Graded Self-Reducibility

Mitsunori Ogihara[*] Kenneth W. Regan[†] Seinosuke Toda[‡]
University of Rochester University at Buffalo Nihon University

February 12, 2007

Abstract

We introduce two special kinds of self-reducible sets that we call *time-graded* and *space-graded* languages. Attaching time and space bounds to them yields a uniform, quantitative definition of resource-bounded self-reducibility that applies to all languages. We apply this to study the relationship between NSPACE$[s(n)]$ and DSPACE$[s(n)^2]$, emphasizing the case $s(n) = O(\log n)$. We show that the class of logspace-graded languages, which is contained in DSPACE$[\log^2 n]$, contains not only NL and uniform NC^2, but also a class of Aux-PDA languages that is not known to be a subclass of P. We also prove that many-one space-graded classes are many-one equivalent to the special case in which the self-reduction makes one query to a string of strictly shorter length. For this latter idea of "instance contraction," we note that some NL-complete problems admit n-to-$n/2$ contraction in logspace, and prove that some NC^1-complete problems are n-to-$n/2$ contractible by finite automata that involve only parity counting.

1 Time and Space Graded Languages

A language Z is *autoreducible* if there is an oracle Turing machine M such that for all x, $M^Z(x)$ decides whether $x \in Z$ without querying x itself. Placing time and/or space bounds (etc.) on M, and perhaps imposing standard *truth-table*, *disjunctive*, *conjunctive*, *many-one* (etc.) restrictions on the way M treats its queries, together define kinds of autoreductions. A *self-reduction* is the special case where all queries y made by $M^Z(x)$ are below x in some partial order. Complexity-bounded self-reductions are commonly defined by placing conditions on the partial order as well as on M. These two concepts—mostly the latter, recently the former—have had wide importance and application in complexity theory.

[*]Supported by a US-Japan Co-operative Research grant from the National Science Foundation, NSF Grant INT-9726724. Also partly supported by NSF Grants CCR-9701911 and CCR-9725021.

[†]Also partly supported by NSF grant INT-9726724. Corresponding author—contact: regan@cse.buffalo.edu

[‡]Also partly supported by NSF grant INT-9726724

In this paper we study the idea of bringing the notion of "below," the bounds on the partial order, and even the complexity bounds *inside* the language Z rather than outside. We present the new definitions before motivating them below.

Definition 1.1. A language $Z \subseteq \Sigma^* \times \mathbf{N}$ is *time-graded* if there is a Turing machine M such that for all $(x, r) \in \Sigma^* \times \mathbf{N}$.

(a) M^Z accepts (x, r) iff $(x, r) \in Z$,

(b) $M^Z(x, r)$ runs for at most r steps, and

(c) any queries (x', r') made by $M^Z(x, r)$ have $r' < r$.

Z is *space-graded* if in (b), $M^Z(x, r)$ runs within space r. The modifier "many-one" may be added if M makes at most one query and accepts iff the answer is "yes"; "truth-table" if M is a non-adaptive query machine; "btt" for a bounded truth-table reduction; "nondeterministic" if M can be nondeterministic; and so on. The machine M itself is called a *graded solver*, with modifiers as appropriate.

Here r is written in binary notation. We adopt the fairly-standard conventions that oracle queries are *erased* immediately after being submitted, and that queries do not count against the space bound (but cf. [HJ97], and also [RST84]).

Intuitively, we regard r as a "grading" of the difficulty of the instance x. The above definition does not make r a function of x. We can meet this condition by replacing x by $x\#r$, but (b) below has the same functional effect in a more useful manner. Now we quantify the steepness of the descent implied by (c) above.

Definition 1.2.

(a) A *stepdown function* for a graded solver M is a function $h : \mathbf{N} \to \mathbf{N}$ such that for all x and r, $M(x, r)$ makes only queries (x', r') with $r' < h(r)$. The same term is also applied to $L(M)$.

(b) A language A is DTIME$[g(n)]$ *self-reducible with stepdown* $h(r)$ if the language $A' = \{ (x, g(|x|)) : x \in A \}$ is time-graded with stepdown $h(r)$. Here $g(n)$ is presumed to be time-constructible.

(c) A is DTIME$[g(n)]$ *gradable* with stepdown $h(r)$ if there is a time-graded language A' with stepdown $h(r)$ such that $A = \{ x : (x, g(|x|)) \in A' \}$.

DSPACE$[g(n)]$ self-reducible and gradable languages are defined analogously, and modifiers "many-one," "tt-," "btt-" may apply as above.

If no stepdown is mentioned, stepdown $h(r) = r - 1$ is assumed, as in Definition 1.1(c). In all cases, M is entitled to make queries (x', r') with $|x'| > |x|$, but the stepdown imposes a bound on the height of the implicit recursion.

