JOURNAL OF COMPUTER AND SYSTEM SCIENCES 37, 269–291 (1988)

Pushdown Automata with Reversal-Bounded Counters*

TAT-HUNG CHAN

Centre of Computer Studies and Applications,
University of Hong Kong, Pokfulam Road, Hong Kong

Received September 21, 1981; revised September 30, 1987

The main result of this paper is that a pushdown automaton M augmented with $R(n)$ reversal-bounded counters can be simulated by a Turing machine in time polynomial in $n^{n^2} + R(n)$ if M is 2-way, and in time polynomial in $n + R(n)$ if M is 1-way. It follows from this and previous results that for sufficiently large $R(n)$, the addition of a pushdown store to $R(n)$ reversal-bounded multicounter machines has little effect on the computing powers of the machines. The proof of the main result yields three interesting corollaries. First, relaxing the reversal bound from one counter in an $R(n)$ reversal-bounded multicounter machine leads to very little increase in computing power. Second, 1-way pushdown automata augmented with 1-reversal counters accept in linear time and are therefore equivalent to 1-way simple multihead pushdown automata. Third, for every 1-way pushdown automaton, there is a constant c such that every accepting computation on an arbitrary nonempty input w contains a subsequence (of not necessarily consecutive operations) which is also an accepting computation on w and which has length at most $c|w|$.

1. INTRODUCTION

An *auxiliary pushdown automaton* (APDA) is a 2-way pushdown automaton (PDA) augmented with an auxiliary read–write worktape. An APDA is said to be $s(n)$ *space-bounded* if it accepts strings of length n using at most $s(n)$ space on the auxiliary worktape. This definition imposes no *a priori* bound on the growth of the pushdown store. Nevertheless, Cook [4] showed that if $s(n) \geqslant \log_2 n$, then both the class of nondeterministic $s(n)$ space-bounded APDAs and the class of deterministic $s(n)$ space-bounded APDAs accept exactly the same class of languages as the class of deterministic Turing machines (TMs) with time bounds that are polynomial in $2^{s(n)}$. Various interesting consequences follow from this result, for example, that both deterministic and nondeterministic 2-way multihead PDAs accept exactly the class **P** of languages recognizable by polynomially time-bounded TMs.

In this paper, we consider a variant of the APDA, namely, a 2-way PDA augmented with counters (pushdown stores with a single-letter alphabet). Clearly even 1-way PDAs augmented with just one unrestricted counter are capable of

* The first draft of this paper was written while the author was with the Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

0022-0000/88 $3.00

accepting all r.e. sets [10]. On the other hand, Ibarra [9] showed that if the counters are allowed to make only a constant number of reversals in any computation, then the augmented PDAs accept only recursive sets. Here we study machines with general recursive reversal bounds imposed on the counters. Our motivation is the following consideration. While it is not known whether the addition of a pushdown store increases the power of $s(n)$ space-bounded multitape TMs for $s(n) \geqslant \log_2 n$, it is obvious that such an addition does not increase the power of time-bounded multitape TMs. Now for bounds that are at least linear, it was shown in [3] that reversal on multicounter machines (1-way or 2-way) is polynomially related to time on TMs. One might therefore suspect that for $R(n) \geqslant n$, the addition of a pushdown store has little effect on the computational powers of $R(n)$ reversal-bounded multicounter machines. The results presented here show that this is indeed the case for nondeterministic machines when $R(n)$ is sufficiently large. Specifically, we show that a 2-way nondeterministic PDA with $R(n)$ reversal-bounded counters can be simulated by a nondeterministic multitape TM with time bound $p(n^{n^2} + R(n))$ for some polynomial p. In the special case of 1-way machines, the time bound of the simulating TM can be lowered to $p(n + R(n))$ for some polynomial p. Hence the above conjecture holds for 2-way machines with $R(n) \geqslant n^{n^2}$ and for 1-way machines with $R(n) \geqslant n$. Our technique is a natural extension of that of [3]—itself derived from a proof from [1]—to include a pushdown store.

Two other special cases are also studied. First, a 1-way PDA with constant reversal-bounded counters is a natural combination of a 1-way PDA and a 1-way multicounter machine with constant reversal bounds. Now both component machines are known to accept in linear time ([5, 1], respectively). It turns out that our analysis of the general model also shows that this special model accepts in linear time and so is equivalent to the 1-way simple multihead PDA of [8]. Second, if the pushdown store is restricted to be a counter, then the general model becomes a 2-way $R(n)$ reversal-bounded multicounter machine with the reversal bound removed from one of the counters. We show that as in [3] the polynomial relation to TM time holds for $R(n) \geqslant n$. In fact, the simulation time bounds are not much worse than those in [3].

It is assumed that the reader is familiar with PDAs, APDAs, TMs, and multicounter machines. Detailed accounts of these automata can be found in [7]. Definitions of less familiar notions will be given when the need arises. We conclude this section by stating a result on linear diophantine systems that plays a crucial role in our proofs:

LEMMA 1.1 (Borosh and Treybig [2]). *For an arbitrary linear diophantine system $Ax = b$, if there is a nonnegative solution, then there is one in which every entry of x is bounded by $rsM_A^2|b|$, where r is the number of rows, s is the number of columns, and M_A is the maximum absolute value of all minors in the matrix A.*

2. Two-Way Pushdown Automata with Reversal-Bounded Counters

In this section, we study the complexity of a 2-way PDA augmented by k $R(n)$ reversal-bounded counters. This is a computing device equipped with a nondeterministic finite control, a 2-way read-only input tape holding an input string from an alphabet Σ delimited by special endmarkers $\mathcal{L}$ and \$ not in Σ, a pushdown store with alphabet Γ containing a special bottom marker, and k counters. We shall use the terms "pushdown store" and "stack" interchangeably. A step in the computation of such a machine depends on the state of the finite control, the input symbol being scanned, the top symbol on the pushdown store, and the subset of the counters which are empty. For each possible combination, the transition function specifies a finite number of choices of action, and the machine can nondeterministically follow any one of these choices to attain the next machine configuration. Each choice of action may involve any combination of the following: (1) change in the control state, (2) motion of the input head by one square in either direction, (3) pushing or popping the pushdown store or one of the counters. Note that an action that changes the stack cannot also change one or more of the counters. Also note that each push or pop involves only one symbol, so it takes two steps to change the top symbol in the pushdown store. It is also assumed that the machine will not attempt to move its input head off the delimited input, push or pop the special bottom marker of the pushdown store, or decrement an empty counter. Initially the finite control is in the initial state, the input head is positioned on the left endmarker $\mathcal{L}$, the pushdown store contains only the bottom marker, and all counters are empty. The machine is said to accept the input if there is some computation that leads from the initial configuration to one in which (1) the finite control is in an accepting state, (2) all counters are empty, and (3) the pushdown store contains only the bottom marker.

In any computation, each of the k counters can independently increase or decrease. Without loss of generality, we assume that each counter will switch from increasing to decreasing only when it has size at least 2. We can also assume that each counter will switch from decreasing to increasing only when it is empty, since the machine can empty and then restore the counter with the aid of the pushdown store. We define a reversal of a counter to encompass a growth from zero and the subsequent decrement back to zero. "$R(n)$ reversal-bounded" means that for any input of length n that is accepted, there is an accepting computation in which each counter goes through at most $R(n)$ reversals.

To derive an upper bound on the time required to simulate such an augmented PDA M on a multitape TM, we use the following strategy. We begin by establishing a "normal form" for accepting computations of the augmented machines. This normal form serves two purposes. First, it admits a succinct description in terms of a system of linear diophantine equations, from which an upper bound on the time of a shortest normal-form accepting computation can be obtained. Second, it facilitates time-efficient simulation by multitape TMs. Putting these results together, we obtain a time bound on the simulating TM.

