Egzamin ze złożoności obliczeniowej – teoria. 14.6.2014

Proszę wskazać i wyjaśnić ewentualne błędy w poniższych rozumowaniach.

1. NL = NP. Pokażemy, jak rozwiązać problem 3-SAT przez złożenie obliczeń dwóch maszyn, jednej niedeterministycznej i jednej deterministycznej:

- maszyna M_1 przepisuje swoje wejście (formułę zdaniową) i dopisuje do niej zgadnięte niedeterministycznie wartościowanie zmiennych,
- maszyna M_2 sprawdza, detereministycznie, czy podane jej wartościowanie spełnia podaną formułę.

Każda z tych dwóch maszyn działa w pamięci logarytmicznej. Ich obliczenia możemy złożyć zwykłą metodą, tzn. możemy wykonywać obliczenie maszynę M_2, a kiedy ta maszyna próbuje przeczytać k-ty element swojego wejścia, to za każdym razem uruchomić od nowa maszynę M_1 tak długo, aż wypisze k elementów swojego wyjścia.

W rezultacje otrzymujemy niedeterministyczną maszynę działającą w pamięci logarytmicznej, rozwiązującą problem 3-SAT. Ale jest to problem **NP**-zupełny. Wykazaliśmy więc, że **NL** = **NP**.

Wyjaśnienie. Przypomnijmy, że składanie maszyn *deterministycznych* pracujących w pamięcie logarytmicznej jest poprawne. Nie jest problemem, że *output* maszyny M_1 stanowiący *input* maszyny M_2 nie mieści się w pamięci logarytmicznej. Nie jest potrzebne reprezentowanie go w całości, lecz gdy M_2 chce przeczytać i-ty bit, symulujemy maszynę M_1 tak długo, aż ten i-ty bit wyprodukuje.

Jednak ta standardowa konstrukcja przestaje dobrze działać dla maszyn niedeterministycznych, bo kiedy kolejny raz uruchamiamy maszynę M_1 na tym samym wejściu, możemy otrzymać różne wyniki. W naszym problemie, jeśli jakaś zmienna wystąpi w formule wiele razy, maszyna M_1 może podać dla niej różne wartości. W ten sposób formuła niespełnialna może zostać uznana za spełnialną (*false positive*), co jest niezgodne z definicją akceptacji przez maszynę niedeterministyczną.

Nie jest natomiast problemem, że formuła spełnialna może być czasem uznana za niespełnialną (*false negative*), bo to należy do natury niedeterminizmu. Deterministyczne sprawdzenie w pamięci logarytmicznej, czy dana formuła jest prawdziwa przy zadanym wartościowaniu jest oczywiście możliwe.

2. NP = co-NP. Korzystając z twierdzenia Immermanna-Szelepcsényiego (**NL** = **coNL**) i posługując się techniką *paddingu*, pokażemy że **NP** = **coNP**.

Rozważmy dowolny język $L \in \mathbf{NP}$. Łatwo pokazać, że język

$$K = \{w\#0^{2^{|w|}} \mid w \in L\}$$

należy do klasy **NL**, wobec czego dopełnienie K także należy do **NL**. Ponieważ łatwo sprawdzić w pamięci logarytmicznej czy słowo kończy się odpowiednią liczbą zer, to także język

$$\tilde{K} = \{w\#0^{2^{|w|}} \mid w \notin L\}$$

należy do **NL**. Z tego wynika, że dopełnienie języka L należy do **NP**.

Wyjaśnienie. Zauważmy na wstępie, że rozpoznawanie słów postaci $w\#0^{2^{|w|}}$ jest wykonalne przez deterministyczną maszynę pracującą w pamięci logarytmicznej. Pozostawiamy to jako ćwiczenie z programowania maszyn Turinga. (*Wskazówka*: w pamięci k możemy listować słowa binarne i w ten sposób liczyć do 2^k. Można próbować kolejno $k = 1, 2, \ldots$. Nie należy zaczynać od $k = |w|$, bo ono w „złych" słowach może być już za duże.) Dlatego też, **gdyby** język K był w **NL**, **to** język $\tilde{K}$ też należałby do tej klasy, będąc przecięciem dopełnienia K (należącego do **NL** na mocy Tw. I-S) i zbioru słów dobrej postaci, który jest rozpoznawalny nawet deterministycznie w **L**.

Zobaczmy najpierw na poziomie intuicyjnym, że w podkreślonych fragmentach „coś nie gra", tzn. naiwne argumenty nie działają. Mając niedeterministyczną maszynę M rozpoznającą język L w czasie wielomianowym, możemy ją łatwo zaadaptować do maszyny rozpoznającej K. Jednak M w ogólności

działa w pamięci wielomianowej $\mathcal{O}\left(n^k\right)$ (niekoniecznie $\mathcal{O}(n)$) i wtedy zaadaptowana maszyna działa w pamięci $\mathcal{O}\left(\log^k n\right)$; nie mamy więc konkluzji $L \in \mathbf{NL}$.

Podobnie, mając hipotetyczną maszynę niedeterministyczną rozpoznającą język $\tilde{K}$ w pamięci logarytmicznej, możemy ją zaadaptować do maszyny rozpoznającej język $\bar{L}$ (dopełnienie L) w pamięci liniowej[1], ale jej czas działania nie musi być ograniczony przez wielomian, nie mamy więc konkluzji $\bar{L} \in \mathbf{NP}$.