Note that the initial $g(n)$ in (b) and (c) becomes a bound on the time for *all* individual levels of the recursion. The natural alternative would be to stipulate that $M(x', r')$ may run in time $t(|x'|)$ or $t(|x'| + |r'|)$ for all queries. However, this would require time bounds to be mentioned in the definition of a time-graded language itself, and more seriously, would enable M to "cheat" by making $|x'|$ progressively greater than $|x|$. Similar remarks hold for space. Our definition avoids this problem cleanly.

Now we can essentially recover the original "OK partial order" definition of polynomial-time self-reducibility [MP79, Ko83] of a language A by saying that A is poly-time self-reducible with stepdown r-to-$(r-1)$. There are some technical distinctions between conditions on the partial order (cf. [BvHT93]) and what we have, but we do not know any case for which they matter. The "lexicographical word-decreasing query" (*wdq*) definition of Balcázar [Bal90], however, does not fall into our framework, because it permits the implicit recursion to have exponential height. All P-self-reducible langauges as above belong to PSPACE, whereas [Bal90] shows that some E-complete (hence also EXP-complete) languages meet the (*wdq*) condition.

The *intent* of our definition is to model self-reducibility structures via time- and space-graded languages independent of a particular complexity bound. The targeting of our questions to properties of especially the *space* measure *itself* comes through in Section 3. The question of what g and h functions can be supplied then becomes one of asking,

How self-reducible is a given language A?

There is always the trivial upper bound $g(x) = t(|x|)$ where $A \in \mathrm{DTIME}[t]$, via an M that makes no queries, so the question always has *some* answer. But here we repeat Balcázar's observation that even assuming PSPACE $\neq$ P or related hypotheses, no one has found a language in PSPACE that is not polynomial-time self-reducible. The best candidates we know, drawing on the main theorem of [BvHT93], are P-selective sets such as various "left cuts" that are not known to be in P. However, their non-membership in P currently requires much stronger hypotheses than P $\neq$ PSPACE or P $\neq$ NP.

Our question seems to be fairly wide-open, even amid all the attention that self-reducibility has received. A related question, "do there exist languages $A \in \mathrm{DTIME}[t]$ that are not $t'(n)$-self reducible (with any stepdown) for $t' = o(t)$?," seems to have been answered only in the many-one self-reduction case, by taking A to be a language for which all many-one reductions from A are 1-1 almost everywhere (cf. the ascription to A. Meyer in [BH77] and also [BS85] and "incompressible" in [Lut97]). Now we motivate our particular definitions with some examples.

Example 1.1. *SAT:* The standard self-reduction for *SAT* is a DLIN-2dtt self reduction. This is because given a formula ϕ in n variables, the two disjunctive queries $\phi[x_n := 0]$ and $\phi[x_n := 1]$ can be computed in $O(|\phi|)$ time by a Turing machine (in one pass over ϕ, in fact). A linear $g(|\phi|)$ can be used for both the initial time bound and the recursion height, with stepdown r-to-$(r-1)$.

Note the technical quirk that one can't take $g(n) = n$ and keep the stepdown in synch with the number of free variables. One simple reason is that the actual bit length $|\phi|$ of ϕ is $\Theta(n \log n)$, not $\Theta(n)$. One can emulate the "natural" stepdown by defining $h(r) = r - |\phi|/kn$ instead, where $g(n) = kn$, and then the recursion has height n. Our choice of definitions tries to be most precise for complexity bounds rather than for features of individual problems.

Example 1.2. *Directed s-t connectivity: Savitch's Theorem* [Sav70] and its proof amount to a logspace 2-tt self-reduction for the directed s-t connectivity problem. However, here we observe that a normal form of this problem admits a logspace many-one self reduction, one in which the stepdown is accompanied by a halving of the instance size. Consider pairs (G, r) where G is a leveled directed n-node graph with distinguished source node s and sink node t, with all sources in level 0 and all sinks in some level d, and with the same number of nodes in each level.

Our graded solver $M(G, r)$ attempts to carry out the following process within r worktape cells. If $d = 1$, then M determines directly whether there is a directed edge from s to t. Else, M goes about computing a leveled graph G' with $\lceil (d + 1)/2 \rceil$ levels such that G' has a path from s to t if and only if G does. Namely, one can determine for each node in G whether it is in an odd or even level by following any path to a sink, and then can "boot out" all the odd levels between 0 and d by squaring the adjacency matrix and deleting rows and columns for odd-level vertices. Then M submits the query $(G', k - 1)$, and accepts iff the answer is yes. Note that G' has roughly half as many nodes as G does, and that under some natural encoding scheme, $|G'| \leq |G|/2$. On the other hand, if k cells is not enough space to carry this out, then M aborts and rejects.