Suppose M has q states $s_1, ..., s_q$. Think of a computation as a string of operations. We say a point P is "in" the string if it is adjacent to an operation in the string; we say P is "within" the string if it is between two consecutive operations in the string. Consider an accepting computation of M on an input string w of length n. We can assume that each counter makes at most $R(n)$ reversals in this computation. Define the *status* of counter i at each point in the computation as one of the values 0, 1, 2, ..., $3r_i$, where $r_i \leqslant R(n)$ is the total number of reversals made by counter i, such that when the status has value m, counter i has completed $\lfloor m/3 \rfloor$ reversals and is currently empty if $m \equiv 0 \pmod 3$, nonempty and increasing (i.e., the last operation on the counter was an increment) if $m \equiv 1 \pmod 3$, and nonempty and decreasing if $m \equiv 2 \pmod 3$. Hence at each point the statuses of the k counters together are represented by a k-vector $(m_1, ..., m_k)$. We call this the *counter status vector*, abbreviated CSV. Clearly the CSV has initial value $(0, ..., 0)$ and final value $(3r_1, ..., 3r_k)$, and changes value exactly $3\Sigma r_i$ times in the computation.

At any point in the computation, the *total configuration* of M is a vector $(s, p, \sigma, \tau, x_1, ..., x_k)$, where s is the current state, p is the current input head position $(0 \leqslant p \leqslant n+1)$, $\sigma \in \Gamma^+$ is the current stack content with the top symbol at the left, τ is the current CSV, and the nonnegative integers $x_1, ..., x_k$ are the current counter contents. Note that τ contains information about $x_1, ..., x_k$: it indicates which of them are zero. Very often we shall be interested in only part of the total configuration. Thus we define a *stack configuration* to be a vector (s, p, σ, τ) and a *surface configuration* to be a vector (s, p, b, τ), where $b \in \Gamma$ and the other components are as before. From each total configuration we obtain a unique stack configuration by deleting the counter values, and from each stack configuration we obtain a surface configuration by deleting all but the top symbol from the stack σ. Note that the surface configuration is sufficient for determining which operation may be performed next, but not for determining the surface configuration after the chosen operation.

We now proceed to manipulate the given accepting computation on w into a suitable normal form. Intuitively, this normal form has a simple structure in the sense that most of the operations are in a "small" number of substrings each having the form z^r, where z is a "short" string of operations and r is a positive integer. The notions "small' and "short" will be formalized in the sequel. As in [3, 6], we divide this process into three phases: marking, deletion, and reinsertion. Marking serves to identify the set of points at which substrings deleted in the second phase will be reinserted in the third phase to form the substrings z^r (after further replacements and permutations).

Marking

First of all, we mark all operations that change the CSV. There are $3\Sigma r_i$, or more simply $O(R(n))$, such operations. Next for each ordered pair of surface configurations $\gamma_1 = (s_1, p_1, b, \tau_1)$ and $\gamma_2 = (s_2, p_2, b, \tau_2)$ with the same top stack symbol b, find, if any, arbitrary but exactly one pair of points (P_1, P_2) with stack

configurations $(s_1, p_1, \sigma, \tau_1)$, $(s_2, p_2, \sigma, \tau_2)$, respectively, with the same stack σ, such that

(i) b is the top symbol of σ, and

(ii) $|\sigma'| > |\sigma|$ at each point P' between P_1 and P_2 with stack σ'.

If these points can be found, mark the two operations adjacent to P_1 (only one if P_1 is the initial point) as well as those adjacent to P_2. We do not allow a point to be chosen for two distinct pairs. Finally, for each surface configuration γ we find, if any, arbitrary but exactly one corresponding point and mark all operations adjacent to it. Here we may use points already chosen above.

LEMMA 2.1. *At most $O(n^2R(n))$ operations may be marked.*

Proof. Clearly, it suffices to show that in the second stage, the marking algorithm can find pairs of points P_1 and P_2 for at most $O(n^2R(n))$ pairs of surface configurations. Since there are at most $O(n^2R(n))$ pairs with $\tau_1 = \tau_2$, it suffices to argue that marking results for at most $O(n^2R(n))$ out of the $O(n^2R(n)^2)$ possible pairs of surface configurations with $\tau_1 \neq \tau_2$. We shall make use of the following claim:

> Let m be any integer $\geqslant 2$. Take an arbitrary string of properly nested parentheses and arbitrarily divide it into m segments colored $c_1, ..., c_m$, respectively. We say a pair of distinct colors (c_i, c_j) $(i<j)$ is *matched* if there exists a left parenthesis colored c_i such that its corresponding right parenthesis is colored c_j (note that for each c_i, there may be more than one c_j such that (c_i, c_j) is matched). Then at most $2m-3$ pairs of distinct colors are matched no matter how the division into m differently colored segments is done.

This claim is proved by induction on m. The base case $m=2$ is immediate. For the inductive step $m \geqslant 3$, consider two cases:

Case 1. There is no k such that $1<k<m$ and (c_1, c_k) is matched. Then the pairs that are matched can be either (c_i, c_j) with $2 \leqslant i < j \leqslant m$, or (c_1, c_m). If we repaint the c_1-segment with the color c_2, all matched pairs other than (c_1, c_m) are preserved. By induction, in the new coloring, at most $2(m-1)-3 = 2m-5$ pairs are matched. Hence in the original coloring, at most $2m-4$ pairs are matched.

Case 2. There is some k such that $1<k<m$ and (c_1, c_k) is matched. Let h be the largest such k. Then because of proper nesting of parentheses, no pair (c_i, c_j) with $1<i<h<j$ can be matched. Hence the matched pairs may only be (1) (c_i, c_j) with $1 \leqslant i < j \leqslant h$, (2) (c_i, c_j) with $h \leqslant i < j \leqslant m$, or (3) (c_1, c_m). Consider repainting the segments colored $c_{h+1}, ..., c_m$ with the color c_h. The matched pairs of type (1) in the original coloring are not destroyed in the new coloring, which by induction contains at most $2h-3$ matched pairs. Similarly, the number of matched pairs of

type (2) is at most $2(m-h+1)-3=2m-2h-1$. Hence the total is at most $(2h-3)+(2m-2h-1)+1=2m-3$.

We now return to showing that there are at most $O(n^2R(n))$ pairs of surface configurations with $\tau_1 \neq \tau_2$ for which marking occurs. Treat each CSV as a distinct color, and paint each operation with the CSV that holds at the point immediately before the operation. Thus we have divided the computation into $O(R(n))$ differently colored segments.

Now delete all operations in the computation except those that perform a push or a pop on the stack. The string of remaining operations may be considered as a string of properly nested parentheses by treating each push as a left parenthesis and each pop as a right parenthesis. The coloring of the original string induces a division of the residual string into $O(R(n))$ differently colored segments. Now consider a pair of points P_1 and P_2 with $\tau_1 \neq \tau_2$ which is chosen for marking. If P_1 and P_2 are originally separated by one operation, then that operation causes a change in CSV. Since there are $O(R(n))$ operations that change the CSV, it only remains to show that marking occurs for $O(n^2R(n))$ pairs that are separated by two or more operations before the deletion. Now for such a pair, because of criterion (ii) for the choice of pairs of points, it must be the case that P_1 is followed immediately by a push and P_2 is preceded immediately by the corresponding pop. Both operations survive the deletion. Furthermore the CSV before the pop must also be τ_2, since we assume that our machines cannot change both the stack and a counter in the same operation. Hence (τ_1, τ_2) is matched. But by the claim above, at most $O(R(n))$ pairs (τ_1, τ_2) can be matched; each such pair leads to at most $O(n^2)$ pairs of surface configurations (allowing for all possible choices of s_1, p_1, s_2, p_2, and b). Hence marking occurs for at most $O(n^2R(n))$ pairs of surface configurations with different CSVs. ▌

