CSC 2401, Introduction to computational complexity
Lecture #21, Wednesday, November 21^{st}, 2001
Lecturer: Allan Borodin
Scribe: Anastasia Bezerianos

1 TRE, $\overline{TRE}$ Continued

In our previous lecture we defined the natural language TRE (Total Regular Expression) which is complete (wrt $\leq_{logspace}$) for $PSPACE$:

$$TRE = \{R \mid \mathcal{L}(R) = \Gamma_1^*\}$$

and consequently we have

$$\overline{TRE} = \{R \mid \mathcal{L}(R) \neq \Gamma_1^*\}$$

Theorem 1 *$\overline{TRE}$ is complete (wrt $\leq_{logspace}$) for $PSPACE$*

Sketch:
To prove that $\overline{TRE}$ is complete (wrt $\leq_{logspace}$) for $PSPACE$, we will show that:

1. $\overline{TRE} \in NSPACE(n)$
2. $L = \mathcal{L}(M)$ and M is a non-deterministic space $s(n)$ TM, then $\forall x$ transform (in logspace) x to $R_{M,x} = R_x$ such that:
   (a) $x \in L = \mathcal{L}(M)$ iff $\mathcal{L}(R_{M,x}) \neq \Gamma_1^*$
   (b) $|R_{M,x}| = C_M(\ |x|^2 + s(n)\ )$: transformation computable in log space with respect to input length $n = |x|$.

Corollary: $L \leq_{logspace} \overline{TRE}$.

Claim: From the above, $\overline{TRE}$ requires $\sim$ non-deterministic linear space. We'll use the (unproven) fact that there exists a hierarchy theorem for non-deterministic space analogous to that of deterministic space. Specifically, for constructible polynomial function $s(n)$, there are languages $L \in NSPACE(s(n)) - NSPACE(o(s(n))$. (A deterministic linear space lower bound follows immediately from the determnninistic space hierarchy theorem.) Thus for space $s(n) \geq n^2$, we can transform any $L \in NSPACE(s)$ into $\overline{TRE}$ (without actually outputting the regular expression) and then if $\overline{TRE}$ could be solved in sublinear space, it would allow us to solve L in space $o(s)$.

Proof:

1. $\overline{TRE} \in NSPACE(n)$

   Given the regular expression R it is known that in linear space we can construct a non-deterministic finite automaton $\mathcal{A}$, such that $\mathcal{L}(R) = \mathcal{L}(\mathcal{A})$ and # of states in $\mathcal{A} \;=\; \mathcal{O}(|R|)$, by induction on how R is constructed.

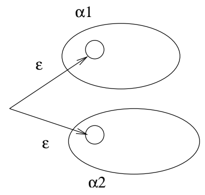

Figure 1: $R_1 \oplus R_2$

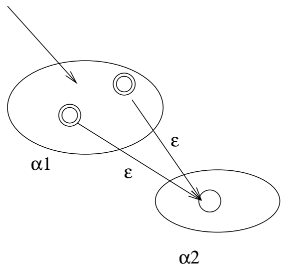

Figure 2: $R_1 \odot R_2$

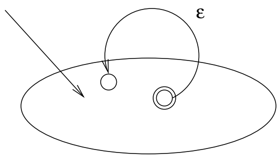

Figure 3: R^*

At each step R is one of the following:

- $R_1 \oplus R_2$: Guess whether to go to $\alpha_1 \in \mathcal{A}_1$ or $\alpha_2 \in \mathcal{A}_2$: linear type transformation (Fig 1).
- $R_1 \odot R_2$: Non deterministically go to the initial state of $\mathcal{A}_2$ (Fig 2).
- R^* : Jump back. Non-deterministic guess (Fig 3).

Say we have a non-deterministic machine $\mathcal{A}$ with k states that we encode in length $n = \mathcal{O}(k \log k)$. We can verify that $\mathcal{L}(\mathcal{A}) \neq \Gamma_1^*$ in non-deterministic space k.
Idea: Keep a length k work tape representing which states in $\mathcal{A}$ we can reach after processing some given string in Γ_1^*.

Guess some $y \in \Gamma_1^* \;:\; y \notin \mathcal{L}(\mathcal{A})$. As we go along we guess some symbol $\gamma \in \Gamma_1^*$. We keep a vector of length k to know which states we can be in. There is a finite number of configurations of the TM and the search has to stop at some point, when we have exceeded the number of possible configurations for the next state. If we exceed that number we are visiting already visited states (loop). If a reachable subset of states does not contain a final state in the automaton, then $\mathcal{L}(\mathcal{A}) \neq \Gamma_1^*$, because the string y in Γ_1^* does not lead to an accepting state of the automaton.

Note: TRE is also in $NSPACE(n)$, since $NSPACE(n)$ is closed under complement.

2. Transformation/Encode Computations:
   Let $x \in \mathcal{L}(m)$, then $\exists$ an accepting computation.

   Here is how we encode an accepting computation (we assume 1 tape, both working and input):
   Alphabets:
   $\Gamma_0 = (\Gamma \cup (\Gamma \times Q))$
   $\Gamma_1 = (\Gamma \cup (\Gamma \times Q) \cup \{\#\})$
   We encode a sequence of configurations $\alpha_0, .., \alpha_t$ in a string $y = \#\alpha_0\#\alpha_1\#...\#\alpha_t\#$. Each $|\alpha_i| = s(n)$ and the symbol $\#$ is not in Γ_0.