There is a function $s_0(n) = O(\log n)$ that bounds the work space needed for an ordinary TM to compute G' from G, or to solve the instance in the $d = 1$ case. If we start M with an allowance of $k = s(n) = s_0(n) + \log(n)$, then M will always have enough space to avoid aborting its computation in any recursive call, and will accept (G, k) iff G has a path from s to t. Hence the s-t path problem for graphs G that are suitably normalized as above, which remains NL-complete, is many-one self-reducible in logspace.

In this case, we have a log-space computable procedure that, while not quite able to solve an instance x, is able to map x to an equivalent instance x' of shorter length—indeed, $|x'| \leq |x|/2$. (This assumes a natural encoding scheme for the graphs.) Thus we say that the s-t connectivity problem—at least in the above form—allows "instance contraction by a factor of 2 in log space." It is easy to see that any constant factor size reduction bigger than 2 can be achieved in log-space by a finite iteration of this process.

For contrast, we note that the logspace self-reducibility of Balcázar [Bal90], in which one may change the final $O(\log n)$ bits of the input string x to make *wdq* queries, gives a completely different self-reduction for this problem. On input $G\#s\#t$, his reduction queries the predecessors of t. Without loss of generality there are at most two of them, so this is a 2tt self-reduction. However, as remarked for *wdq* above the linear height of the recursion is more than our scheme allows for being log-graded. Our scheme captures a stronger self-reducibility feature of this problem, one reflected also in the last motivating example.

Example 1.3. *Undirected s-t connectivity:* Nisan, Szemerédi, and Wigderson [NSW92] show that the *undirected s-t* connectivity problem has log-space instance contraction from n to $2^{\sqrt{\log n}}$, and this is the basis of their $(\log n)^{3/2}$ space algorithm for it. This becomes stepdown r-to-$\sqrt{r}$ in our framework.

Studying problems that admit this kind of instance contraction was the original motivation for this work, and is revisited in later sections.

2 Classes of Graded Languages

Now we investigate some particular cases of our definitions, and also explain the difference between (b) and (c) in Definition 1.2

Definition 2.1. A language A belongs to the class LG, for *logspace-graded*, if A is DSPACE$[O(\log n)]$-graded, per Definition 1.2(c).

Expanded, $A \in$ LG iff some space-graded solver M decides whether $x \in A$ by starting itself on input $(x, s(|x|))$, where $s(n)$ is $O(\log n)$ and space-constructible. Note that this definition defines a log-space many-one reduction from A to $L(M)$, via $f(x) = (x, s(|x|))$. This is not a "general" log-space many-one reduction, since the "x" part does not change. The following closure property may seem surprising at first.

Proposition 2.1 *The class* LG *is closed downward under logspace Turing reducibility.*

Proof. Suppose $B \leq_T^{\log} A$, where the conventions defining $\leq_T^{\log}$ are those following Definition 1.1, and where A, M, $s(n)$, and $Z =_{\text{def}} L(M)$ are as above. (We do not need attention to the equivalence of space-bounded Turing and tt-reducibility [LL76], and show a mechanism that works for time as well.) Let Q be the oracle TM that computes the Turing reduction. Then there is a $c > 0$ such that $c \log n$ is a bound on the space used by Q and n^c is a bound on the length of queries that Q can make on inputs x of length n. Also take $d > 0$ such that $s(n)$ is computed in $d \log n$ space. We need to find a space-graded language Z' of which B is an $O(\log n)$ projection. The basic trick is to define Z' to have the form

$$Z' = \{\, (x, 2k) : (x, k) \in Z \,\} \cup \{\, (x, 2k+1) : \ldots \}.$$

We complete the "$\ldots$" by describing a space-graded solver M' for Z'. On inputs $(x, 2k)$, M' simulates $M(x, k)$. On inputs $(x, 2k+1)$, M' attempts to simulate $Q^{(\cdot)}(x)$ within the allotted space $2k+1$, except that queries y made by Q are transformed to $y' = (y, s(|y|))$. If and when this space becomes insufficient, M' "aborts" and rejects. Thus M' is a space-graded solver.

Now with $r(n) = 2\max\{\, s(n^c), (c+d)\log n \,\} + 1$, we claim that B is the $r(n)$-projection of $Z' =_{\text{def}} L(M')$. This $r(n)$ is high enough for the top-level computation $M'(x, r(|x|))$ to finish without aborting. All queries y' to Z' are answered the same way y would be answered by A, so M' accepts $(x, r(|x|)) \iff Q^A$ accepts $x \iff x \in B$. $\square$

Insofar as self-reducibility is intuitively the kind of structural property of individual languages that gets "fuzzed up" by reductions, this makes the distinction between (b) and (c) in Definition 1.2 important. However, the complexity levels of these languages are the same up to reduction equivalence. An inspection of the above proof shows that the only particular property of logarithmic space (as defined for oracle TMs) that it uses is the following.