Deletion

In this phase (not to be confused with the deletion used in the proof of Lemma 2.1), we delete unmarked operations from the initially valid computation string, using Algorithm D to be presented below. We shall think of this computation as a string of operations with surface configurations labelling all points at both ends and between every two consecutive operations. We then perform "cut-and-pastes" with respect to surface configurations. That is, at every step, we shall delete a substring of unmarked operations only if its two ends are labelled with the same surface configuration. Thus in each residual string, every point "inherits" unambiguously a surface configuration from the initially valid computation, and the operation immediately following the point is allowable by the transition function of M under the surface configuration. Of course, these residual strings may no longer represent valid computations. For example, an operation that in the initial computation takes place when a counter is zero may now occur—after removal of preceding operations which decrement the counter—at a point where the counter is nonzero. However, it will be clear from the algorithm that the

inherited surface configurations truly reflect the effects of the remaining operations—possibly executed "against the rule" at some points—except for the CSV, which may be viewed as being dictated by the marked operations that change its values. We shall refer to the residual strings as *residual computations* or simply computations, and we say they are *valid modulo the counters*. Configurations in residual computations are defined as follows. The surface configuration at a point is inherited from the initial valid computation as described above. For the stack configuration and the total configuration, the CSV τ is the same as that of the surface configuration while the other components are the results of actually simulating the operations. Clearly the expected relations among these configurations still hold at each point.

Next for an arbitrary positive integer h we define an *h-loop* in a computation to be a nonempty string of operations such that

(i) the initial and final stack configurations are the same, with stack σ

(ii) at all interior points in the sequence, the stack σ' satisfies $|\sigma| \leqslant |\sigma'| \leqslant |\sigma| + h$

(iii) none of the operations in the sequence is marked.

Clearly σ remains unchanged at the bottom of the stack throughout the h-loop. For a string of operations, we define the *stack variation* to be the difference between the maximum and the minimum stack heights in the course of the string. Thus an h-loop has stack variation at most h.

LEMMA 2.2. *For each* $h \geqslant 0$, *let* $L(h) = (q(n+2)+3)^{h+1}$. *In a residual computation, any string of operations which satisfies the following conditions must contain an h-loop as a proper substring*:

(i) *the sequence contains at least* $L(h)$ *operations*

(ii) *none of the operations in the sequence is marked*

(iii) *the stack variation of the sequence is at most h.*

Proof. By induction on h. For the base case $h = 0$, the stack, and hence also the top stack symbol, are unchanged throughout the string of operations. The CSV is also constant because there is no marked operation. Delete the first operation. We are still left with $L(0) - 1$ operations, and hence $L(0) = q(n+2)+3$ points. By the pigeonhole principle, there must be two points which have the same s and p components in their surface configurations. The substring delimited by these points is the desired 0-loop.

For the inductive step, assume the lemma holds for some $h \geqslant 0$, and consider a string of $L(h+1)$ unmarked operations with stack variation at most $h+1$. Let H_0 and H_1 be the minimum and maximum stack heights in this string (so that $H_1 - H_0 \leqslant h+1$), and let m be the number of points at which the stack height is H_0. There are two cases.

Case 1. $m \leqslant q(n+2)+1$. Then the m points divide the substring into at most $m+1$ segments, at least one of which has length at least

$$\frac{L(h+1)}{m+1} > \frac{L(h+1)-1}{m+1} \geqslant \frac{L(h+1)-1}{q(n+2)+2}.$$

Now

$$\frac{L(h+1)-1}{q(n+2)+2} = \frac{(q(n+2)+3)^{h+2}-1}{(q(n+2)+3)-1}$$
$$= (q(n+2)+3)^{h+1} + (q(n+2)+3)^{h} + \cdots + 1 \geqslant L(h)+1.$$

Hence there is a substring of length $> L(h)+1$, i.e., $\geqslant L(h)+2$ such that at every point *within* the substring, the stack height is at least H_0+1. Deleting the first and last operations, we obtain a substring of length at least $L(h)$ with stack variation at most $H_1-(H_0+1) \leqslant h$. By induction, this substring contains an h-loop, which is perforce an $(h+1)$-loop.

Case 2. $m \geqslant q(n+2)+2$. The proper substring delimited by the second and the mth points contains at least $q(n+2)+1$ points with stack height H_0, from which we derive an $(h+1)$-loop by the pigeonhole principle as in the base case. ▮

The next notion we introduce is that of a *peak*. This is a string of operations which begins and ends with the same stack σ and such that at every interior point the stack σ' satisfies $|\sigma'| > |\sigma|$. Thus the criterion for the choice of a pair of points in the marking phase is that they have the appropriate surface configurations and that they delimit a peak.

Finally we define an *arm pair*. Intuitively, this consists of two strings of unmarked operations—the *arms*—such that the first arm pushes a string on the stack which is subsequently popped by the second arm. Furthermore, each arm begins and ends with the same surface configuration, so that their simultaneous removal preserves validity *modulo* the counters. Formally, let P_1, P_2, P_3, and P_4 be four points in that order in a computation, satisfying the conditions

(i) P_1 and P_2 have the same surface configuration (s_1, p_1, b, τ_1)

(ii) P_3 and P_4 have the same surface configuration (s_2, p_2, b, τ_2)

(iii) P_1 and P_4 delimit a peak with initial stack σ_1

(iv) P_2 and P_3 delimit a peak with initial stack σ_2

(v) the operations between P_1 and P_2, as well as those between P_3 and P_4, are all unmarked.

Then the string of operations between P_1 and P_2 is the *left arm* and the string of operations between P_3 and P_4 is the *right arm* of an arm pair (see Fig. 1). Notice that the stacks σ_1 and σ_2 have the same top symbol b. The length of an arm pair is the total number of operations in the two arms.

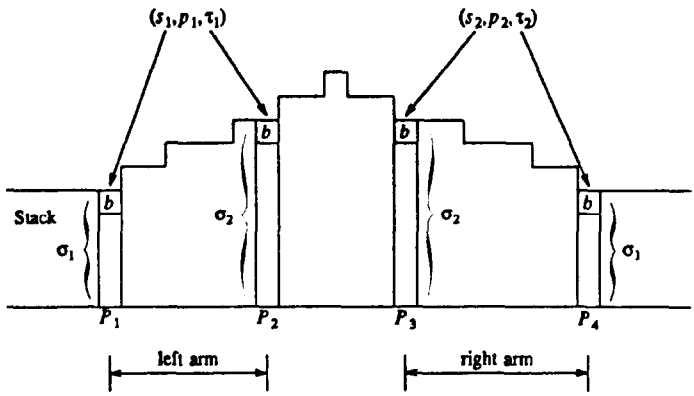

FIG. 1. An arm pair.

Let $h_1 = (q(n+2))^2|\Gamma| + 1$ and $h_2 = 2h_1$.

LEMMA 2.3. *Let α be a string of unmarked operations in a residual computation, with initial stack height H_0, final stack height H_1, and maximum stack height H. If $H - H_1 \geqslant h_1$ and $H - H_2 \geqslant h_1$, then α must properly contain an arm pair.*

Proof. Take a point P at which the stack height is H. Then for $i = 1, 2, ..., h_1$, consider the pair of points (P_i, P_i') such that $P_i(P_i')$ is the last (first) point before (after) P with stack height $H - i$. The existence of these points is guaranteed by the hypothesis of the lemma and by the fact that the stack height can change by at most one with each operation. For each i, P_i, and P_i' delimit a peak and the top stack symbols at P_i and P_i' must be the same. The CSV stays constant throughout α because there is no marked operation. Hence there are at most $(q(n+2))^2|\Gamma|$ distinct pairs of surface configurations that can occur at the pairs of points (P_i, P_i'). By the pigeonhole principle there are integers $i < j$ such that (P_i, P_i') and (P_j, P_j') have the same pair of surface configurations. P_j, P_i, P_i', and P_j' then yield an arm pair as desired. ∎

Remark. The proof of Lemma 2.3 can be used to obtain the following result of Greibach [5]: for a nondeterministic 2-way k-head PDA (with no additional counters), a shortest accepting computation on an input of length n will attain a maximum stack height of at most $q^2(n+2)^{2k}|\Gamma| + 1$; hence such a machine accepts in $O(n^{2k})$ space.

