Fast Equality Test for Straight-Line Compressed Strings

Manfred Schmidt-Schauß[1] and Georg Schnitger[1]

Institut für Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de,
georg@thi.informatik.uni-frankfurt.de

Technical Report Frank-45

Research group for Artificial Intelligence and Software Technology
Institut für Informatik,
Fachbereich Informatik und Mathematik,
Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany

April 6, 2011

Abstract. The paper describes a simple and fast randomized test for equality of grammar-compressed strings. The thorough running time analysis is done by applying a logarithmic cost measure.

Keywords randomized algorithms, straight line programs, grammar-based compression

1 Introduction

Compression of data like strings and trees improves space usage, a well-known method is Lempel-Ziv encoding [JA84]. The standard way of applying algorithms to the data is a decompression prior to the application perhaps followed by a compression of the generated or modified data. Algorithms that can be translated such that they work efficiently on the compressed data are of interest, and complement the space efficiency by also improving running times.
To avoid the peculiarities of a specialized compression mechanism and to keep the generality of analyses, grammar based compression was proposed. The grammars are called *straight line programs (SLP)* which are used for string compression and algorithms on strings [Pla94,PR99,KRS95] as well as for compressions of trees and algorithms on them [BLM05,BLM08,GGSS08].

A central algorithmic problem used as a subalgorithm in several other algorithms on compressed data is the following: given two compressed representations r_1 and r_2, say of strings s_1 and s_2, respectively, decide whether $s_1 = s_2$. The first efficient algorithm that works without prior decompression is Plandowski's algorithm [Pla94,Pla95]. It uses grammars as compression device and shows that the equality test can be done in time polynomially in the size of the grammars. An improvement of this equality test is in [Lif07] where an algorithm is described that works in time $O(n^3)$, where n is the size of the grammar.
Randomized algorithms for variants of this test are described in [GKPR96] using 2×2 matrices and in [BKL$^+$02] for the generalisation to two-dimensional strings, where a polynomial interpretation is used.
In this paper we describe and analyze a randomized algorithm for Plandowski's equality problem that runs in quadratic time even using a logarithmic cost measure for arithmetic operations. It is correct if the answer is "no", and in case the answer is "yes", it is correct for identical strings and for nonidentical strings it does not detect inequality with a small probability δ, and with δ^n after n repetitions of the test. The algorithm requires modulo computation where the modulus is exponential in the size of G. It is open whether smaller numbers, for example numbers with a polynomial number of digits are sufficient. The randomized equality test is faster than the deterministic Lifshits-test [Lif07], which has a cubic running time, but presumably an $O(n^4)$ running time using the logarithmic cost measure. The equality test is also applicable to grammar-compressed ranked trees by applying it to the SLCF grammar representing the preorder traversals, which can be generated in linear time (see [BLM05,BLM08]).

2 Grammars and Equality

Definition 2.1. ***(a)*** *A straight-line context-free grammar (SLCFG) (equivalent to SLP) G is a quadruple $(\Sigma, \mathcal{N}, S, \mathcal{R})$ where*

(1) Σ is a finite alphabet, (we assume $|\Sigma| = O(1)$)
(2) $\mathcal{N} = \{B_1, \ldots, B_N\}$ is a set of nonterminals,
(3) $S = B_N$ is the start symbol and
(4) $\mathcal{R}$ is a finite set of productions. A production has either the form $B_i \rightarrow B_j B_k$ for $i > j, k$ or $B_i \rightarrow a$ for $a \in \Sigma$. Moreover for each nonterminal B_i there is exactly one production in $\mathcal{R}$.

(b) *Every nonterminal $A \in \mathcal{N}$ generates exactly one string $val(A)$. The string generated by the start symbol B_N is denoted by $val(G)$.*
(c) *The size $|G|$ of G is the number of productions of $\mathcal{R}$.*

The length of $val(G)$ may be as large as $2^{|G|}$. As an example, for every integer $n > 1$ there is an SLCFG G_n of size $\lceil(\log_2(n))\rceil$ such that $val(G_n)$ is a string of 0's of length n.
The *EQ problem* for SLCFGs is: given an SLCFG G and two nonterminals A_1, A_2, determine whether $val(A_1) = val(A_2)$.