Definition 2.2. A family R of resource bounds defined for OTMs is "closed under query composition" if for all $r \in R$, the function $r'(n) = r(n')$ also belongs to R, where n' is the maximum possible length of a query that an $r(n)$-bounded OTM can make on an input of length n.

Second, if M and Q both satisfy the same one of the various "qualitative" restrictions on OTMs noted in the last section—including (i) truth-table, (ii) dtt, (iii) ctt, (iv) btt, and (v) many-one—then their "composition" into M' as defined in the proof obeys the same restriction. Moreover, Z' reduces to Z by the same type (i)–(v) of reduction, while Z many-one reduces to Z'. These observations suffice to prove the following general statement.

Theorem 2.2 *For any family T of time bounds that is closed under query composition, including linear and polynomial time, the class of* DTIME$[T]$ *Turing-graded [resp. tt-graded, ctt-graded, dtt-graded, btt-graded, many-one graded] languages is closed downward under* DTIME$[T]$ *Turing [resp. tt-, dtt-, ctt-, btt-, many-one] reductions. Moreover, the time-graded languages witnessing the closure are all equivalent under the respective kind of reduction.*

The analogous statements hold for space complexity.

Now we observe that the time- and space-graded classes have complete sets for the corresponding kind of reductions. Again we illustrate in the concrete case of LG and leave other cases to the reader.

Proposition 2.3 *The class* LG *has many-one complete sets.*

Proof Sketch. The basic idea is that every query machine M can be forced to be a graded solver by making a call $M(x, r)$ abort and reject if it cannot complete its mission within space r. Now define $U = \{M\#x\#0^{n^k} : M$ is a space-graded solver and M accepts $(x, k \log n)\}$. Then universal simulation, always padding queries (x, r) made by M to $(x\#0^{2^r}, r))$, places U into LG, and Definition 1.2(c) amounts to a logspace many-one reduction of any $A \in$ LG to U. (Note that if LG is replaced by many-one LG, then U belongs to many-one LG, and likewise for bounded-tt LG and so on, since the qualitative restrictions on M can be made syntactic. Whether any of these classes have *natural* complete sets is another matter.) $\square$

Thus the classes of languages defined by Definition 1.2(c) are quite robust. It is therefore natural to ask what relationship these classes have to standard time and space complexity classes. We give a definitive answer for time complexity in the many-one case, so as to motivate our central question for space complexity in the next section. DTISP$[t(n), s(n)]$ denotes the class of languages accepted by TMs that run simultaneously in time $t(n)$ and space $s(n)$.

Proposition 2.4 *For any time-constructible time bound $t(n)$, a language A is* DTIME$[t(n)]$*-many-one-graded if and only if A belongs to* DTISP$[t(n)^2, t(n)]$.

Proof. First suppose that A is DLIN-many-one-graded. To decide whether a given x belongs to A, our time-graded solver M starts out on input $(x, t'(|x|))$, where $t' = O(t)$. Since M has at most $t'(n)$ steps at each stage to write its single query, since the space used for queries can be recycled when unrolling the single-branch recursion, and since the recursion has height at most $t'(n)$, the whole recursion can be simulated in $t'(n)^2$ time and $t'(n)$ space.

Conversely, let T be an ordinary TM that accepts a language A in $Ct(n)^2$ time and $Ct(n)$ space. Now we define a time-graded solver M to recognize the L we need. On inputs (x, k) where k is odd, M computes the initial ID $I_0(x)$ of T on input x and queries $(I_0(x), k-1)$. If k is even, then M interprets its "x" part as an ID I. M then simulates T beginning at ID I for $k/2$ steps. If T halts within that time, M gives the same answer that T gave. Otherwise, M uses the other $k/2$ steps it has to copy the resulting ID J together with $k-2$ to its query tape as the next query. If M cannot do this in time, its computation aborts and rejects.

To arrange that $x \in A \iff (x, t'(|x|)) \in L(M)$, we need only fix t' to be a large enough multiple of t so that the above recursion reaches a halting ID of $T(x)$ without aborting. We claim by inspection that $t'(n) = 2(Ct(n) + t(n)) + 1$ is good enough. A key point here is that the IDs themselves do not become bigger than the initial bound $t'(n)$ on the time for each stage of M, as might happen if we merely specify that $A \in \text{DTIME}[t(n)^2]$. $\square$

This raises the question of whether any interesting collapses happen if, say, DTIME$[n^2]$ = DTISP$[n^2, n]$. The text [WW86] devotes an entire chapter (26.) to time-space problems like this one, but has no mention of whether this implies P = DTISP$[n^{O(1)}, n]$ or even P $\subset$ DSPACE$[n]$. This is plausible on padding-and-translation grounds, but we have been unable to prove it. The obstacle to applying the above proof to (say) an n^2-padded version of a language in DTIME$[n^4]$ is that the intermediate IDs, in recursion levels below the first query, may have size n^2—without any padding.