Having developed the necessary terminology, we are now in a position to present the deletion algorithm.

ALGORITHM D.

while there is an h_2-loop or an arm pair **do**
delete a minimal h_2-loop or arm pair;

Here a "minimal" h_2-loop is one that does not properly contain another h_2-loop or arm pair, and similarly for a "minimal" arm pair. Because we only delete h_2-loops and arm pairs, we do preserve validity *modulo* the counters as we claimed at the beginning of this phase. As the deletion proceeds, some points vanish altogether while others may become identified. For each point P in the initial computation that has not vanished, let $\bar{P}$ denote the point it becomes in the current residual computation. Conversely, for a point P in the current residual computation, let P^{L} (P^{R}) denote the leftmost (rightmost) point Q in the initial computation such that $\bar{Q} = P$.

LEMMA 2.4. *Suppose P_1 and P_2 are the points chosen for a pair of surface configurations in the marking phase (so that they delimit a peak initially). Then $\bar{P}_1$ and $\bar{P}_2$ exist and still delimit a peak in a residual computation. Conversely suppose P_1, P_2 are points in a residual computation that delimit a peak which begins and ends with unmarked operations. Then P_1^{R} (resp. P_2^{L}) has the same surface configuration as P_1 (resp. P_2), and P_1^{R}, P_2^{L} delimit a peak in the original computation.*

Proof. It suffices to prove this lemma for the special case in which only one h_2-loop or only one arm pair is deleted, since the extension to the general case can be achieved by an easy induction. In the rest of this proof, we shall not state explicitly whether we are talking about the original computation or the residual computation; this can be inferred easily from the context. For two points P, Q in a string, we shall use (P, Q) to denote the substring delimited by P and Q.

In the forward direction, it is clear that $\bar{P}_1$ and $\bar{P}_2$ exist because the operations adjacent to P_1 and P_2 are marked and so cannot be deleted. Because an h_2-loop contains only unmarked operations, it must be disjoint from the peak (P_1, P_2) or properly contained in it. In either case, it is obvious that $(\bar{P}_1, \bar{P}_2)$ is a peak. Similarly, each arm of an arm pair must be disjoint from (P_1, P_2) or properly contained in it. Clearly, if one arm is in the peak then so must be the other arm, and removal of both arms still leaves a peak.

In the other direction, we proceed by case analysis depending on the positions of P_1^{R} and P_2^{L} relative to the endpoints of the h_2-loop or arms. The arguments are simple but tedious, and are omitted. ∎

Lemma 2.4 leads to

LEMMA 2.5. (1) *For every h_2-loop deleted by the algorithm, there is an initially chosen point left in the residual computation with the same surface configuration as at the beginning of the h_2-loop.*

(2) *For every arm pair deleted, there is an initially chosen pair of points remaining in the residual computation with the same surface configurations as at the beginnings of the arms; furthermore, these points still delimit a peak.*

Proof. (1) is trivial because of the third stage in the marking phase and the fact that no marked operation is ever deleted. (2) is also trivially true for an arm pair

which is deleted before any other h_2-loop or arm pair. For an arm pair that is deleted subsequently, let P_1 be the beginning of its left arm and P_2 be the end of its right arm, with surface configurations γ_1, γ_2, respectively. By the definition of an arm pair, P_1 and P_2 delimit a peak in the current residual computation; the first and last operations in this peak are unmarked because they are inside an arm pair that is being deleted. By Lemma 2.4, P_1^{R}, P_2^{L} are candidates for the choice of a pair of points for (γ_1, γ_2) in the marking phase. Since they were not chosen, it must be the case that there was another pair of points (Q_1, Q_2) chosen for (γ_1, γ_2). By Lemma 2.4, Q_1 and Q_2 are the desired points for the arm pair. ∎

From Lemmas 2.2 and 2.3 we obtain bounds on the net stack changes and the lengths of the h_2-loops and arm pairs that are deleted:

LEMMA 2.6. *Every deleted h_2-loop has length less than $L(h_2)$. Every deleted arm causes a net change of less than h_1 in stack height and has length less than $L(h_2)$.*

Proof. The first assertion is just Lemma 2.2. For the second assertion, note that the initial and final stacks for an arm must differ by less than h_1 in height, as otherwise the same argument as in Lemma 2.3 shows that there is a shorter arm pair nested inside the pair being removed, contrary to the algorithm. It follows that neither arm can have a stack variation exceeding $h_2 = 2h_1$, for otherwise the hypothesis of Lemma 2.3 is satisfied, so that the arm would also contain an arm pair, contradicting minimality. Finally, by the algorithm and Lemma 2.2, each arm must have length less than $L(h_2)$. ∎

Clearly the algorithm must terminate. The following lemma bounds the length of the final residual computation.

LEMMA 2.7. *There is a constant c depending on M alone such that when the deletion algorithm terminates, the final residual computation has length at most $cn^2R(n)\,L(h_2)$.*

Proof. We first establish the following claim: in the final residual computation, any peak containing m marked operations has length at most $(5m-3)\,L(h_2)$ provided $m > 0$. The proof of this claim is by induction on the stack variation of the peak. In the base case, consider any peak with variation at most h_1. The m marked operations divide the peak into $m+1$ parts, each of which has length less than $L(h_2)$ because it does not contain an h_2-loop (Lemma 2.2). Hence the peak has length at most $(m+1)(L(h_2)-1)+m$ which is bounded by $(5m-3)\,L(h_2)$ since $m \geqslant 1$ and $L(h_2) \geqslant 1$.

For the inductive step, consider a peak of variation at most $(i+1)h_1$ for some $i \geqslant 1$. Let the initial and final stack height be H_0. By considering transitions of the stack height between $H_0 + h_1$ and $H_0 + h_1 + 1$, we can decompose the peak into "subpeaks" of stack variation $\leqslant ih_1$ and "valleys" in which the stack height stays between H_0 and $H_0 + h_1$. Suppose u of the subpeaks and v of the valleys contain

marked operations. Further suppose that the numbers of marked operations in them are $m_1, ..., m_u, m'_1, ..., m'_v$, respectively. Then clearly the entire peak contains

$$m = m_1 + \cdots + m_u + m'_1 + \cdots + m'_v$$

marked operations. By induction, the u subpeaks have lengths at most $(5m_1 - 3)\,L(h_2), ..., (5m_u - 3)\,L(h_2)$, respectively. By an argument as in the base case, the v valleys have lengths at most $(5m'_1 - 3)\,L(h_2), ..., (5m'_v - 3)\,L(h_2)$, respectively. Now any subpeak not containing a marked operation has stack variation at most h_1 as otherwise it must contain an arm pair by Lemma 2.3. Hence between two subpeaks/valleys containing marked operations, we have a string of operations which has stack variation at most $2h_1 = h_2$, and so of length at most $L(h_2)$ by Lemma 2.2. There are at most $u + v - 1$ such strings; in addition there might also be a similar string at either end of the peak. Hence the entire peak has length bounded by

$$(5m - 3(u+v))\,L(h_2) + (u+v+1)\,L(h_2) = (5m - (2(u+v) - 1))\,L(h_2).$$

If $u + v \geqslant 2$, then $2(u+v) - 1 \geqslant 3$, and so the above expression is bounded by $(5m - 3)\,L(h_2)$ as desired. We are left with the case $u + v = 1$, which means either $u = 0$ and $v = 1$ or $u = 1$ and $v = 0$. If $u = 0$ and $v = 1$, the proof is similar to that for the base case, using the fact that any subpeak must have stack variation less than h_1 as shown above. If $u = 1$ and $v = 0$, we can obtain an arm pair by the proof of Lemma 2.3, contrary to the termination condition of the algorithm. This completes the proof of the claim. Without loss of generality we can assume that accepting computations are peaks, for example by modifying M to push a special symbol on the stack in its first operation and not pop this symbol until the very last operation. The lemma then follows from the fact that the residual computation is a peak containing $O(n^2 R(n))$ marked operations (Lemma 2.1). ∎