Let $b \geq |\Sigma| + 1$ be a number (the base) and let num be an injective function num : $\Sigma \to \{1, \ldots, b-1\}$. Observe that a string $d = d_1 \cdots d_m$ over Σ can be interpreted as the b-ary representation of the natural number $\mathrm{num}_b(d) = \sum_{i=0}^{m-1} \mathrm{num}(d_{m-i}) \cdot b^i$. Hence $d = d'$ iff $\mathrm{num}_b(d) = \mathrm{num}_b(d')$.
The EQ problem is non-trivial, since the strings $val(A)$ may have length exponential in $|G|$, i.e as large as $2^{|G|}$. Thus the associated natural numbers $\mathrm{num}_b(A_i) := \mathrm{num}_b(val(A_i))$ may have representational size (number of digits) exponential in $|G|$. Therefore we determine $\mathrm{num}_b(A)$ modulo a randomly selected integer m smaller than some bound and check whether $\mathrm{num}_b(A_1) \equiv \mathrm{num}_b(A_2) \bmod m$ holds.
For computing running times we apply a logarithmic cost measure, i.e., $n_1 \circ n_2$ requires running time $O(\log n)$ for arithmetic operations including modulo computation where $n = \max(n_1, n_2)$. This is justified, since the representational size of numbers cannot be neglected in the analysis, even for the computation of $|val(G)|$.

3 A Randomized Equality Test

We analyze the properties of a randomized equality test for natural numbers. Let $e = 2.718\ldots$ be the Euler-number.

Fact 3.1. Let $c \geq e$ be arbitrary and let a be a positive integer. For any two natural numbers $x, y < a$, if $x \neq y$ then

$$x \equiv y \bmod p$$

holds with probability at most $\ln(ec)/c$, provided a prime $p \leq 2c \cdot \ln a$ is selected uniformly at random.

Proof. First we show that asymptotically the number of prime divisors of an integer a is less than $\pi(2 \ln a)$ where $\pi(z)$ is the number of primes less than z. First observe that $\sum_{p, p \leq N} \ln p \sim N$, where we sum over all primes at most N (see [BS96]). As a consequence $\ln(\prod_{p,p \leq N} p) \geq N/2$ and hence $\prod_{p,p\leq N} p \geq e^{N/2}$. Let $0 < x < a$ and P_x be the set of all primes p with $x \equiv 0 \bmod p$. Then $\prod_{p \in P_x} p$ is a divisor of x and in particular $\prod_{p \in P_x} p < a$.
If $|P_x| \geq \pi(2 \ln a)$, then by our previous argument

$$a \leq \prod_{p, p \leq 2 \ln a} p \leq \prod_{p \in P_x} p < a$$

and hence $|P_x| \leq \pi(2 \ln a)$ follows.
We apply the prime number theorem and obtain $\pi(z) \sim z/\ln(z)$. As a consequence $\pi(c \cdot z) \sim cz/\ln(cz) \geq [c/\ln(ec)] \cdot [z/\ln(z)]$, provided $c, z \geq e$. Hence $\pi(c \cdot z) \geq [c/\ln(ec)] \cdot \pi(z)$. We set $z = \ln a$. If we choose a prime $p \leq cz$ at random, then we do not detect inequality of x and y with probability at most $\pi(z)/\pi(cz) \leq \ln(ec)/c$.

An alternative method is testing division modulo an arbitrary number m, which does not require to find prime numbers and saves a factor $|G|$ in the overall running time see Remark 4.5.

Fact 3.2. Let a be a positive integer. For any two natural numbers $x, y < a$, if $x \neq y$ then

$$x \equiv y \bmod m$$

holds with probability at most 0.5, provided a number $m \leq (2 \ln a)^2$ is selected uniformly at random, and $2 \ln a \geq 355991$.