3 Results for Space-Graded Languages

The inclusion for space analogous to the inclusion for time in the last section is the easy observation that every L-graded language belongs to DSPACE$[\log^2 n]$, by unrolling the recursion. Whether the converse inclusion holds is our central question.

Problem. *Is every language in* DSPACE$[O(\log^2 n)]$ L-*graded?*

Put another way, is DSPACE$[O(\log^2 n)]$ = LG? By padding and translation, this really asks whether every DSPACE$[s(n)^2]$ computation can be given a special structured form, broken up into hierarchical DSPACE$[s(n)]$ segments. Now we try to locate LG, in its Turing down through many-one variants, in the interval between NL and DSPACE$[\log^2 n]$.

Proposition 3.1 *All languages that have* DSPACE$[s(n)]$-*uniform circuits of depth* $s(n)^2$ *and width* $2^{O(s(n))}$ *are* DSPACE$[s(n)]$ *many-one graded. In particular,* NC$^2 \subseteq$ LG.

Proof Sketch. The simple idea is to divide the circuit into $s(n)$ many segments, each of depth $s(n)$. The technical details of encoding depth-$s(n)$ circuit-value problems and building the $s(n)$ space-graded solver are analogous to those in Proposition 2.4 and omitted in this version. One actually obtains that NC^2 equals the class of languages that are NC^1 many-one graded, for DLOGTIME uniformity. $\square$

Auxiliary-pushdown automata (Aux-PDAs) were defined by Sudborough [Sud78] to include a stack that is not subject to the space bound imposed on a Turing machine's worktapes. One may also impose a bound on the height of the stack. The standard proof of

Savitch's theorem gives an Aux-PDA with $O(\log^2 n)$ stack height, with (other) worktapes restricted to log space. Savitch's Aux-PDA also runs in polynomial time (and the languages accepted by polynomial-time log-space Aux-PDAs are exactly those $\leq_m^{log}$-reducible to DCFLs [Sud78]), but it open whether some such Aux-PDAs accept languages of time complexity no less than $2^{\log^2 n}$. The proof of the next theorem is in the Appendix.

Theorem 3.2 *Every language accepted by an deterministic auxiliary-tape pushdown automaton with $O(\log n)$ space and with $O(\log^2 n)$ stack height belongs to* LG.

Now we discuss the issue of a converse to this theorem. Let $A \in$ LG via a space-graded solver M such that on input x to A, M starts with (x, s), $s = k \log |x|$. Let (y_1, s') be the first query made by M, and let w be the worktape contents of M at the time the query is submitted. Then our simulating AUX-PDA D can push $[w, 1, s']$ onto its stack, and begin simulating $M(y_1, s')$ in a depth-first manner. So let us look ahead to suppose we are simulating $M(y_{r_d}, s^{(d)})$ at some depth d, where y_{r_d} is the r_dth query made by $M(y_{r_{d-1}}, s^{(d-1)})$. The top of stack is $[w_r, r, s^{(d)}]$ for some string w of length at most $s^{(d-1)}$. The queries z_r themselves are too long to place on the stack. The problem then is how to *read* a requested bit j of the current z_r. One needs to (re-)generate bit j of z_r from the point of the level-$(d-1)$ simulation at which the query z_r was generated. The current worktape contents w_r from that simulation can be recovered from the stack, but the string $y_{r_{d-1}}$ it was running on cannot be retrieved so easily. One needs to look up a bit j' of that string, and so on down the stack. However, popping the stack destroys information that seems to be needed elsewhere. The upshot is that seemingly one needs to obtain random access either to a sequence $(j_1, j_2, \ldots, j_d)$ of bit requests at respective depths, or—with the intent of re-generating everything from the beginning without disturbing the current stack—to the path $(r_1, r_2, \ldots, r_d)$ that indexes the current query in the current simulation. Either sequence has size $O(\log^2 |x|)$, which is more than the space allowance s that bounds the initial and all subsequent stages.

However, if the sequence $(r_1, r_2, \ldots, r_d)$ can be written down in $O(\log n)$ space, then it can be passed on the stack and saved on a worktape, and this leads to:

Theorem 3.3 *Every* L*-btt graded language is acceptable by an Aux-PDA in log space with $O(\log^2 n)$ stack height.* □

To summarize the results of this section, we have defined a class LG that captures those problems with $\log^2 n$ space computations that can be hierarchically organized in blocks of log-space routines. This class can simulate automata that organize these blocks explicitly on a linear stack. However, the latter automata currently seem unable to pass the address information implicitly used by our more-general scheme. A symptom of this is that the equivalence between logspace Turing and tt-reductions [LL76], which works for any one level of this recursion, seems not to survive across levels. However, our more-general scheme still does not seem to capture all $\log^2 n$ space computations. This stands in contrast to the simple and natural way that time-bounded computations and (NC^2) circuit computations are segmented.