Reinsertion

By Lemma 2.5, we can reinsert the deleted h_2-loops at appropriate initially chosen points and the deleted arm pairs at apropriate initially chosen pairs of points, such that the surface configurations are maintained correctly. Furthermore, since h_2-loops have no net effect on the stack, their reinsertion will in fact preserve validity *modulo* the counters. This will be true of the arm pairs also, provided we reinsert them according to the stack principle. That is, at each pair of points, two arm pairs (A_1, A'_1) and (A_2, A'_2) must be in the order

$$A_1 \cdots A_2 \cdots A'_2 \cdots A'_1 \qquad \text{or} \qquad A_2 \cdots A_1 \cdots A'_1 \cdots A'_2.$$

The formal argument amounts to establishing analogs of Lemmas 2.4 and 2.5 for the reinsertion process.

The reinsertion produces a string of operations which is a permutation of the original valid computation. However, any h_2-loop or arm deleted from between two

operations initially marked for CSV changes will be reinserted between the same two operations. It follows that the new string is also a valid computation, because in addition to validity *modulo* the counters, for each counter, the total increment is equal to the total decrement in each reversal, so that the CSV changes do in fact take place at the initially marked operations. The new valid computation is "simpler" in the sense that its loops and arm pairs are concentrated at the points chosen in the marking phase. We now manipulate this valid computation into another one which is even simpler, as follows.

For each surface configuration that occurs, we classify the h_2-loops according to their effects on the k counters. If we take an h_2-loop in a valid computation and replace it by another h_2-loop in the same class, the resulting string is still a valid computation, since both h_2-loops have no net effect on the s, p, σ components and the same net effect on each of the k counters. So at each point initially chosen for a surface configuration, we choose a representative from each class, and replace occurrences of other members of the class by it. Finally, at each point permute the h_2-loops so that occurrences of the same (representative) h_2-loop appear consecutively. This clearly does not destroy the validity of the computation as the changes to counters are only permuted within the same CSV value. Since an h_2-loop has length less than $L(h_2)$, its total increase/decrease to each counter must be between 0 and $L(h_2)-1$, so there can be at most $L(h_2)^k$ distinct classes of h_2-loops at each chosen point. Next we carry out the same replacement and permutation of arm pairs for each pair of surface configurations, taking care that the permutation takes place symmetrically on the two sides. Now each arm has stack variation less than h_1 and length less than $L(h_2)$. Hence there are at most

$$|\Gamma|^{h_1-1} + |\Gamma|^{h_1-2} + \cdots + |\Gamma| < |\Gamma|^{h_1}$$

different possibilities for the net change in stack, and $L(h_2)^k$ different possible changes on the k counters. Consequently, there are at most $|\Gamma|^{h_1} \times L(h_2)^{2k}$ distinct arm pairs at each pair of points.

Transformation to a Linear Diophantine System

We have now arrived at the desired normal form. The h_2-loops are concentrated at a "small" number of points, the number of different h_2-loops that actually occur at each point is "small," and occurrences of the same h_2-loop appear consecutively. Here "small" means being bounded by the functions given above, and similarly for arm pairs. The next step is to generate a class of computation strings from the normal form by allowing the number of occurrences of each distinct h_2-loop at each chosen point and the number of occurrences of each distinct arm pair at each chosen pair of points to vary over the nonnegative integers. The number of variables so introduced is at most

$$cn^2R(n)(L(h_2)^k + |\Gamma|^{h_1} L(h_2)^{2k})$$

for some constant c. For each set of values for these variables, we have a com-

putation that like residual computations is valid *modulo* the counters. The computation will be completely valid if and only if the total increment equals the total decrement for each counter in each reversal. Since the increments and decrements are linear expressions in the variables, this validity condition can be translated into a linear diophantine system of at most $kR(n)$ equations. We add an extra variable and an extra equation to express the total time of the computation in terms of the above variables, and convert the whole system into the matrix form $Ax = b$. Each nonnegative integral solution to this system yields a valid computation whose length is given by the value of the "time-variable" in the solution. The reader is referred to [1, 3, 6] for similar constructions of linear diophantine systems.

Using the fact that $L(h_2) \leqslant n^{c_1 n^2}$ for some constant c_1, we can estimate the size of this system. The matrix A has at most $kR(n)+1$ rows and at most $R(n)n^{c_2 n^2}$ columns, respectively, for some constant c_2; the latter bound is obtained by applying the bound on $L(h_2)$ to the bound on the number of variables given above. The absolute value of each entry in A, being the length of an h_2-loop or an arm pair (for the "time equation") or its effect on a counter (for the other equations), is at most $L(h_2) \leqslant n^{c_1 n^2}$. Finally each entry of b is derived from the length of the final residual computation (for the "time equation") or its effect on a counter in some reversal (for the other equations); the sum of their absolute values, written $|b|$, is at most $R(n)^{c_3 n^2}$ for some constant c_3 by Lemma 2.7 and the bound on $L(h_2)$.

To obtain an upper bound on minimal nonnegative solutions of the system, we observe that our diophantine system is known to have a nonnegative solution, namely the one corresponding to the normal-form accepting computation obtained at the end of the reinsertion phase. By Lemma 1.1, it has a nonnegative solution in which every entry is bounded by $rsM_A^2|b|$, where

$$r \leqslant kR(n)+1, \qquad s \leqslant R(n)n^{c_2 n^2},$$

$$M_A \leqslant 2(kR(n)+1)!(n^{c_1 n^2})^{kR(n)+1} \leqslant (R(n)n^{c_1 n^2})^{c_4 R(n)},$$

and $|b| \leqslant R(n)n^{c_3 n^2}$, c_1, c_2, c_3, and c_4 being constants depending on M alone. Hence an upper bound on each entry of the desired solution is

$$(kR(n)+1) \times R(n)n^{c_2 n^2} \times (R(n)n^{c_1 n^2})^{2c_4 R(n)} \times R(n)n^{c_3 n^2} \leqslant (n^{n^2}R(n))^{cR(n)},$$

where c is a constant depending on M alone. Thus:

LEMMA 2.8. *Let M be a nondeterministic 2-way PDA with k $R(n)$ reversal-bounded counters. Then there is a constant c depending on M alone such that if M accepts an input w of length n, there is an accepting computation of M on w which contains*

(1) $n^{cn^2}R(n)$ *chains of identital cn^2-loops each of length at most n^{cn^2},*

(2) $n^{cn^2}R(n)$ *"chains" of identical arm pairs, each with a net change of at most cn^2 in stack height and of length at most n^{cn^2}, and*

(3) $n^{cn^2}R(n)$ *other operations.*

Furthermore, the length of this computation, and hence the numbers of repetitions in the chains of cn^2-loops and arm pairs, are at most $(n^{n^2}R(n))^{cR(n)}$.