Proof. The proof of Fact 3.1 shows that if $0 < x < a$ and P_x is the number of primes p with $x \equiv 0 \mod p$, then $|P_x| \leq \pi(2 \ln a)$.
In an interval $[k, k^2]$, the number of multiples $k'p \in [k, k^2]$ of primes $p \in [k, k^2]$ has as lower bound $\int_{2k}^{k^2} k^2/(x \ln(x))dx = k^2(\ln\ln(k^2) - \ln\ln(2k))$ provided k is sufficiently large ($k \geq 355991$). (For a rigorous argument, taking into account that we use an approximation of the number of primes and their density see [SSS11]. Using the result that the probability that a number m has a prime factor at least $\sqrt{m}$ approaches $\ln 2$, see e.g. [Dic30,D.E98], it is easy to derive that the estimation holds in the interval $[k, k^4]$ since the probability that numbers from $[k^2, k^4]$ have a prime factor at least k approaches $\ln 2$.) Note that the multiples are unique in the interval. An easy computation shows that the ratio compared to all integers in the interval, which is $k^2 - 2k + 1$, is > 0.6 for $k \geq 10^6$ and approaches $\ln 2 = 0.693..$ if $k \to \infty$.
Since some primes from P_x may be in the interval, a lower bound for the number of integers in $[k, k^2]$ with a prime divisor not in P_x is $0.6k^2 - |P|k$. For $k = (2 \ln a)$, we obtain a ratio $(0.6k^2 - |P_x|k)/k^2 = 0.6 - 1/(2 \ln\ln a) > 0.5$.
If we select $m \leq (2 \ln a)^2$ uniformly at random, then $x - y \equiv 0 \bmod m$ holds with probability at most 0.5.

4 Equality-Test Algorithms

By utilizing a table with $B_i \mapsto |\mathrm{val}(B_i)|$ computed as $|val(B_i)| := 1$ if $B_i \to a$ is the production, and $|\mathrm{val}(B_i)| := |\mathrm{val}(B_j)| + |\mathrm{val}(B_{j'})|$ if $B_i \to B_j B_{j'}$ is the production for B_i, and taking care of the logarithmic cost measure, we obtain:

Observation 4.1. For an SLCFG G the length of $\mathrm{val}(A)$ can be determined simultaneously for all nonterminals A in time $O(|G| \cdot \log |val(G)|)$.

Given a positive integer m, we store the values $(b^{|val(A)|} \bmod m)$ in a table τ computed as follows: $\tau(B_i) := (b \bmod m)$ for $B_i \to a$, and $\tau(B_i) := (\tau(B_j) \cdot \tau(B_{j'}) \bmod m)$ for $B_i \to B_j B_{j'}$. We also determine $(\mathrm{num}_b(B) \bmod m)$ for all nonterminals B using another table σ computed as follows: $\sigma(B_i) := (\mathrm{num}(a) \bmod m)$ if $B_i \to a$ and $\sigma(B_i) := (((\sigma(B_j) \cdot \tau(B_{j'}) \bmod m) + \sigma(B_{j'})) \bmod m)$ if $B_i \to B_j B_{j'}$. Since addition and multiplication are modulo m, the entries in σ, τ are smaller than m.

Observation 4.2. Assume that an SLCFG $G = (\Sigma, \mathcal{N}, S, \mathcal{R})$ and a positive integer $m \geq b$ is given. Then the numbers $(b^{|val(B)|} \bmod m)$ and $(\mathrm{num}_b(B) \bmod m)$ can be determined simultaneously for all $B \in \mathcal{N}$ in time $O(|G| \cdot \log m)$.

Algorithm 4.3 (Equality Test by Modulo). Checking whether $val(A_1) = val(A_2)$ holds requires first to determine the lengths $|val(A)|$ for all nonterminals A with Observation 4.1. We then randomly select a number $m \leq (2 \cdot \ln(b^{|val(G)|}))^2 = (2 \cdot \ln b)^2 |val(G)|^2$ and determine $(\mathrm{num}_b(A_1) \bmod m)$ and $(\mathrm{num}_b(A_2) \bmod m)$ with Observation 4.2, and then compare the outcomes.

Theorem 4.4 (Modulo-test). *Assume that an SLCFG G and two nonterminals A_1, A_2 of G are given. Using Algorithm 4.3: if the answer is "no", then $val(A_1) \neq val(A_2)$. If the answer is "yes", then the answer is correct if $val(A_1) = val(A_2)$; in case $val(A_1) \neq val(A_2)$, then we do not detect inequality with probability at most* 0.5. *The running time is bounded by* $O(|G| \cdot \log |val(G)|)$.

Since $|val(G)| \leq 2^{|G|}$, the running time is at most quadratic.

