the stack height in one of the computations to grow.

LEMMA A.3. *Let M_1 and M_2 be seminormalized superdeterministic dpdas of delay d with at most s states and g pds symbols. Let*

$$k_1 = (s^4g^2 + 1)(d + 1)$$
$$k_2 = (k_1 + 1)^2$$

and

$$r = 3d + (d + 1)((s + 1)\, sg^{2k_2+2d+1} + 1).$$

Then r is a matched popping number for M_1 and M_2.

Proof. Let RLIVE(c) be defined as in the proof of Lemma A.2. This time, we consider property $Q(w, t)$.

$Q(w, t)$: There are equivalent accessible configurations c_1 in M_1 and c_2 in M_2 such that

$$c_1 \downarrow (w)\, c_1', \qquad c_2 \xrightarrow{w} c_2',$$

c_1' and c_2' live and in reading mode,

$$|c_1| \geqslant |c_1'| + t, \qquad |c_2'| \geqslant |c_2| - d, \qquad \text{RLIVE}(c_2') \geqslant t.$$

The arguments we used before show that r is a matched popping number if $Q(w, r-2d)$ fails for all w. So suppose to the contrary that w is a minimum length word for which $Q(w, r - 2d)$ holds.

We have computations

$$C_1\colon c_1 \downarrow (w)\, c_1' \text{ in } M_1, \qquad C_2\colon c_2 \xrightarrow{w} c_2' \text{ in } M_2, \qquad c_1 \text{ and } c_2 \text{ accessible},$$

$$c_1 \equiv c_2, \text{ and } c_1' \text{ and } c_2' \text{ are live and in reading mode},$$

$$|c_1| \geqslant |c_1'| + r - 2d, \qquad |c_2'| \geqslant |c_2| - d, \qquad \text{RLIVE}(c_2') \geqslant r - 2d.$$

By the same arguments used in the proof of Lemma A.2, we can establish the following CLAIMS; the proofs are omitted.

CLAIM 1. *Computation C_1 cannot have a subcomputation during which the stack height increases and then decreases by at least k_1. Hence, C_1 cannot have a subcomputation during which the stack increases by at least k_1.*

CLAIM 2. *Computation C_2 cannot have a subcomputation during which the stack height increases and then decreases by at least k_1.*

Now, we show that C_2 must have a subcomputation which increases the stack height by at least k_2. The essential point is that the maximum difference between $|c_2|$ and the stack height of any configuration of C_2 must grow at least as the log (base g) of the number of steps in C_2 and, if C_2 has a decrease in stack of t, it must have a subsequent increase of $t - d$.

CLAIM 3. *Configuration C_2 must have a subcomputation during which the stack height increases by at least k_2.*

Proof. First, suppose that, for all configurations c of C_2, $|\,|c_2| - |c|\,| \leqslant k_2 + d$. Hence, there are at most $t = sg^{2k_2+2d+1}$ distinct configurations which can occur during C_2. Now C_1 involves a stack drop of at least $r - 2d$ and M_1 is of delay d, so $d + (d+1)|w| \geqslant r - 2d$, or $|w| \geqslant (r - 3d)/(d+1)$. In addition, C_2 must have at least as many steps as the length of w, so C_2 must involve at least

$$(r - 3d)/(d+1) = (s+1)\, sg^{2k_2+2d+1} + 1 \text{ steps}.$$

Hence, some configuration c must repeat at least $s + 1$ times. Since M_2 is finite delay, the input between these repeats must be nonempty. So, for two occurrences of c, the corresponding configurations of M_1 in reading mode must be in the same state. That is, we must have $w = uvx$ with $v \neq e$, and

$$c_1 \xrightarrow{u} c_u \xrightarrow{v} c_v \xrightarrow{x} c_1'$$

$$c_2 \xrightarrow{u} c \xrightarrow{v} c \xrightarrow{x} c_2'$$

with c_u and c_v in reading mode and state(c_u) = state(c_v). By Lemma A.1(1), $|c_v| \geqslant |c_u|$.

Hence, we have:

$$c_2 \xrightarrow{ux} c_2', \qquad c_1 \xrightarrow{u} c_u \xrightarrow{x} c_1'',$$

with c_1'' in reading mode and $|c_u| - |c_1''| = |c_v| - |c_1'|$. Thus

$$\begin{aligned} |c_1| - |c_1''| &= (|c_1| - |c_u|) + (|c_u| - |c_1''|) = (|c_1| - |c_u|) + (|c_v| - |c_1'|) \\ &\geqslant (|c_1| - |c_u|) + (|c_u| - |c_v|) + (|c_v| - |c_1'|) \\ &= |c_1| - |c_1'| > r - 2d. \end{aligned}$$

So, $Q(ux, r - 2d)$ holds, contradicting the minimality of w.