Simulation by an NTM

Finally, we are in a position to simulate M on a nondeterministic multitape TM M_1. M_1 will try to guess a valid computation in normal form satisfying Lemma 2.8. To save time, M_1 will not simulate M step by step all the time. Instead, it watches out for cn^2-loops in the computation of M, and whenever one is found it guesses the number of times the loop is repeated, and updates the status of M's computation for a chain of these many repetitions of the loop "in one fell swoop." Similarly for arms. The status of M's computation consists of the state of its finite control, the position of the input head, the stack content, and the counter values. M_1 maintains M's state in its own finite control and stores each of the other items on a separate worktape. The counters and the input head position are stored in binary. Clearly $\log_2 n$ space suffices for the head position; by Lemma 2.8, $s(n)$ space suffices for the counter tapes, where

$$s(n) = R(n) \log_2(n^{n^2}R(n)).$$

The stack tape contains both individual symbols of Γ and entries of the form (α, r), where $\alpha \in \Gamma^+$, $|\alpha| \leqslant cn^2$, and r is a positive number encoded in binary; (α, r) represents α^r. The individual symbols from Γ are pushed by the operations outside cn^2-loops or arms; these operations are simulated individually by M_1. An entry of the form (α, r) is pushed by a chain of r instances of an arm each effecting a net increase of α on the stack. For the normal-form computation of Lemma 2.8, $s(n)$ space suffices for r.

As long as the simulation has not reached an accepting configuration of M, M_1 can choose to simulate either an individual step of M or a chain of cn^2-loops or arms. To simulate an individual operation of M, M_1 requires up to $s(n)$ time to update the simulated counter values and head position (note that $s(n) \geqslant \log_2 n$). Thus in simulating a normal form accepting computation satisfying Lemma 2.8, M_1 need to spend a total of at most $n^{cn^2}R(n)\,s(n)$ time to simulate these "other operations."

To guess a cn^2-loop or arm, M_1 carries out stepwise simulation of M, except that instead of updating the status of M's computation, M_1 simply records the accumulated effects of the individual steps while making sure that the criteria for a loop or arm are not violated. For example, while guessing a left arm with initial surface configuration (s, p, b, τ), M_1 maintains the state s', position p', partial stack σ' and the accumulated increases $d_1, ..., d_k$ to the counters (each d_i may be negative, zero, or positive). Here σ' is the part of the stack *above* the b in the initial surface configuration. For each individual step, M_1 has to check that $|\sigma'| > 0$, that the counter effects are consistent with τ (for example, if counter i is decreasing but positive, then $d_i \leqslant 0$ and $c_i + d_i > 0$ where c_i is the ith counter content before M_1 starts guessing the left arm). When M_1 finds that $s' = s$, $p' = p$ and the top symbol of σ' is b, it knows that it has found a left arm. At this point, M_1 guesses a

repetition factor r in binary, and updates the status of the simulated computation as follows: the ith counter is increased by rd_i, the multiplication and addition (or subtraction) being performed in binary, and an entry (σ', r) is pushed on top of the stack. Of course M_1 also checks that the counter changes rd_i are consistent with the current CSV. The details for guessing a cn^2-loop or a right arm are similar, except that M_1 will only guess and simulate a chain of right arms if it finds an entry of the form (α, r) at the top of the simulated stack, and it will use r as the repetition factor when it has found a right arm whose net effect on the stack is to pop off the string α.

Now we analyze the time required by M_1 to simulate loops and arms in a normal form accepting computation satisfying Lemma 2.8. When guessing a cn^2-loop or an arm, M_1 takes at most $\log_2(n^{cn^2}) = cn^2 \log_2 n$ time to simulate one step of M; this time bound is derived from the bound on the length in binary of each d_i. Since the loop or arm has length at most n^{cn^2}, the total time is $n^{c'n^2}$ for some constant $c' > c$. A repetition factor need to be at most $(n^{n^2}R(n))^{cR(n)}$, so M_1 takes $s(n)$ time to guess it. The updates then take time at most $(cn^2 \log_2 n)\, s(n)$ (for multiplication of two binary integers of lengths $cn^2 \log_2 n$ and $s(n)$, respectively). Since there are $O(n^{cn^2}R(n))$ chains of loops and arms altogether, the total time is of the order of

$$n^{cn^2}R(n)(n^{c'n^2} + cn^2(\log_2 n)\, s(n)).$$

Combined with the time for simulating individual steps, this yields a bound of

$$n^{cn^2}R(n)\, s(n) + n^{cn^2}R(n)(n^{c'n^2} + cn^2(\log_2 n)\, s(n)) \leqslant n^{cn^2}R(n)^2(\log_2 R(n) + 1)$$

on the total computation time of M_1, for an appropriate constant c depending on M alone. Hence

THEOREM 2.1. *Let M be as in Lemma 2.8. Then $L(M)$ is accepted by an*

$$n^{cn^2}R(n)^2(\log_2 R(n) + 1)$$

time-bounded nondeterministic multitape TM, where c is a constant depending on M alone.

5. ONE-WAY MACHINES

In this section, we study the effect of restricting the input tape to be 1-way. The general construction of Section 2 is modified to show that the simulating TM now takes time polynomial in $n + R(n)$. For $R(n) \equiv 1$, we show that the augmented PDAs accept in linear time and hence are equivalent to 1-way simple multihead PDAs.

Let M be a 1-way nondeterministic PDA augmented with $k\, R(n)$ reversal-bounded counters. The normal form algorithm will be the same as in Section 2,

except for the following simplifications. First of all, a total configuration is now defined to be a vector $(s, a, \sigma, \tau, x_1, ..., x_k)$, with the symbol $a \in \Sigma \cup \{\not{\epsilon}, \$\}$ currently scanned by the input head replacing the actual input head position p in the vector. Stack configurations and surface configurations are modified accordingly.

Next we mark operations that move the input head or change the CSV. There are $O(n + R(n))$ such operations. Then for each pair of surface configurations with the same top stack symbol we choose a pair of points satisfying the same criteria as in Section 2, and for each surface configuration we choose a single point, and mark the adjacent operations. There are $O(R(n))$ surface configurations and $O(R(n)^2)$ pairs of surface configurations with the same top stack symbol, but the proof of Lemma 2.1 can be modified to show that marking occurs for at most $O(R(n))$ of the pairs. Hence altogether at most $O(n + R(n))$ operations may be marked.

We now define $L(h) = (q+3)^{h+1}$, $h_1 = q^2|\Gamma| + 1$, and $h_2 = 2h_1$; note that h_1, and hence also h_2 and $L(h_2)$, are constants depending on M alone. Then we apply the deletion algorithm of Section 2. Lemmas 2.2 to 2.6 hold for the new definitions of $L(h)$, h_1, and h_2, with essentially the same proofs. Note that in the 1-way case, since operations that move the input head are marked, the input symbol component of each type of configuration (surface, stack, or total) is constant throughout a string of unmarked operations. Instead of Lemma 2.7, we have an $O(n + R(n))$ bound on the length of the final residual computation. The proof of this bound differs from the proof of Lemma 2.7 only in the last step: after showing that a final residual computation containing m marked operations has length at most $(5m-3)\,L(h_2)$, we make use of the facts that (1) at most $O(n + R(n))$ operations are marked (see above), and (2) $L(h_2)$ is a constant.

We note in passing that the deletion algorithm in the 1-way case can be used to show that for a 1-way (unaugmented) PDA, there is a constant c such that every accepting computation on an input of length n contains a subsequence (of not necessarily consecutive operations) which is also an accepting computation on the same input and which has length at most cn. This implies that 1-way PDAs accept in linear time, a result proved by Greibach [5] using derivation trees of context-free grammars.