We have, however, yet been unable to obtain any radical "collapse" consequence of LG being equal either to DSPACE[$\log^2 n$] or to the Aux-PDA class. We offer these results as promoting a more in-depth analysis of space-bounded computations. Now we turn to direct study of self-reductions in which the lengths of queries decrease rapidly.

4 Instance Contraction

Our general framework does not make it easy to *force* a constant-factor reduction in the length of queries, so we define this "CFR" notion separately.

Definition 4.1. For any class $\mathcal{Q}$ of query machines, CFR($\mathcal{Q}$) denotes the class of languages A such that for some Q in $\mathcal{Q}$ and $c > 1$, $L(Q^A) = A$, and for all x, $Q^A(x)$ queries only strings of length at most $|x|/c$.

The logspace many-one case yields the class of languages A such that for some logspace-computable function f and all x with $|x| \geq 2$, $|f(x)| \leq |x|/2$, and $x \in A \iff f(x) \in A \vee f(x) = 1$. We call this class simply CFR(log). If f is computable by a finite automaton, we call it CFA.

The s-t connectivity example in Section 1 belongs to CFR(log), and undirected s-t connectivity admits a stronger length contraction from n to $2^{\sqrt{\log n}}$ in logspace [NSW92]. The second author once hoped that CFR(log) would have some canonical association to NL, but it turns out to be many-one equivalent to log-space self-reducibility as follows.

Proposition 4.1 *For all types X of Turing, tt-, btt-, ctt-, dtt-, or many-one query machines, every logspace X-self-reducible language A is logspace many-one equivalent to a language in* CFR($\log X$).

Proof. Given $g(n) = k \log n$ and a graded solver M for $A' = \{\,(x, g(n)) : x \in A\,\}$ (here $n = |x|$), define $U_A = \{\, x\#0^{n^{r+1}} : (x,r) \in A'\,\}$. The reduction from A to U_A takes x to $x\#0^{n^{k+1}}$, while that from U_A to A just checks that the padding has the right length compared to $g(|x|)$. The extra padding, compared to the definition of U in Proposition 2.3, ensures that enough padding remains so that the space-graded solver simulating M always halves its query lengths. $\square$

Note that the equivalence is to true logspace self-reducibility. We do not know whether it holds if A merely belongs to LG, or equivalently, whether CFR(log) (for any "X") is closed downward under $\leq_m^{log}$. However, LG has the same many-one completeness level as CFR(log), so questions of containment with regard to other classes closed under these reductions are the same for our two classes. But one latent structural difference is that we know how to diagonalize for the many-one CFR case. Even a little more space $s_2(n) > s_1(n)$ suffices to recognize a language that allows no contraction in $s_1(n)$ space at all.

Proposition 4.2 *Let $s_1(n) \geq \log n$, let s_2 be a fully space-constructible function such that $s_1(n) = o(s_2(n))$. Then there is a language $L \in$* DSPACE$[s_2(n)]$ *such that L is not even n-to-(n-1) contractible in space $s_1(n)$.*

Proof Sketch. The proof uses standard diagonalization techniques; the language L can be made arbitrarily sparse. $\square$

Instance contraction in classes smaller than logspace promises more information, however. It follows from remarks in the last section that the class of languages for which some NC^1-computable function gives n-to-$n/2$ instance contraction is just NC^2. We look lower down at languages contractible by finite automata, formally the *sub-sequential machine* (SSM) model of [Cho77, RS91].

Example 4.1. The language D_1 of balanced-parenthesis strings is contracted by the SSM S described as follows: If the first bit of x is ')', S translates x to $)^{|x|/2}$. This has the same effect as if S rejected right away. If the first bit is '(', S erases it and takes succeeding bits in pairs, translating $((\mapsto ($, $)) \mapsto)$, $() \mapsto \lambda$, $)(\mapsto \lambda$. If $|x|$ is odd or x does not end in ')', S also rejects immediately; technically, the exit function writes a final '('. Then for all x, $|S(x)| \leq |x|/2$, and $x \in D_1 \iff S(x) \in D_1$.

The *Boolean sentence value problem* (BSVP) is to determine whether a given variable-free formula ϕ over the propositional constants 0 and 1 and the full set (or some subset such as $\{\wedge, \vee, \neg\}$ or $\{\mathrm{NAND}\}$) of propositional connectives evaluates to 1. For circuit complexity this is the same as the more-familiar problem: given a Boolean formula with variables $x_1, \ldots, x_n$ and an assignment $\vec{a} \in \{0,1\}^n$, does the assignment satisfy the formula? S. Buss [Bus87] was the first to show that this problem belongs to NC^1, and that it is complete for NC^1 under both AC^0 and DLOGTIME reductions. R. Kosaraju and A. Delcher [KD90] give NC^1 circuits of size $n \cdot \log^{O(1)} n$ which use the AKS sorting network. Buss defined a *PLOF* sentence to be one written in postfix notation with longer operands occurring before shorter ones. Because we picture finite-state machines as moving left-to-right, we reverse the sentence so that it is in prefix notation with shorter operands first, but still use the name PLOF. The problem remains complete for NC^1 under the restriction to PLOF sentences [Bus87, BCGR92].

