A Natural Problem that is NOT in P
Exposition by Bill Gasarch

1 Introduction

We present the proof, due to Meyer and Stockmeyer [1] that the following (somewhat natural) problem is NOT in P (without any assumptions). It will take some definitions to get to it.

Def 1.1 Let Σ be a finite alphabet. A *Regular Expression* (henceforth Reg Exp) is defined as follows.

- For all $\sigma \in \Sigma$, σ is a reg exp.
- $\emptyset$ is a reg exp
- If α and β are reg exps then so is $\alpha \cup \beta$, $\alpha\beta$, and α^*

If α is a regular expression then $L(\alpha)$ is the set of strings that α generates.

In a textbook you might see the expression a^5b^2. This is not formally a Reg Exps; however, its meaning is clear and it can can be rewritten as the regular expression *aaaaabb*. Does allowing exponents matter? In terms of getting more languages NO, you will still get regular languages. In terms of length of representation YES. If we allow exponents then Reg Exps can be represented far more compactly. Note that a^n takes $O(\log n)$ space to write, where as $aaa\cdots a$ (n times) takes $O(n)$ space to write.

We define textbook reg expressions.

Def 1.2 Let Σ be a finite alphabet. A *Textbook Regular Expression* (henceforth t-Reg Exp) is defined as follows.

- For all $\sigma \in \Sigma$, σ is a t-reg exp.
- $\emptyset$ is a t-reg exp
- If α and β are t-reg exps then so is $\alpha \cup \beta$, $\alpha\beta$ and α^*
- If α is a t-reg exp and $n \in \mathsf{N}$ then α^n is a t-reg exp.

If α is a t-reg exp then $L(\alpha)$ is the set of strings that α generates.

Here is the question which we call *t-reg expression equivalence*

$$TRE = \{\alpha \text{is a t-reg exp} \mid L(\alpha) \neq \Sigma^*\}.$$

Note 1.3 In the original paper they called t-regular expressions *Regular expression with squaring*. For example, they would write a^{10} as $(((a)^2)^2)^2aa$.

How hard is TRE?

Theorem 1.4 $TRE \in DTIME(O(2^{2^{O(n)}}))$.

Proof:

1. Input(α), length n.
2. Replace σ^m with m σ's to get reg exp β which is equivalent to t-reg exp α. Note that β is of length $2^{O(n)}$.
3. Convert β to an NDFA M of size $2^{O(n)}$.
4. Look at ALL strings of length $2^{O(n)}$ For each one run it through the NDFA by keeping track of the set of states you might be in.

This takes time $O(2^{2^{O(n)}})$.
∎

Can we do better if we do this non-deterministically? Yes- sort of. We can do better on space.

Theorem 1.5 $TRE \in NSPACE(2^{O(n)}) \subseteq NEXPSPACE$.

Proof:

1. Input(α), length n.
2. Replace σ^m with m σ's to get reg exp β which is equivalent to t-reg exp α. Note that β is of length $2^{O(n)}$.
3. Convert β to an NDFA M of size $2^{O(n)}$.
4. Guess a string char-by-char. As you guess it keep track of the SET of states you would be in. If ever you find a set of states where all are NOT final states, then there must be a string that is NOT accepted. Stop and say YES.

This takes space $O(2^{O(n)})$. ∎

We will show that TRE *requires* this much space. Hence (obviously) TRE *requires* this much time. Hence TRE is NOT in P.

2 Completeness

Recall:

Def 2.1

1. $PSPACE = DSPACE(n^{O(1)})$.
2. $NEXPSPACE = DSPACE(2^{n^{O(1)}})$.

We want a notion of a problem being hard for these classes.

Def 2.2 Let A, B be sets. $A \leq B$ if there exists $f \in P$ such that

$$(\forall x)[x \in A \text{ iff } f(x) \in B].$$

We leave the following proof to the reader.

Lemma 2.3 *If $A \leq B$ and $B \in P$ then $A \in P$.*

Def 2.4 Let X be any complexity class.

1. A set B is *X-hard* if, for all $A \in X$, $A \leq B$.
2. A set B is *X-complete* if $B \in X$ and B is X-hard.

We want the following to be true:

Conjecture 2.5 *Let X be any complexity class. Let B be X-complete. If $A \leq B$ then $A \in X$.*

There are some X's for which this is not true. Consider $DTIME(2^{O(n)})$. Let B be $DTIME(2^{O(n)})$-complete. Let $A \leq B$ via f. Here is the obvious algorithm for A:

On input x, compute $f(x)$ and test if it is in B. If $f(x) \in B$ then output YES, else output NO.

This might not work. For example, f might be in time $O(n^2)$. Hence $|f(x)| \leq |x|^2$. So the question $f(x) \in B$ takes time $2^{O(n^2}$.

The reason that this happened is that $DTIME(2^n) \neq DTIME(2^{p(n)}$.