Reinsertion and replacement by representatives are the same as in Section 2. Here loops and arm pairs all have lengths bounded by constants (independent of the length of the input). Also it is possible for a loop to be reinserted at a point with a different input head position (but the same surface configuration) and similarly for arm pairs. Hence in the resulting normal form accepting computation, the representative loops and arm pairs are concentrated at the $O(R(n))$ points chosen for surface configurations and at the $O(R(n))$ pairs of points chosen for pairs of surface configurations with the same top stack symbol. Furthermore, the number of representative loops (resp. arm pairs) at each point (resp. pair of points) is constant. Therefore the resulting diophantine system has $O(R(n))$ variables, and its coefficients (which represent the lengths or effects on counters of loops and arms) are constants depending on M alone. The number of equations is still $O(R(n))$. The sum of the absolute values of the right-hand sides is bounded by twice the length of

the final residual computation and so is $O(n + R(n))$. Hence Lemma 1.1 guarantees that on an accepted input of length n, there is a normal-form accepting computation of length $O(n(2R(n))^{cR(n)})$ for some constant c.

The last step is to simulate the normal-form computations on a nondeterministic TM as in Section 2. The only difference here is that M_1 can store all possible loops and arm pairs in its finite control, so whenever it wants to simulate a chain of repetitions of a loop (or arm), all it has to do is to pick one that is applicable from its finite control, and perform the update with a guessed repetition factor. Since the net effect of a loop or arm to a counter or the stack is finitely bounded, the update to each counter and to the stack tape can be performed in one pass over all worktapes, as the repetition factor is being guessed. Now the length of each tape is bounded by $\log_2(n(2R(n))^{cR(n)})$, which is therefore the time required to simulate either an individual step or a chain of loops or arms. The normal-form computation contains $O(n + R(n))$ individual steps and $O(R(n))$ chains of loops and arms. Hence:

THEOREM 2.1. *Let M be a nondeterministic 1-way PDA with k $R(n)$ reversal-bounded counters. Then $L(M)$ is accepted by a nondeterministic multitape TM with time bound*

$$(n + R(n)) \cdot (R(n) \log_2 R(n) + R(n) + \log_2 n).$$

Finally, we consider the special case in which the counters are allowed to make only 1 reversal each. Ibarra [9] showed that such machines accept languages with effectively constructible semilinear Parikh maps and hence have a decidable emptiness problem. The same result holds for 1-way *simple* multihead PDAs studied in [8]; these are normal 1-way PDAs augmented with 1-way input heads, called *counting heads*, which can only tell whether the symbol they are scanning is an endmarker. The proofs for the two results are similar, and we now show that in fact the two classes of machines are equivalent.

THEOREM 3.2. *The class of languages accepted by 1-way simple multihead PDAs is the same as the class of languages accepted by 1-way single-head PDAs augmented with 1-reversal counters.*

Proof. Clearly a 1-way simple multihead PDA with k counting heads can be simulated by a 1-way single-head PDA with $k + 1$ 1-reversal counters: the simulator guesses the length of the input on the $k + 1$ counters, and then uses k of them to simulate the k counting heads. The last counter is used to verify that the length was guessed correctly. In the other direction, the analysis of this section shows that PDAs with 1-reversal counters accept in linear time. Hence the counters can be made to grow to at most $\lfloor n/2 \rfloor$ on input of length n. Each such counter C can be simulated by two simple heads H_1, H_2 as follows. For every increment of C, H_1 is

moved one square and H_2 is moved two squares to the right. When C switches to decreasing after attaining a size of $x \leqslant \lfloor n/2 \rfloor$, H_1 is x squares to the left of H_2 which in turn is to the left of the right endmarker. Now move H_1 and H_2 simultaneously to the right until H_2 reaches the endmarker. Then H_1 is x squares from the endmarker, and so the decrementing of C is simulated by moving H_1 to the right. ∎

COROLLARY 3.1. *Every deterministic* 1*-way PDA with* 1*-reversal counters can be simulated by a deterministic* 1*-way simple multihead PDA.*

Proof. The second simulation given in the proof of Theorem 3.2 preserves determinism. ∎

Remark. The guess of the input length is essential in the first simulation. As a result, we do not know whether the converse to Corollary 3.1 holds.

4. THE PUSHDOWN STORE IS A COUNTER

This section is devoted to the special case in which the pushdown store of the augmented machine is restricted to be a counter. Thus the machine M under consideration is really a nondeterministic 2-way multicounter machine with one unrestricted counter C_0 and k $R(n)$ reversal-bounded counters $C_1, ..., C_k$. The results of this section show that M can be simulated by a nondeterministic $p(n + R(n))$ time-bounded multitape TM, where p is a polynomial. Hence by the results of [3], for $R(n) \geqslant n$, removing the reversal bound from one counter in an $R(n)$ reversal-bounded multicounter machine leads to very little gain in computational power. Note that for this class of machines, in order to satisfy the requirement that each restricted counter switches from decreasing to increasing only when it is empty, we may have to add an extra counter whose number of reversals is the sum of the numbers of reversals of the original restricted counters.

We first modify the analysis of Section 2 to take advantage of the fact that the pushdown store is now restricted to a single letter alphabet. This modification is necessary for reducing the number of chains of loops and arm pairs in the normal form—a reduction which is crucial for eliminating the n^{cn^2} factor in the results of Section 2.

Throughout this section, we shall use C_0 rather than σ to denote the pushdown store (i.e., the unrestrictted counter), and $|C_0|$ to denote its current size. For consistency with previous sections, we shall still refer to C_0 as the stack. While $C_1, ..., C_k$ are restricted to $R(n)$ reversals apiece, C_0 may independently make many more reversals; furthermore, C_0 is not required to be empty when switching from decreasing to increasing. Total configurations, stack configurations, and surface configurations can be defined as before, except that the "top symbol" component of

a surface configuration will be suppressed, since we shall make use of surface configurations only at points where C_0 is known to be nonempty.

As in Section 2, we begin by marking certain operations in the given $R(n)$ reversal-bounded accepting computation. The first operations to be marked are those which change the CSV. Next consider the peaks of the computation defined by the transition of $|C_0|$ between h_2+1 and h_2+2, where $h_2=2h_1$ and $h_1=(q(n+2))^2+1$. We shall refer to these as the (h_2+1)-*peaks*. Note that $|C_0|$ is h_2+1 at both ends of an (h_2+1)-peak, and at least h_2+2 in between. Then for every triple $\gamma=(s, p, \tau)$ find a point in some (h_2+1)-peak with surface configuration γ, and for every pair of triples $\gamma_1=(s_1, p_1, \tau_1)$, $\gamma_2=(s_2, p_2, \tau_2)$ find a pair of points in some (h_2+1)-peak such that they delimit a peak and have γ_1, γ_2 as their respective surface configurations. Mark the operations adjacent to the chosen points. By Lemma 2.1, at most $O(n^2R(n))$ operations will be marked this way.

Next we modify the definitions of two of the terms we use, as follows:

(1) An h-loop is a string of operations with the same initial and final stack configurations and with stack variation at most h (note that we drop the requirement that the stack height never falls below its initial value);

(2) $L(h)=(h+1)(q(n+2)+1)$.

Note that an h-loop between points P_1 and P_2 can be moved to another point P_3 without destroying the validity of the computation if P_3 has the same surface configuration as P_1, and $|C_0|>h$ at both P_1 and P_3. Lemma 2.2 holds with these new definitions, as shown by the following argument. Take a string of at least $L(h)$ unmarked operations. The proper prefix of length $L(h)-1$ contains $L(h)$ points. Since there are at most $h+1$ different stack heights in a string with stack variation $\leqslant h$, there must be one stack height at which there are at least $L(h)/(h+1)=q(n+1)+1$ points. By the pigeonhole principle, two of these points have the same state and the same input head position. Since the CSV never changes (none of the operations is marked), and a stack which is a counter is completely characterized by its height, the two points have the same stack configurations and hence define an h-loop.