Remark 4.5 (Equality Test by Modulo Primes). Using Fact 3.1 allows to modify Algorithm 4.3 using primes in the range up to $2 \cdot \ln(b^{|val(G)|})$. However, randomly selecting primes requires first to select numbers, and check them for being prime, and iterating this until a prime is found. The density of primes in this range is $(\ln |val(G)|)^{-1}$, hence in the worst case $O(|G|)$ numbers have to be tried. The computational cost (using the logarithmic cost measure) for primality testing of m are for the known tests at least $O(\log^2 m)$, which sums up in the worst case to at least $O(|G|^3)$.

As a special case, let Σ be a one-letter alphabet. If we have to check whether $val(A_1) = val(A_2)$ holds it is sufficient to check whether $|val(A_1)| = |val(A_2)|$. Therefore, given a number $m \geq b$, we compute a table with $lmod(B_i) = 1$ for rules $B_i \to a$, and $lmod(B_i) = ((lmod(B_j) + lmod(B_{j'})) \bmod m)$ for rules $B_i \to B_j B_{j'}$. In analogy to Observation 4.2, this can be done in time $O(|G| \log m)$. We use Fact 3.2 with $a = |val(G)|$, and exploit $\ln a \leq |G|$.
We randomly select a number $m \leq (2 \cdot |G|)^2$. Finally, with Fact 3.2, we do not detect inequality with probability at most 0.5.

Theorem 4.6. *For a one-letter alphabet Σ Theorem 4.4 holds with a running time $O(|G| \cdot \log |G|)$.*

Note that this is faster than the naive comparison $|val(A_1)| = |val(A_2)|$, which runs in time $O(|G| \cdot \log |val(G)|)$, resp. quadratic in the worst-case.

Remark 4.7 (Some Practical Hints). The theoretical results may require large modulo-numbers, seen from a practical viewpoint, for a safe randomized test. However, since SLCFGs-generated numbers are rare, in practice smaller modulo-bases may be sufficient. However, it is not hard to see that the selection of prime numbers $\leq |G| \ln 2$ is unsafe (see [SSS11]).

Assuming ideal properties, the following computation is possible and gives a rough estimate for the practically necessary range of tiny numbers (or primes) (mathematically unsafe, but useful as a practical hint). Given $|G|$ and assuming $|G| \geq b$, there are at most $|G|^{|G|^2}$ different grammars of size $|G|$. Assuming that the generated numbers are all different and are exactly $1, \ldots, |G|^{|G|^2}$, then using Fact 3.1, we obtain that primes in the range up to $2\log(|G|) \cdot |G|^2$ (resp. modulo numbers m up to $(2\log(|G|) \cdot |G|^2)^4$) have to be chosen for a useful modulo test with tiny primes.

Algorithm 4.8. Modulo-test with *tiny* numbers. Use the randomized algorithm 4.3, but use numbers in the range up to $(2\log(|G|) \cdot |G|^2)^2$.

Conjecture 4.9. Algorithm 4.8 is correct in the sense of Theorem 4.4, perhaps with another polynomial upper bound for the range of m.

Remark 4.10. Our algorithm can also be applied to the EQPREF-problem. This problem was also considered in [GKPR96]. The *EQPREF problem* for SLCFGs is: given an SLCFG G and two nonterminals A_1, A_2 determine whether $val(A_1) = val(A_2)$ and if $val(A_1) \neq val(A_2)$ then determine the length of the longest common prefix of $val(A_1)$ and $val(A_2)$.
We perform an interval bisection method. Given an SLCFG G, a single bisection step requires to compute the length, and two grammars G_1, G_2 according to the bisection, where G_1, G_2 are smaller than G. The number of bisection steps is $O(\log|val(G)|)$. The construction can be done in running time $O(|G| \cdot \log|val(G)|)$ and the test in time $O(|G| \cdot \log|val(G)|)$. The test must be repeated several times in every step, where a fixed number like 20 or 50 may be used. This sums up to $O(|G| \cdot \log^2|val(G)|)$ using a logarithmic cost measure. Under a uniform cost measure we obtain $O(|G| \cdot \log|val(G)|)$.

5 Summary

The following table summarizes the complexities of the different randomized or sample equality tests, where we use the logarithmic cost measure.