Thus, C_2 must contain a configuration c with either $|c| \geqslant |c_2| + k_2 + d$ or else $|c| \leqslant |c_2| - k_2 - d$. In the first case, there must be a subcomputation with a stack height increase of at least $k_2 + d$ and, in the second case, a subcomputation with stack height decrease of at least $k_2 + d$. But, since $|c_2'| \geqslant |c_2| - d$, any stack height decrease of $k_2 + d$ must be followed by a stack height increase of at least k_2. ∎

By CLAIM 3, we know that C_2 must contain a stack height increase of at least k_2. By CLAIM 2, this cannot be followed by a stack height decrease of k_1. Thus, we must have $w = uv$ and

$$C_2\colon c_2 \xrightarrow{u} c \uparrow (v)\, c_2', \qquad |c_2'| > |c| + k_1(k_1 + 1) + 1.$$

Since $\mathrm{RLIVE}(c_2') > r - 2d$, we certainly have an input z and computation

$$C_3\colon c_2' \downarrow (z)\, c_3, \qquad |c_2'| > |c_3| + k_1(k_1 + 1) + 1.$$

So there must be a stack height increase of $k_1(k_1 + 1) + 1$ during C_2 followed by a matching decrease during C_3. Hence, the arguments used in the proof of CLAIM 3 of Lemma A.2 show that there must be a stack increase of at least k_1 during C_1, contradicting CLAIM 1. This completes the proof. ∎

References

1. C. Beeri, An improvement on Valiant's decision procedure for equivalence of deterministic finite turn pushdown machines, *Theor. Comput. Sci.* **3** (1976), 305–320.
2. E. P. Friedman, Equivalence problems in monadic recursion schemes, *in* "Proc. IEEE 14th Ann. Symp. on Switching and Automata Theory, University of Iowa, Iowa City, October 1973," pp. 26–33.
3. E. P. Friedman, The inclusion problem for simple languages, *Theor. Comput. Sci.* **1** (1976), 297–316.
4. E. P. Friedman, Equivalence problems for deterministic context-free languages and monadic recursion schemes, *J. Comput. System Sci.* **14** (1977), 344–359.
5. E. P. Friedman, Simple context-free languages and free monadic recursion schemes, *Math. Systems Theory* **11** (1977), 9–28.

6. E. P. FRIEDMAN AND S. A. GREIBACH, Length uniformity in grammars and machines, in preparation.
7. S. GINSBURG AND S. GREIBACH, Deterministic context-free languages, *Inform. Contr.* **9** (1966), 620–648.
8. S. GINSBURG, S. GREIBACH, AND M. A. HARRISON, One-way stack automata, *J. Assoc. Comput. Mach.* **14** (1967), 389–418.
9. S. GREIBACH, Characteristic and ultrarealtime languages, *Inform. Contr.* **18** (1971), 65–98.
10. S. A. GREIBACH AND E. P. FRIEDMAN, Superdeterministic Pdas: A subcase with a decidable equivalence problem, submitted for publication.
11. M. A. HARRISON, I. M. HAVEL, AND A. YEHUDAI, On equivalence of grammars through transformation trees, in press.
12. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1968.
13. A. J. KORENJAK AND J. E. HOPCROFT, Simple deterministic languages, *in* "Proc. IEEE 7th Ann. Symp. on Switching and Automata Theory, Berkeley, Calif. 1966, pp. 36–46.
14. M. LINNA, Two decidability results for deterministic pushdown automata, *J. Comput. System Sci.* **18** (1979), 92–107.
15. T. OLSHANSKY AND A. PNUELI, A direct algorithm for checking equivalence of LL(k) grammars *Theor. Comput. Sci.* **4** (1977), 321–349.
16. M. OYAMAGUCHI AND N. HONDA, The decidability of equivalence for deterministic stateless pushdown automata, *Inform. Contr.* **38** (1978), 367–376.
17. M. OYAMAGUCHI, N. HONDA, AND Y. INAGAKI, The equivalence problem for realtime strict deterministic languages, unpublished manuscript.
18. E. POST, A variant of a recursively unsolvable problem, *Bull. Amer. Math. Soc.* **52** 264–268.
19. K. TANAGUCHI AND T. KASAMI, A result on the equivalence problem for deterministic pushdown automata, *J. Comput. System Sci.* **13** (1976), 38–50.
20. L. G. VALIANT, "Decision Procedures for Families of Deterministic Pushdown Automata," Ph. D. dissertation, University of Warwick, U.K., July 1973.
21. L. G. VALIANT, The equivalence problem for deterministic finite-turn pushdown automata, *Inform. Contr.* **25** (1975), 123–133.
22. L. G. VALIANT AND M. S. PATERSON, Deterministic one-counter automata, *J. Comput. System Sci.* **10** (1975), 340–350.
23. D. WOOD, Some remarks on the KH algorithm for *s*-grammars, *BIT* **13** (1973), 476–489.
24. V. A. YAFFE, On two classes of CF-languages with a solvable equivalence problem, *Kibernetika (Kiev)* **2** (1974), 89–93 (in English translation).
25. Y. IGARASHI, Tape bounds for some subclasses of deterministic context-free languages, *Inform. Contr.* **37** (1978), 321–333.

Printed in Belgium