   Define $R_{m,x}$: $R_{m,x}$ is going to describe the set of all strings in Γ_1 except a string describing an accepting computation, if there is one. We will prove that the following construction of $R_{m,x}$ indeed includes all possible strings in Γ_1 except accepting computations.

   $R = R_1 \oplus R_2 \oplus R_3 \oplus R_4$, where,

   R_1: describes strings that don't have the correct format (don't start or finish with $\#$, configurations are of incorrect length or not of the correct form, or the null string λ).

   R_2: describes strings that don't start in the initial configuration α_0.

   R_3: describes strings that don't end in an accepting configuration $\alpha_{c^{s(n)}}$.

   R_4: describes strings that for some i, $\alpha_i \nvdash \alpha_{i+1}$

   Formally:

   R_1: $R_1 = \{\lambda\} \oplus (\Gamma_0\Gamma_1^*) \oplus (\Gamma_1^*\Gamma_0) \oplus (\Gamma_1^*\#(R_{1,1} \oplus R_{1,2})\#\Gamma_1^*)$ where
   $(\Gamma_0\Gamma_1^*)$: string doesn't start with symbol $\#$.
   $(\Gamma_1^*\Gamma_0)$: string doesn't end with symbol $\#$.
   $R_{1,1}$: string not of correct length $s(n)$, it is either too long or too short.
   - If string is too long then it can expressed as $\Gamma_1^{s(n)+1}\Gamma_1^*$.
   - If string is too short then it can be expressed as $(\Gamma_1 \oplus \{\lambda\}) \odot (\Gamma_1 \oplus \{\lambda\}) \odot \ldots (\Gamma_1 \oplus \{\lambda\})$ There are $s(n) - 1$ occurences of $(\Gamma_1 \oplus \{\lambda\})$ in this concatenation so that we get all strings of length strictly less than $s(n)$.

   $R_{1,2}$: string has either no symbol in $(\Gamma \times Q)$ or more than one symbol in $(\Gamma \times Q)$.

   R_2: $R_2 = (\bigoplus_j R_{2,j}) \odot \Gamma^{s(n)-n}$, $1 \leq j \leq n$ where $R_{2,j}$ states that the j^{th} symbol is wrong. For input $x = x_1x_2...x_n$,
   $R_{2,1}$: $\#(\Gamma_1 - (x_1, q_0))\Gamma_1^*$
   $R_{2,j}$: $\#\Gamma_1^{j-1}(\Gamma_1 - \{x_j\})\Gamma_1^*$

R_3: $R_3 = \#\Gamma_1^*(\Gamma \oplus \Gamma \times (Q - F))\#$

R_4: What defines $\alpha_i \vdash \alpha_{i+1}$?
There are two cases for $\alpha_i \vdash \alpha_{i+1}$:
Assume part of a configuration $\alpha_{i,k-1}, \alpha_{i,k}, \alpha_{i,k+1}$, where the "action" is. Let $\alpha_{i,k}$ denote the k^{th} symbol in the i^{th} configuration.

i) If $\alpha_{i,k-1}, \alpha_{i,k}, \alpha_{i,k+1} \notin (\Gamma \times Q)$ then $\alpha_{i+1,k} = \alpha_{i,k}$.

ii) If $\alpha_{i,k} \in (\Gamma \times Q)$ then $\alpha_{i,k-1}, \alpha_{i,k}, \alpha_{i,k+1}$ " $\vdash$ " $\alpha_{i,k-1}, \alpha_{i,k}, \alpha_{i,k+1}$

What can go wrong ($\alpha_i \nvdash \alpha_{i+1}$)?
Let's say that:

$R_{4,1}$: is the case where i) above is violated for some i.

$R_{4,2}$: is the case where ii) above is violated for some i.

i) $\forall a, b, c \in \Gamma\{\notin (\Gamma \times Q)\}$ we can define $\bigoplus_{a,b,c \in \Gamma} R_{a,b,c} = R_{4,1}$ which violates (i).
$R_{a,b,c} = \Gamma_1^* abc \Gamma_1^{s(n)}(\Gamma - \{b\})\Gamma_1^*$

ii) For each pair $(b, q) \in (\Gamma \times Q)$, $a, c \in \Gamma$ we can define regular expression $R_{q,a,b,c}$, $a, b, c \in \Gamma$, $q \in Q$ which violates (ii) by representing strings with two consecutive configurations that do not satisfy (ii).

ex for (ii): If $\delta(b, q) = (d, q', R)$, then

$$R_{q,a,b,c} =$$
$$\Gamma_1^* a{<}b, q{>}c\Gamma_1^{s(n)-1}(\Gamma_1 - \{a\}) \oplus \Gamma_1^{s(n)}(\Gamma_1 - \{d\}) \oplus \Gamma_1^{s(n)+1}(\Gamma_1 - \{(c, q')\})$$

So $R_4 = R_{4,1} \oplus R_{4,2}$ and
$R_{4,2} = (\bigoplus_{a,c \in \Gamma \cup \{\#\}; b \in \Gamma} R_{a,b,c}) \oplus (\bigoplus_{(b,q) \in \Gamma \times Q; a,c \in \Gamma} R_{q,a,b,c})$

2 Next lecture sneak preview

Totality of Extended Regular Expressions (TERE) where extended regular expressions also include an "intersection" operation "$\wedge$".

Claim: $\overline{TERE}$ is complete for $EXPSPACE$.