Modulo-Test Algorithm	worst case	general case
modulo small m (Alg. 4.3)	$O(\|G\|^2)$	$O(\|G\|\log(\|val(G)\|))$
modulo tiny m (Alg. 4.8)	$O(\|G\|\log(\|G\|))$	$O(\|G\|\log\log m)$
$\|\Sigma\| = 1$ (Theorem 4.6)	$O(\|G\|\log\|G\|)$	$O(\|G\|\log\|G\|)$

References

[BKL+02] Piotr Berman, Marek Karpinski, Lawrence L. Larmore, Wojciech Plandowski, and Wojciech Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. Syst. Sci.*, 65(2):332–350, 2002.

[BLM05] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML documents. In *Proc. of DBPL 2005*, volume 3774 of *LNCS*, pages 199–216, 2005.

[BLM08] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4–5):456–474, 2008.

[BS96] E. Bach and J. Shallit. *Algorithmic Number Theory Vol 1: Efficient Algorithms.* MIT Press, Cambridge, MA, 1996.

[D.E98] D.E.Knuth. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, MA, 1998.

[Dic30] K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude. *Arkiv för Mat., Astron. och Fys*, 22A(10):1–14, 1930.

[Dus98] Pierre Dusart. *Autour de la fonction qui compte le nombre de nombres premiers.* PhD thesis, Université de Limoges, 1998. Nr. 17-1998.

[GGSS08] A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context matching for compressed terms. In *LICS 2008*, pages 93–102. IEEE Computer Society, 2008.

[GKPR96] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Randomized efficient algorithms for compressed strings: The fingerprint approach (extended abstract). In *7th CPM 96*, volume 1075 of *LNCS*, pages 39–49. Springer, 1996.

[JA84] J.Ziv and A.Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Inf. Theory*, 17:8–19, 1984.

[KRS95] M. Karpinski, W. Rytter, and A. Shinohara. Pattern-matching for strings with short description. In *CPM '95*, number 937 in LNCS, pages 205–214. Springer-Verlag, 1995.

[Lif07] Yury Lifshits. Processing compressed texts: A tractability border. In *18th CPM 2007*, number 4580 in LNCS. Springer, 2007.

[Pla94] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470. Springer, 1994.

[Pla95] Wojciech Plandowski. *The Complexity of the Morphism Equivalence Problem for Context-Free Languages.* PhD thesis, Department of Mathematics, Informatics and Mechanics, Warsaw University, 1995.

[PR99] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever*, pages 262–272. Springer, 1999.

[SSS11] Manfred Schmidt-Schauß and Georg Schnitger. Fast equality test for straight-line compressed strings. Frank report 45, Institut für Informatik, Goethe-Universität Frankfurt am Main, April 2011.

A An Estimation of the Number of Multiples of Primes

Let $\mathbb{P}$ be the set of primes and $\pi(x)$ be the number of primes smaller than x. An estimation for $\pi(x)$ is (see [BS96,Dus98]):

$$\begin{array}{lll} \frac{x}{\ln x}\big(1+\frac{1}{\ln x}\big) < \pi(x) & & \text{for } x \leq 599 \\ \pi(x) & < \frac{x}{\ln x}\big(1+\frac{1}{\ln x}+\frac{2.51}{(\ln x)^2}\big) & \text{for } x \leq 355991 \end{array}$$

The goal is to prove that asymptotically, the ratio of multiples of primes from $[k, k^2]$ that are also in the interval $[k, k^2]$ approaches $\ln 2 \approx 0.69$. This has similarity to the result that the probability that a number m has a prime factor at least $\sqrt{m}$ approaches $\ln 2$ (see e.g. [Dic30,D.E98]). This result implies that our estimation holds in the interval $[k, k^4]$ since the probability that numbers from $[k^2, k^4]$ have a prime factor at least k approaches $\ln 2$.
In the following paragraph, we make a rigorous, but elementary, estimation for the interval $5k, k^2]$.

