Streaming Property Testing of Visibly Pushdown Languages

Nathanaël François Frédéric Magniez Michel de Rougemont Olivier Serre

SUBLINEAR Workshop - January 7, 2016
Definition

A VPL is a language of $\Sigma = \Sigma_+ \cup \Sigma_- \cup \Sigma_-$ that is recognized by a stack automaton that pushes when it reads a symbol in $\Sigma_+$ and pops when it reads a symbol of $\Sigma_-$. In particular, a regular tree language with the tree read in DFS order (such as an XML document) is a VPL.
Motivation and context

- Checking the validity of large documents needs to be done efficiently.
- High stack \(\rightarrow\) cannot be done with small memory in streaming.
- An efficient property tester can pre-reject some documents before a more costly check.

VPLs are hard to recognize in streaming and hard to test for in the query model:

- Recognizing some VPLs in streaming requires memory \(\Omega(n)\).
  (Disjointness)
- A query-model property tester for the parenthesis language requires \(\Omega(n^{1/11})\) queries. [Parnas, Ron, Rubinfeld ’03]
Consider $\nu = \nu_+ \nu_-$. We still want to know if $\nu \in L$.

- Known to be hard to decide exactly (encoding of Set Disjointness).
- Any solution may give insight to general problem.
“Slices” of $\nu$ can be interpreted as a word $\hat{\nu}$, with $\hat{\nu}$ in some regular language if and only if $\nu \in L$.
There is an algorithm for testing regular languages in \( O(1/\varepsilon^2) \) non-adaptive queries [Alon, Krievelich, Newman, Szegedy ’00]. To get sampling of \( \hat{v} \), remember sampled letters in \( v_+ \) (memory \( O(1/\varepsilon.\log n) \) for the heights) then read letters of matching height in \( v_- \).

Can do more than just accept/reject: can test for all pairs of states \((p, q)\) if there is a run of \( A \) on \( v \) from \( p \) to \( q \).

We now have a black-box streaming tester for non-alternating sequences that outputs some \( R \subset Q \times Q \) indicating the possible beginning and end states for \( v \). From this we build an algorithm for the general problem.
FROM non-alternating sequences TO THE general PROBLEM: general IDEA

- Input $x \in \Sigma^*$. 
- Find $v$ a “peak” in $x$ and use the non-alternating sequence tester on it.
- Repeat this process
From non-alternating sequences to the general problem: general idea

- Input $x \in \Sigma^*$.
- Find $v$ a "peak" in $x$ and use the non-alternating sequence tester on it.
- Repeat this process.
From non-alternating sequences to the general problem: General idea

- Input $x \in \Sigma^*$.
- Find $v$ a "peak" in $x$ and use the non-alternating sequence tester on it.
- Repeat this process.
To compute $R$ from a peak $v$ we need $1/\varepsilon^2$ samples inside the peak.

We do not know in advance how large the peaks will be.

Perform $(1 + \varepsilon)$-suffix sampling: reservoir sampling on several suffixes $w_1, \ldots, w_{j_v}$, each $(1 + \varepsilon)$ times large than the last.

$$w_1 = v(1, i): \frac{1}{\varepsilon^2} \text{ samples} \quad \leftarrow \quad 1/\varepsilon^2 \text{ samples} \quad \quad \quad \text{w}_3: 1/\varepsilon^2 \text{ samples} \quad \leftarrow \quad \text{w}_{j_v} = v(i)$$

$$v(1) \quad \quad \quad \quad \text{w}_2: 1/\varepsilon^2 \text{ samples} \quad \leftarrow \quad \frac{1}{\varepsilon^2} \text{ samples} \quad \text{w}_4: 1/\varepsilon^2 \text{ samples}$$

Total amount of samples: $\log(|v|)/(\varepsilon^2 \log(1 + \varepsilon)) \approx \log(|v|)/\varepsilon^3$. 
From non-alternating sequences to the general problem:
handling $R$’s

- Each $R$ corresponds to some $v'$ potentially $\varepsilon|v|$-far from peak $v$.
- If too many $R$’s within $R$’s, risk of accumulation of error.

Solution: not compute $R$ immediately, wait to see if the next peak is much smaller.
- This has a cost: $\log n$ peaks waiting in the stack.
- $\log n$ potential nested $R$’s mean we have to use $\varepsilon' = \varepsilon/\log n$ for the tester for peaks.
FROM NON-ALTERNATING SEQUENCES TO THE GENERAL PROBLEM:
HANDLING $R$’S

- Each $R$ corresponds to some $v'$ potentially $\varepsilon|v|$-far from peak $v$.
- If too many $R$’s within $R$’s, risk of accumulation of error.

Solution: not compute $R$ immediately, wait to see if the next peak is much smaller.
- This has a cost: $\log n$ peaks waiting in the stack.
- $\log n$ potential nested $R$’s mean we have to use $\varepsilon' = \varepsilon / \log n$ for the tester for peaks.
Algorithm for the general case

- Use $\varepsilon' = \varepsilon / \log n$ because of error accumulation.
- Maintain $\log^3 n/\varepsilon^2$ independent and well-distributed sampling of factors of size $\log n/\varepsilon$.
- Because computing $R$’s messes with the sampling, we in fact need memory $O(\log^6 n/\varepsilon^4)$.
- Maintain a stack of past peaks not transformed into a $R$ yet.
- If a peak is finished (i.e. returned to starting height), compute the $R$, get previous peak out of the stack.
- If current peak has at least half the weight of previous peak (in the stack), remove that peak from the stack and compute the $R$.
- Total memory cost: $O(\log^7 n/\varepsilon^4)$. 

$\rightarrow R$

$\rightarrow R$
There may still be some hope of reducing the memory cost:

- Maybe each element of the stack does not need to preserve all the sampling as it grows older. This would remove a log $n$ factor.
- One of the log $n$ factors is due to the assumption that all $R$’s are correct (up to a relative error of $\varepsilon$) with high probability. Maybe we can afford a few completely wrong $R$’s.
- The high stack, small peaks, and nested $R$’s are what makes our algorithm costly. But they mostly occur when the height is low, and we have an exact algorithm for checking VPLs with memory cost $\text{height}(x)$ (run the automaton). Can we find a compromise?

Thank you for your attention