We are NOT going to formally define a notion of *closed under polynomial stuff*. However, we will only deal with such classes.

Theorem 2.6 *Let X be a complexity class such that $P \subset X$ (strict containment). If A is X-hard then $A \notin P$.*

Proof: Assume, by way of contradiction, that $A \in P$. Let $C \in X - P$. Since A is X-hard we have $C \leq A$. By Lemma 2.3 $C \in P$. This contradicts $C \notin P$. ∎

Theorem 2.6 gives us a way to show that a set (perhaps even a natural one!) is NOT in P. Recall that $PSPACE \subset EXPSPACE$ by the Space Hierarchy Theorem. Hence if A is $EXPSPACE$-hard then A is NOT in $PSPACE$. Note that we are actually using the Space Hierarchy theorem to prove that a somewhat natural language is NOT in P, even though the Space Hierarchy theorem itself yielded a non-natural language that is $EXPSPACE - PSPACE$.

3 TRE is $NEXPSPACE$-Complete

Before we prove this theorem we need to set up conventions for Nondet TMs.

Notation 3.1 Let M be a NONDET TM that runs in space $S(n)$. Let x be of length n. Let Σ be the alphabet to the TM. Let \$ be a symbol that is not in Σ. We will use it as a separator. Let Q be the states. Let $\Gamma = \Sigma \cup \Sigma \times Q \cup \{\$\}$ A *configuration* is a string that (1) begins with \$, (2) after the \$ has $S(n)$ characters, and (3) the characters are all from Σ except for one element of Q. We describe how to interpret it by an example.

If the configuration is

$$\$aabba###bb(a,q)abba###ab$$

It means that

- The \$ has no meaning but we will need it as a separator later.
- The tape has
$$aabba###bbaabba###ab$$
on it. All other symbols to the right are blank; however, the head will never go there.
- The state is q.
- The head is at the space which in the configuration we have (a, q).
- Note that if we know the configuration then we know the possibilities for the next one (recall that M is NONDET). Also note that the next one configuration and this one will only differ in a at most three consecutive characters that are close to (and including) the state.

We need to know when a string z represents an accepting computation of $M(x)$.

Def 3.2 Let M be a NONDET TM that runs in space $S(n)$. Let x be of length n. Let Σ be the alphabet to the TM. Let \$ be a symbol that is not in Σ. We will use it as a separator. Let Q be the states. Let $\Gamma = \Sigma \cup \Sigma \times Q \cup \{\$\}$. The function $STEP : \Gamma \times \Gamma \times \Gamma \to 2^{\tau_1\tau_2\tau_3}$ is defined as follows.

- If $\sigma_1\sigma_2\sigma_3 \in (\Gamma - \$)(\Gamma - \$)(\Gamma - \$)$ then $\tau_1\tau_2\tau_3 \in STEP(\sigma_1\sigma_2\sigma_3)$ if when $\sigma_1\sigma_2\sigma_3$ are consecutive symbols in a configuration of a computation for M, it is possible for $\tau_1\tau_2\tau_3$ to be consecutive symbols in the next configuration. Note that if none of $\sigma_1, \sigma_2\sigma_3$ contain a state then $\sigma_1\sigma_2\sigma_3 \in STEP(\sigma_1\sigma_2\sigma_3)$. There may be other strings also since we don't know where the head is— it could be just to the left and have a move-right command. In short- anything NOT ruled out is allowed.
- If $\sigma_1\sigma_2\sigma_3$ contains a \$ then think of it as representing the end of one config and the beginning of another. STEP doesn't change that much, its still the set of successors (though now in parts of two configurations) that are not ruled out.

Def 3.3 Let M be a NONDET TM that runs in space $S(n)$. Let x be of length n. Let Σ be the alphabet to the TM. Let \$ be a symbol that is not in Σ. We will use it as a separator. Let Q be the states. Let $\Gamma = \Sigma \cup \Sigma \times Q \cup \{\$\}$ A string $z \in \Gamma^*$ *represents an accepting computation* $M(x)$ if the following hold:

1. $z \in (\$(\Gamma - \$)^{S(n)})^*$. Hence we can write $z = \$C_1\$C_2 \cdots \$C_m$ where each C_i is exactly $S(n)$ long.
2. For every $\sigma_1\sigma_2\sigma_3$ if $z = \Gamma^*\sigma_1\sigma_2\sigma_3\Gamma^{S(n)-2}\tau_1\tau_2\tau_3$ then $\tau_1\tau_2\tau_3 \in STEP(\sigma_1\sigma_2\sigma_3)$.

We will be more concerned with when a string z is NOT an accepting computation.

Theorem 3.4 *TRE is $NEXPSPACE$-complete and hence NOT in $PSPACE$, and hence NOT in P.*

Proof: By Theorem 1.5 $TRE \in NEXPSPACE$. We now need to show that TRE is $NEXPSPACE$-hard.