A.1 An Estimation under Uncertainty

Let there be a real (positive) interval $[a, b]$, a monotone ascending function $f : [a, b] \to \mathbb{R}$, such that $\pi(x) \geq f(x) > 0$ for all $x \in [a, b]$, a positive, monotone descending function $g : [a, b] \to \mathbb{R}$. Note that $\pi(x)$ is also monotone ascending. Assume that the derivative f' exists, is continuous and monotone.
The goal is to determine a lower bound of

$$S_0 := \sum_{x \in \mathbb{P} \cap [a,b]} g(x)$$

We select a sequence $a = a_0 < a_1 < \ldots < a_n = b$. Let us assume that it is possible to select a_1 such that $\rho := \pi(a_1) - \pi(a_0) \geq \pi(a_1) - f(a_1)$.
The idea is to omit the sum $\sum_{x \in \mathbb{P} \cap]a_0,a_1]} g(x)$ from the sum S_0 above and use this to smooth the other sum contributions.
We use a step function w.r.t. the chosen sequence:

$$\Big(\sum_{i=1,\ldots,n} (\pi(a_i) - \pi(a_{i-1}))g(a_i)\Big) \leq \sum_{x \in \mathbb{P} \cap [a,b]} g(x)$$

Let $R_0 := \sum_{x \in \mathbb{P} \cap [a_0,a_1]} g(x)$ and $S := \big(\sum_{i=2,\ldots,n} (\pi(a_i) - \pi(a_{i-1}))g(a_i)\big)$.
For $i = 1, \ldots, n-1$ let:

$$\begin{aligned} R_i &:= (\pi(a_{i+1}) - \pi(a_i))g(a_{i+1}) - (f(a_{i+1}) - f(a_i))g(a_{i+1}) + R_{i-1} \\ R'_i &:= (\pi(a_{i+1}) - \pi(a_1) + \rho)g(a_{i+1}) - (f(a_{i+1}) - f(a_1))g(a_{i+1}) \end{aligned}$$

Lemma A.1. *The following estimations hold:*

- $R_0 \geq \rho g(a_1)$.

– $R_i \geq R'_i \geq 0$ *for all* $i \geq 1$.

Proof. Since $\pi(a_1) - \pi(a_0) = \rho$ and g is positive and monotone decreasing, we obtain $R_0 \geq \rho g(a_1)$.
Since $\pi(a_2) \geq f(a_2)$, $R_0 \geq \rho g(a_1) \geq \rho g(a_2)$, and $\rho \geq \pi(a_1) - f(a_1)$, we obtain:

$$\begin{aligned} R_1 = &\quad \big((\pi(a_2) - \pi(a_1)) - (f(a_2) - f(a_1))\big)g(a_2) + R_0 \\ \geq R'_1 = &\quad \big((\pi(a_2) - \pi(a_1) + \rho) - (f(a_2) - f(a_1))\big)g(a_2) \\ \geq 0 & \end{aligned}$$

Since $\pi(a_{i+1}) \geq f(a_{i+1})$ and $\rho \geq \pi(a_1) - f(a_1)$, we see that for all $i \geq 1$, the inequation $R'_i = (\pi(a_{i+1}) - \pi(a_1) + \rho)g(a_{i+1}) - (f(a_{i+1}) - f(a_1))g(a_{i+1}) \geq 0$ holds.
For $i = 2, \ldots, n-1$: we show the inequation $R_i \geq R'_i$ by induction on i:

$$\begin{aligned} R_i &= (\pi(a_{i+1}) - \pi(a_i))g(a_{i+1}) - (f(a_{i+1}) - f(a_i))g(a_{i+1}) + R_{i-1} \\ &\geq (\pi(a_{i+1}) - \pi(a_i))g(a_{i+1}) - (f(a_{i+1}) - f(a_i))g(a_{i+1}) + R'_{i-1} \\ &= \quad (\pi(a_{i+1}) - \pi(a_i))g(a_{i+1}) - (f(a_{i+1}) - f(a_i))g(a_{i+1}) \\ &\quad + (\pi(a_i) - \pi(a_1) + \rho)g(a_i) - (f(a_i) - f(a_1))g(a_i) \end{aligned}$$

Since R'_{i-1} is positive, we can replace $g(a_i)$ by $g(a_{i+1})$ and obtain:

$$\begin{aligned} &\geq \quad (\pi(a_{i+1}) - \pi(a_i))g(a_{i+1}) - (f(a_{i+1}) - f(a_i))g(a_{i+1}) \\ &\quad + (\pi(a_i) - \pi(a_1) + \rho)g(a_{i+1}) - (f(a_i) - f(a_1))g(a_{i+1}) \\ &= (\pi(a_{i+1}) - \pi(a_1) + \rho)g(a_{i+1}) - (f(a_{i+1}) - f(a_1))g(a_{i+1}) \\ &= R'_i \geq 0 \end{aligned}$$