Now we apply the deletion algorithm of Section 2, but only to the (h_2+1)-peaks. As before, the deletions preserve validity *modulo* the counters $C_1, ..., C_k$. Note that because h_2-loops have zero net effect on C_0, the operations remaining in an (h_2+1)-peak after the deletion of an h_2-loop still form an (h_2+1)-peak in the residual computation. Similarly for arm pairs. Lemmas 2.3 to 2.6 are still valid with the new definitions of h-loops, h_1, h_2, and $L(h)$, and with the understanding that $|\Gamma|=1$ for a stack which is a counter. From Lemma 2.6, we conclude that each deleted h_2-loop or arm has length $O(n^3)$.

When the deletion algorithm terminates, a marked (h_2+1)-peak containing m marked operations will have length at most $(5m-3)L(h_2)<cmn^3$ for some constant c, by the same argument as in Lemma 2.7. Also an unmarked (h_2+1)-

peak has stack variation at most h_1, in order that it does not contain an arm pair (Lemma 2.3). Thus all points outside marked (h_2+1)-peaks must have $|C_0| \leqslant h_3$, where $h_3 = h_1 + h_2 + 1 = 3h_1 + 1$. Note that h_3 is $O(n^2)$.

Before we reinsert the deleted h_2-loops and arm pairs into the final residual computation, we impose some order on the strings of operations outside the marked (h_2+1)-peaks. Partition each such string by the marked operations in it. These must be operations that are marked because of CSV changes. We end up with a grand total of $O(n^2R(n))$ strings of operations which contain no marked operation and hence have constant CSV, and which have $|C_0| \leqslant h_3$ throughout. Each of these strings will now be converted into chains of simple h_3-loops as far as possible by permutations and replacements, so that the chains are concentrated at at most $q(n+2)(h_3+1)$ points, one for each different stack configuration. Note that moving an h_3-loop to a different point with the same stack configuration does not destroy validity *modulo* the counters. Now a simple h_3-loop has length less than $q(n+2)(h_3+1)$. This means that the simple h_3-loop can be classified into $(q(n+2)(h_3+1))^k$ distinct classes according to their effects on $C_1, ..., C_k$. As a result, each of the unmarked strings of operations is transformed into at most $(q(n+2)(h_3+1))^{k+1}$ chains of identical simple h_3-loops, at at most $q(n+2)(h_3+1)$ points. The operations outside these chains are partitioned into at most $q(n+2)(h_3+1)+1$ segments, each of length at most $q(n+2)(h_3+1)-1$ in order not to contain an h_3-loop; the total number of such operations is therefore at most $(q(n+2)(h_3+1))^2-1$. Thus outside the marked (h_2+1)-peaks there are $O(n^{3k+5}R(n))$ chains of identical h_3-loops each of length $O(n^3)$, and $O(n^8R(n))$ other operations.

Finally the deleted h_2-loops and arm pairs are reinserted into the (h_2+1)-peaks, again after suitable replacement by representatives, in such a way that identical h_2-loops or arm pairs are consecutive. The insertion of arm pairs clearly preserves the validity of the computation *modulo* $C_1, ..., C_k$, since the increments of C_0 precede the corresponding decrements. The same is true of h_2-loops, since they have zero net effect on C_0, have stack variation bounded by h_2, and are only inserted at points where $|C_0| > h_2$. It is easy to see that for each surface configuration, there are $O(n^{3k})$ distinct h_2-loops, and for each pair of surface configurations, there are $O(n^{6k+3})$ distinct arm pairs ($O(n^{3k})$ different possible effects on the k counters for each arm, and $O(n^3)$ different possible gains in stack height by the left arm), for a total of $O(n^{3k+1}R(n))$ chains of h_2-loops and $O(n^{6k+5}R(n))$ chains of arm pairs.

The corresponding linear diophantine system has a total of $O(n^{6k+5}R(n))$ variables representing the numbers of occurrences of h_2-loops, arm pairs and h_3-loops, and $O(R(n))$ equations to enforce the integrity of $C_1, ..., C_k$ in each reversal. The coefficients have size $O(n^3)$, and the sum of the absolute values of the right-hand sides is $O(n^8R(n))$, being bounded by the number of operations that are outside h_2-loops, arm pairs and h_3-loops. By Lemma 1.1, we obtain

LEMMA 4.1. *Let M be a nondeterministic 2-way machine with one unrestricted counter and k $R(n)$ reversal-bounded counters. Then there is a constant c such that M*

accepts in $(nR(n))^{cR(n)}$ *time. Furthermore, these time-bounded accepting computations conform to the normal form described above.*

Finally, an obvious modification of the simulation in Theorem 2.1 yields

THEOREM 4.1. *Let M be as in Lemma* 4.1. *Then* $L(M)$ *is accepted by a nondeterministic multitape TM with time bound*

$$n^{6k+5}R(n)(\log_2 n)(n^3 + R(n)\log_2(nR(n))).$$

5. CONCLUDING REMARKS

The main result of this paper is the extension of the polynomial relation between TM time and counter machine reversal to counter machines augmented with an unrestricted pushdown store. It can be viewed as further evidence of the limited power of a *single* pushdown store. Several questions remain open:

(1) Do Theorems 2.1, 3.1, and 4.1 hold for deterministic machines? We surmise that they do, but have not worked out the details of the deterministic simulation (the normal form results hold for the deterministic machines, but the unique computation on an accepted input may be decomposed into normal form in many ways).

(2) How good is the NTM simulation time in Theorem 2.1? For example, can it be lowered to a polynomial in $n + R(n)$ for 2-way PDAs augmented with $R(n)$ reversal-bounded counters? In particular, note that 2-way multicounter machines with constant reversal bounds on the counters accept only languages in NSPACE($\log n$) and hence in **P** [6], but the NTM simulation time of Theorem 2.1 for the same machines augmented with a pushdown store is n^{cn^2}, where c is a machine-dependent constant.

(3) Does the converse to Corollary 3.1 hold, i.e., can every deterministic 1-way simple multihead PDA be simulated by a deterministic 1-way PDA augmented with 1-reversal counters?

ACKNOWLEDGMENTS

The author is grateful to the referees for their suggestions.

REFERENCES

1. B. S. BAKER AND R. V. BOOK, Reversal-bounded multipushdown machines, *J. Comput. System Sci.* **8** (1974), 315–332.
2. I. BOROSH AND L. B. TREYBIG, Bounds on positive integral solutions of linear Diophantine equations, *Proc. Amer. Math. Soc.* **55** (1976), 299–304.

3. T.-H. CHAN, Reversal complexity of counter machines, *in* "Proceedings, 13th Annu. ACM Symp. on Theory of Computing, Milwaukee, Wisconsin, May 1981," pp. 146–157.
4. S. A. COOK, Characterizations of pushdown machines in terms of time bounded computers, *J. Assoc. Comput. Mach.* **18** (1971), 4–18.
5. S. A. GREIBACH, A note on the recognition of one counter languages, *RAIRO* **R-2 9** (1975), 5–12.
6. E. M. GURARI AND O. H. IBARRA, The complexity of decision problems for finite-turn multicounter machines, *J. Comput. System Sci.* **22** (1981), 220–229.
7. J. E. HOPCROFT AND J. D. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.
8. O. H. IBARRA, A note on semilinear sets and bounded-reversal multihead pushdown automata, *Inform. Process. Lett.* **3** (1974), 25–28.
9. O. H. IBARRA, Reversal-bounded multicounter machines and their decision problem, *J. Assoc. Comput. Mach.* **25** (1978), 116–133.
10. M. MINSKY, Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines, *Ann. of Math.* **74** (1961), 437–455.