Let $A \in NEXPSPACE$. We show $A \leq TRE$. Let M be the $NEXPSPACE$ machine for A. Let c be such that $S(n) = 2^{n^c}$ bounds the amount of space that M uses on a string of length n. Let $\Gamma = \Sigma \cup \Sigma \times Q \cup \{\$\}$.

Let $x = x_1x_2 \cdots x_n$. We build a t-reg exp α such that

$$x \in A \text{ iff } L(\alpha) \neq \Gamma^*$$

We define α so that if $M(x)$ does not accept then $L(\alpha) = \Gamma^*$, and if $M(x)$ does accept then $L(\alpha) \neq \Gamma^*$.

α is the union of the following which we present with explanation.

1.

$$\begin{aligned}(\Gamma - \$)\Gamma^* \cup \quad & \$(\Gamma - \{x_1\})\Gamma^* \\ \cup \quad & \$x_1(\Gamma - \{x_2\})\Gamma^* \\ \cup \quad & \cdots \\ \cup \quad & \$x_1x_2\cdots x_{n-2}(\Gamma - \{x_{n-1}\})\Gamma^* \\ \cup \quad & \$x_1x_2\cdots x_{n-1}(\Gamma - (q_{start}, x_n))\Gamma^*\end{aligned}$$

This is the set of all strings which do NOT begin $\$x_1x_2\cdots x_{n-1}(q_{start}, x_n)$.

This t-reg expression is of length $O(n^2)$.

2.

$$\$x_1x_2\cdots x_{n-1}(q_{start}, x_n)(\Gamma - \$)^*(\Gamma - \{\#, \$\}(\Gamma - \{\$\})^*$$

This is the set of strings that DO begin $\$x_1x_2\cdots x_{n-1}(q_{start}, x_n)$ but then have a NON-BLANK sign before the next \$.

This t-reg exp is of length $O(n)$.

3. Take the union over all $\sigma_1, \sigma_2, \sigma_3, \tau_1, \tau_2, \tau_3 \in \Gamma$ such that $\tau_1\tau_2\tau_3 \notin STEP(\sigma_1\sigma_2\sigma_3)$ of the following t-reg exps:

$$\Gamma^*\sigma_1\sigma_2\sigma_3\Gamma^{S(n)-3}\tau_1\tau_2\tau_3\Gamma^*.$$

This is the set of strings that DO NOT represent a computation since you do not have that C_{i+1} follows from C_i. This also includes strings where the \$ are not $S(n)$ apart.

Note that we write $S(n)$ in binary (or anything except unary). Hence this t-reg expression is of length $O(\log(S(n))$.

4. $\Gamma^*q_{rej}\Gamma^*$. These cannot possibly be strings that represent an ACCEPTING computation

This t-reg exp is of length $O(1)$.

If $M(x)$ accepts then the accepting computation coded as a sequence of configurations is NOT in $L(\alpha)$.

If $M(x)$ rejects then every string is in $L(\alpha)$.

Note that the t-reg expression can be produced in poly time and is of length $O(n^2 + \log(S(n)))$.

Since $S(n) = 2^{O(n^c)}$ this is $O(n^{\max\{2,c\}})$ which is polynomial.

∎

4 A $PSPACE$-complete Problem for Free

Let

$$RE = \{\alpha \text{ is a reg exp} \mid L(\alpha) \neq \Sigma^*\}.$$

Theorem 4.1 *RE is $PSPACE$-complete.*

Proof:

$RE \in PSPACE$: This is similar to how we showed $TRE \in EXPSPACE$.

RE is $PSPACE$-hard: This is similar to how we showed TRE is $EXPSPACE$-complete. The only difference: when we write $\Gamma^{S(n)}$ we really do write out $S(n)$'s Γ. Note that $S(n)$ is a polynomial, so we can do this. ∎

5 Variants of TRE for higher classes

In the proof that TRE is $EXPSPACE$-hard the only time we used the 't' in t-reg exp was when we wrote

$$\Gamma^{S(n)}.$$

The key was that we wrote this using binary so it took $O(\log S(n))$ space.

What is instead of using binary we used an even more efficient coding system. AH- but don't I NEED $\log n$ bits to represent all numbers of length n? YES. But we don't' need to represent ALL such numbers.

Consider $NSPACE(2^{2^{n^{O(1)}}})$. Let $S(n) = 2^{2^{n^{O(1)}}}$. We want a variant of reg expressions that can represent $\Gamma^{S(n)}$ compactly.

The definition will use $\alpha^{2^{n^c}}$ instead of α^n.

We leave the details to the reader; however, note that we can find a set that is complete for $NSPACE(2^{2^{n^{O(1)}}})$.

One can do higher as well by adding more stacks of 2's.

References

[1] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. of the 13th Annual IEEE Sym. on Switching and Automata Theory*, pages 125–129, 1972.