The previous lemma shows that

$$\Big(\sum_{i=1,\ldots,n} (\pi(a_i) - \pi(a_{i-1}))g(a_i)\Big) \geq \Big(\sum_{i=2,\ldots,n} (f(a_i) - f(a_{i-1}))g(a_i)\Big)$$

Using a sequence $a_1, \ldots, a_n$ with $a_i = a_1 + ih$, we obtain using $h \to 0$:

$$\sum_{i=1,\ldots,n} (f(a_i) - f(a_{i-1}))g(a_i) \geq \int_{a_1}^{b} f'(x)g(x)dx$$

Number of Multiples of Primes in an Interval Now we apply this to the following problem concerning prime numbers and their multiples. Let $k \geq 355991$. A lower bound on the the following number is required:

$$|\{x \mid x \in [k, k^2], x \text{ has a prime-factor in } [k, k^2]\}|$$

This is the same as the number of multiples of the primes $p \in [k, k^2]$ that are also in $[k, k^2]$. If $p \in [k, k^2]$ is any prime, then for a multiple $ip \in [k, k^2]$, we obtain $i < k$, hence the multiples are all distinct. We use $\frac{x}{\ln x}\big(1 + \frac{1}{\ln x}\big)$ as the

function f, and $g = \dfrac{k^2}{x}$ for counting the number of multiples of x in the interval. For the integral we use a further estimation: $\dfrac{x}{\ln x}\big(1 + \dfrac{1}{\ln x}\big) \geq \dfrac{x}{\ln x} =: f_1$, and thus $\dfrac{1}{\ln x}$ as the density f_1'.
For a safe estimation, we have to cut away a prefix of the interval $[k, k^2]$. For large numbers k, ρ is roughly $\dfrac{2k}{\ln(2k)} - \dfrac{k}{\ln k}$ which for $k \geq 355991$ is larger than $\dfrac{3k}{(\ln k)^3}$. Thus, using Dusart's formula, it is sufficient to cut away the interval $[k, 2k]$ and use $[2k, k^2]$ for the integral.
According to the above estimation method, we obtain the following lower bound on the number of multiples:

$$\int_{2k}^{k^2} k^2/(x\ln(x))dx = k^2(\ln\ln(k^2) - \ln\ln(2k))$$

An easy computation shows the following: the ratio is $\approx \ln\ln(k^2) - \ln\ln(2k)$ which is > 0.55 and approaches $\ln 2 = 0.693\ldots$ if $k \to \infty$.

B On Minimal Number of Primes

Remark B.1 (Lower Bound for Primes).
We show that prime factors greater than $(\ln 2) \cdot |G|$ are required for the equality test. More rigorously:
Let $\Sigma = \{0, \ldots, b-1\}$, n be an integer and let G be an SLCFG such that $val(A_0)$ is a string of only 0's of length n, and $val(A_1)$ is a string of only 1's of length n. G can be chosen such that $|G| \leq 2\lceil\log_2(n)\rceil$. Let $b < p_1 < p_2 < \ldots p_k \leq N$ be the sequence of all primes greater than b and bounded by N, and let $n = (p_1 - 1) \cdot \ldots \cdot (p_k - 1)$. For all $p \in \{p_1, \ldots, p_k\}$: $(\sum_{i=0}^{p-2} b^i) \cdot (b-1) = b^{p-1} - 1 \equiv 0 \mod p$, since $p > b$. Hence $(\sum_{i=0}^{p-2} b^i) \equiv 0 \mod p$, and $val(A_0)$ and $val(A_1)$ are indistinguishable by the modulo algorithm for all primes $p \in \{p_1, \ldots, p_k\}$. Using the equations in the proof of Fact 3.1 we obtain $\ln n = \ln\prod_{p,b<p\leq N}(p-1) \leq \ln\prod_{p,p\leq N} p \sim N$. Thus $\ln\prod_{p,p\leq N} p < 1.1N$, for N not too small, and also $\ln n = (\ln 2)\cdot\log_2(n) \geq (\ln 2)\cdot(\lceil G\rceil - 1)$. Thus $N > c\cdot|G|$ with $c \approx \ln 2/1.1 \approx 0.6$. Hence, it must be possible to select prime factors larger than $(\ln 2) \cdot |G|$.