Theorem 4.3 *The BSVP restricted to PLOF sentences is n-to-$n/2$-contractible by an SSM that needs only odd-even state information.*

There exist regular languages that are complete for NC^1 [Bar89], but the finite automata accepting them maintain the whole symmetric (or alternating) permutation group on 5 elements in their finite control. The odd-even construction here may have some algebraic significance.

Proof Sketch. After contracting any occurrence of two or more consecutive unary operators, we picture the PLOF sentence as a binary tree, with each leaf labeled 0 or 1, each interior node labeled by a binary connective $\circ$, and each edge labeled by a unary operator u. The point of (reverse) PLOF is that every leaf which is a right descender has a leaf for its sibling on the left. This property allows us to carry out the tree-contraction method of [ADKP89] with a finite-state machine S, and is preserved in each iteration of S. Briefly described: in a first pass, S marks every odd leaf a. In the second pass, S recognizes a marked left descender a by a pattern $yu_i \circ u_j a u_k z$ and translates it to $yu_{ijka}z$. In the third pass, S recognizes a marked right descender a with left sibling b by a pattern $yu_i \circ u_j b u_k a z$ and translates it to $yu_{ijka}bz$. The values of u_{ijka} come from a finite table (see section 3 of [ADKP89]). The three passes can be combined into one and remove half the leaves from the tree at each turn. □

The familiar Myhill-Nerode limitation on finite automata also sharply limits the structure of languages that admit constant-factor instance contraction by SSMs.

Proposition 4.4 *The following languages do not belong to* CFA*:*

(a) The language Pal of palindromes over $\{0, 1\}$.

(b) The language D_2 *of balanced strings of* $(,), [,]$.

(c) The language of binary trees, whether in infix, prefix, or postfix notation.

(d) The language of true PLOF sentences (without the "promise" of PLOF form). □

Pal and Boolean matrix multiplication are, however, NC^0-reducible to single finite-automaton computations, simply by shuffling a string with its reversal and shuffling rows with columns, respectively. However, whether the languages in (b)-(d) are NC^0-reducible to CFA seems not easy to answer. The main interesting open questions about CFA are whether it is all contained in NC^1, or alternatively whether it contains languages complete for L under NC^1 reductions.

5 Conclusions

We have boiled down previous definitions of self-reducibility into a form that enables us to ask fundamental questions about time- and space-bounded computation itself, not just for particular complexity bounds. We have placed logspace self-reducible languages fairly tightly between $\mathrm{DSPACE}[\log^2 n]$ and a hard place defined by Aux-PDAs with $O(\log^2 n)$ stack height. These include the Aux-PDAs in Savitch's theorem, but without polynomial time-bounds, and this is some evidence that logspace self-reducible languages may not all belong to P.

We have squeezed one more complexity class (actually several, but all $\leq_m^{log}$-equivalent) into the already well-populated interval between NL and the $\mathrm{DSPACE}[\log^2 n]$ upper bound from Savitch's Theorem. We hope that our tools will help further work shed light on the question of whether this upper bound is optimal. We did find an exact characterization of many-one time-bounded self-reducible classes, via simultaneous time and space complexity. This turned up a basic problem about time/space translation that seems to be still open.

Space-bounded self-reductions that halve the lengths of instances turn out to be equivalent to the general kind, up to space-bounded many-one reducibility. Sharper length-halving self-reductions, however, apply to complete problems for NC^1, and may be effective in analyzing the relationship between NC^1 and L.

References

[ADKP89] K. Abrahamson, N. Dadoun, D. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

[Bal90] J. Balcázar. Self-reducibility structures and solutions of NP problems. *Rivesta Matematica*, 4(4):175–184, Dec. 1990.

[Bar89] D. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comp. Sys. Sci.*, 38:150–164, 1989.

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21:755–780, 1992.

[BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.*, 6:305–321, 1977.

[BS85] J. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *Math. Sys. Thy.*, 18:1–10, 1985.

[Bus87] S. Buss. The boolean formula value problem is in ALOGTIME. In *Proc. 19th Annual ACM Symposium on the Theory of Computing*, pages 123–131, 1987.

[BvHT93] H. Buhrman, P. van Helden, and L. Torenvliet. P-selective self-reducible sets: A new characterization of p. In *Proc. 8th Annual IEEE Conference on Structure in Complexity Theory*, pages 44–51, 1993.

[Cho77] C. Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant des relations rationelles. *Theor. Comp. Sci.*, 5:325–337, 1977.