Powyższe wyjaśnienia pokazują jedynie, że podkreślone implikacje są „problematyczne"[2]. W istocie nie wiemy, czy są one fałszywe; możemy jednak pokazać, że ich obalenie jest niesprzeczne z naszą wiedzą. Przypomnijmy, że jako wniosek z Twierdzenia o Hierarchii Pamięciowej otrzymaliśmy nierówność $\mathbf{P} \neq \mathbf{DSPACE}(n)$, choć nie potrafimy wykluczyć żadnej z inkluzji (*Corollary 2* w notatkach). W analogiczny sposób możemy udowodnić[3] $\mathbf{NP} \neq \mathbf{NSPACE}(n)$. Tu również nie potrafimy wykluczyć żadnej z inkluzji z osobna. W szczególności, jest niesprzeczne z naszą wiedzą

(a) istnieje język $L_1 \in \mathbf{NP} - \mathbf{NSPACE}(n)$,
(b) istnieje język $L_2 \in \mathbf{NSPACE}(n) - \mathbf{NP}$.

Dla języka L_1 odpowiedni język K_1 nie jest w $\mathbf{NL}$, bo w przeciwnym razie otrzymalibyśmy $L_1 \in \mathbf{NSPACE}(n)$ (przez argument analogiczny jak dla $\tilde{K}$ i $\bar{L}$ powyżej), wbrew założeniu. Z kolei język $K_2 = \{w\#0^{2^{|w|}} \mid w \in L_2\}$ jest w $\mathbf{NL}$, a mimo to $L_2 \notin \mathbf{NP}$, co falsyfikuje drugą podkreśloną implikację.

3. NP = PSPACE. Niech

$$\mathrm{NEQ} = \{(M_1, M_2) : M_1, M_2 \text{ są skończonymi niedeterministycznymi automatami i } L(M_1) \neq L(M_2)\}$$

Aby rozwiązać ten problem w NP wystarczy zgadnąć słowo w i sprawdzić (już deterministycznie), że w należy do różnicy symetrycznej $(L(M_1) - L(M_2)) \cup (L(M_2) - L(M_1))$. Ponieważ problem NEQ jest zupełny w PSPACE, otrzymujemy, że NP = PSPACE.

Wyjaśnienie. Problem NEQ jest rzeczywiście PSPACE-zupełny (nawet dla ustalonego $L(M_2) = \Sigma^*$). Dla niedeterministycznego automatu skończonego M i słowa w, sprawdzenie, czy $w \in L(M)$ może być wykonane deterministycznie w czasie[4] $|w| \cdot \mathit{poly}(|M|)$. A zatem sprawdzenie, czy w należy do różnicy symetrycznej może być wykonane deterministycznie w czasie $|w| \cdot \mathit{poly}\,(|M_1| + |M_2|)$. Problem w tym, że długość najkrótszego słowa rozróżniającego M_1 i M_2 może być wykładnicza, więc cały algorytm nie jest $\mathbf{NP}$.

4. NP $\subseteq$ BPP. Prawdopodobieństwo, że losowy graf nie zawiera kliki 4-elementowej, jest dla dostatecznie dużych grafów bardzo małe. Dokładniej, jeśli dla grafu o wierzchołkach $\{1, 2, \ldots, 4n+\alpha\}$ ($\alpha < 4$) interesują nas tylko kliki na blokach $4i+1, 4i+2, 4i+3, 4i+4$, to prawdopodobieństwo, że nie będzie takiej kliki, $\left(1 - \frac{1}{2^6}\right)^m$ dąży do 0, gdy $m \to \infty$.

Oczywiście, graf zawierający klikę 4-elementową nie może być 3-kolorowalny. Rozważmy następujący algorytm probabilistyczny dla problemu 3-kolorowalności. Algorytm zawsze mówi NIE.

Czas działania tego algorytmu jest wielomianowy (a nawet stały równy 1), a jego błąd, zgodnie z tym co powiedzieliśmy powyżej, jest na pewno $\leq \frac{1}{4}$ (od pewnego miejsca, ale z tym możemy sobie łatwo poradzić, wbudowując początkowe przypadki w maszynę Turinga).

Jako że problem 3-kolorowalności jest NP-zupełny, wykazaliśmy, że NP $\subseteq$ BPP, wbrew stwierdzeniu z Notatek, że tak prawdopodobnie nie jest.

---

[1] Zachowanie głowicy na „wirtualnej" części taśmy wejściowej $\#0^{2^{|w|}}$ symulujemy przy pomocy liczników.

[2] Co w zupełności wystarczało do zaliczenia tego punktu „na maxa".

[3] Potrzebna jest tu jakaś forma Twierdzenia o Hierarchii Pamięciowej dla maszyn niedeterministycznych, którą możemy wydedukować z przypadku deterministycznego i Tw. Savitcha.

[4] Nie trzeba determinizować M ani przeglądać wszystkich ścieżek; wystarczy obliczać *zbiory* stanów osiągalne po kolejnych prefiksach słowa w. Jest to elementarny fakt z wykładu JAO, na którym opiera się szybkie wyszukiwanie wzorca zadanego przez wyrażenie regularne.

Wyjaśnienie. Przykład ilustruje częste nieporozumienie co do tego, o jakie prawdopodobieństwo chodzi. Istotnie, nasz „algorytm" w większości przypadków działa dobrze, jeśli patrzymy na wszystkie słowa jak na przestrzeń probabilistyczną. Jednak definicja **BPP** zakłada, że dla każdego słowa wejściowego w prawdopodobieństwo błędu jest ograniczone, podczas gdy tu dla nieskończenie wielu słów wynosi ono 1.