[HJ97] L. Hemaspaandra and Z. Jiang. Logspace reducibility: models and equivalences. *International Journal of Foundations of Computer Science*, 8, 1997.

[KD90] S.R. Kosaraju and A. Delcher. A tree-partitioning technique with applications to expression evaluation and term matching. In *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 163–172, 1990.

[Ko83] K. Ko. On self-reducibility and weak P-selectivity. *J. Comp. Sys. Sci.*, 26:209–211, 1983.

[LL76] R. Ladner and N. Lynch. Relativization of questions about log-space computability. *Math. Sys. Thy.*, 10:19–32, 1976.

[Lut97] J. Lutz. The quantitative structure of exponential time. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 225–260. Springer Verlag, 1997.

[MP79] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report TM-126, MIT, 1979.

[NSW92] N. Nisan, E. Szemerédi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.

[RS91] C. Reutenauer and M. Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20:669–685, 1991.

[RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computation. *J. Comp. Sys. Sci.*, 28:216–230, 1984.

[Sav70] W. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. Comp. Sys. Sci.*, 4:177–192, 1970.

[Sud78] I.H. Sudborough. On the tape complexity of deterministic context-free languages. *J. Assn. Comp. Mach.*, 25:405–414, 1978.

[WW86] K. Wagner and G. Wechsung. *Computational Complexity.* D. Reidel, 1986.

6 Appendix—Proof of Theorem 3.2

Let L be recognized by a $c \log n$ space-bounded Turing machine M with $d \log^2 n$ size-bounded pushdown, where c and d are both natural numbers. Without loss of generality we may assume that there is a unique accepting state of M and that before entering that unique accepting state M empties the stack, clears the work-tapes, and moves the heads all the way back to the leftmost positions. We also assume that N's rejection exhibits the same uniqueness. Define A to be the set of all strings of the form

$$\langle x\#w\#w'\#S, k + c\log|x|\rangle$$

satisfying the following conditions:

1. both w and w' are configurations on an input of length $|x|$ of the machine M viewed as a logspace machine without stack (i.e., they encode only the state, the head positions, and the work-tape contents);

2. S is a string of the stack symbols of M and is of length $(\log|x|)(d\log n - k)$; and

3. M on x, starting from configuration (w, S), arrives at (w', S) while keeping the stack height between $|S|$ and $d\log^2|x|$.

Then L is reducible to A by the reduction f_L that maps x to

$$\langle x\#I_0\#I_f\#\lambda, (d + c)\log|x|\rangle,$$

where I_0 is the initial configuration of M on x and I_f is the unique (by our assumption on M's behavior before accepting inputs) accepting configuration of M on x, where M is viewed as a logspace machine without stack. Define N to be the machine that, on an legitimate input $y = \langle x\#w\#w'\#S, k + c\log|x|\rangle$, simulates M on x starting from configuration (w, S) as follows:

(i) N stores the contents of M's stack above S in its work space as a string, say T.

(ii) If the configuration of M has become (w', S) then N immediately accepts y.

(iii) If M will pop in the current configuration and T is empty (namely the stack height is $|S|$) then N rejects y.

(iv) If M will push in the current configuration but $k = 0$ then N rejects y.

(v) If M will push in the current configuration and $|T| = \log |x|$ (namely the stack height is $|S| + \log |x|$) then N does the following:

Let z be the current configuration of M (not including the stack contents). For each configuration z' (again, not including the stack contents), N asks whether

$$\langle x\#z\#z'\#S \cdot T, (k-1) + c\log|x|\rangle$$

belongs to A, where $S \cdot T$ is the concatenation of S and T. N picks the first z' with respect to which A provides a positive answer to the query and continues the simulation from $(z', S \cdot T)$. If there is no such z' then N rejects y.

Note that if N encounters either the situation in (iii) or that in (iv) and rejects y then y does not satisfy Condition 3 of the definition of A in the above, so $y \notin A$. Also if N encounters the situation in (v) and if (w, S) encodes an actual configuration of M on S then there exists at least one z' with respect to which A returns a positive answer, so N will follow the computation of M. Thus, N decides A. By compressing tape symbols N's space requirement will be $\leq c \log |x|$ for all possible legitimate inputs whose first components has x as the first segment. Now define A' to be the union of A and the set $\{\, \langle x, (d+c)\log|x| + 1\rangle : x \in L \,\}$. Define N' to be the machine that, on input $u = \langle v, m\rangle$, many-one reduces v to $f_L(v)$ if $m = (d+c)\log|v| + 1$ and simulates N on u otherwise. Then N' with oracle A' accepts A'. Define $r(n) = (d+c)\log n + 1$. Then $L = \{\, x : \langle x, r(|x|)\rangle \in A' \,\}$. Thus L is DSPACE[$\log n$]-graded by a graded solver of Turing type, so $L \in$ LG. □