Dowody trudności

Wprowadzenie

Na poprzednich wykładach rozważaliśmy następujący problem: czy drzewo generowane przez dany automat ze stosem rzędu n (równoważnie: schemat rekurencyjny rzędu n lub λY-term o złożoności n) jest akceptowane przez dany automat na drzewach. Podaliśmy n-krotnie wykładniczy algorytm rozwiązujący ten problem. Teraz udowodnimy, że nasz problem jest n-EXPTIME-trudny.

W dowodzie skupimy się na automatach ze stosem wyższego rzędu. Rozważany problem jest n-EXPTIME-trudny nawet dla automatów bez operacji collapse. W dodatku automat na drzewach będzie ustalony i to nawet bardzo prosty. Alfabet składa się z trzech symboli: OR (o randze 2), AND (o randze 2), tt (o randze 0). Akceptowany język L_{tt} to najmniejszy zbiór drzew taki, że:

- $\mathsf{tt} \in L_{\mathsf{tt}}$,
- jeśli $t_1 \in L_{\mathsf{tt}}$ i $t_2 \in L_{\mathsf{tt}}$, to $\mathsf{AND}\, t_1\, t_2 \in L_{\mathsf{tt}}$,
- jeśli $t_1 \in L_{\mathsf{tt}}$ oraz t_2 jest dowolnym drzewem, to $\mathsf{OR}\, t_1\, t_2 \in L_{\mathsf{tt}}$ i $\mathsf{OR}\, t_2\, t_1 \in L_{\mathsf{tt}}$.

Innymi słowy L_{tt} to „formuły prawdziwe" (w drzewie z L_{tt} mogą istnieć ścieżki nieskończone, lecz do prawdy musi się dać dojść w skończenie wielu krokach). Łatwo rozpoznać powyższy język L_{tt} automatem z ko-trywialnym warunkiem akceptacji[1] – wystarczy jeden stan, przypisywany drzewom z L_{tt}; jego przejścia definiujemy zgodnie z powyższą definicją zbioru L_{tt}.

Zanim zaczniemy nasz dowód, uprośćmy jeszcze trochę nasz problem: będziemy rozważać gry osiągalności. W problemie tym rozważamy automat ze stosem rzędu n, który nic nie wczytuje (nic nie wypisuje). Nie wymagamy determinizmu (może być wiele przejść dla ustalonej pary: stan i symbol na stosie). Dodatkowo, stany automatu rozdzielone są między dwóch graczy: Ewę i Adama. Jeśli automat jest w stanie Ewy, to właśnie ona wybiera jedno z dozwolonych przejść; podobnie dla Adama. Dany jest także zbiór stanów docelowych. Pytamy, czy Ewa może tak grać, aby w skończonym czasie dojść do jednego ze stanów docelowych, niezależnie od ruchów Adama.

Łatwo zauważyć, że problem z grą osiągalności łatwo możemy zredukować do wcześniejszego problemu z językiem L_{tt}. Po prostu, jeśli będąc w stanie Ewy mamy dwa możliwe przejścia, to wypisujemy OR, wykonując jedno z możliwych przejść w jednym dziecku, a drugie w drugim (przy liczbie przejść innej niż 2 wypisujemy odpowiednią liczbę symboli OR bezpośrednio po sobie). Dla Adama wypisujemy AND. W stanach docelowych wypisujemy tt. Widzimy, że Ewa może wygrać wtedy i tylko wtedy, gdy wygenerowane drzewo należy do L_{tt}.

Niech $\exp_0(m) = m$ oraz $\exp_{n+1}(m) = 2^{\exp_n(m)}$.

Problem n-EXPTIME-trudny, który wykorzystamy w dowodzie, to następująca „gra układania puzzli" rzędu n. Dany jest zbiór puzzli T z wyróżnionym puzzlami $c_1, c_2, r \in T$ oraz warunki $V, H \subseteq T \times T$ mówiące, które puzzle pasują do siebie pionowo i poziomo. Dane jest także słowo wejściowe $w \in T^*$. Rozważamy grę, w której dwoje graczy wypełnia prostokąt puzzlami ze zbioru T (każdy puzzel może zostać wykorzystany dowolnie wiele razy). Prostokąt ma szerokość $\exp_{n-1}(|w|)$. Gracze wypełniają prostokąt wiersz po wierszu, zaczynając od góry. Dla każdego wiersza najpierw Adam wybiera jeden z puzzli c_1, c_2 i układa w pierwszym polu wiersza. Następnie Ewa uzupełnia resztę wiersza dowolnie wybranymi przez siebie puzzlami. Ewa wygrywa, jeśli w którymś momencie (czyli dla pewnej wysokości prostokąta) uda jej się uzyskać ułożenie spełniające warunki:

- w pierwszym wierszu, na pozycjach od drugiej do $(|w|+1)$-szej, zapisane jest słowo wejściowe w;[2]
- jeśli puzzel v znajduje się gdzieś bezpośrednio pod puzzlem u, to $(u, v) \in V$;
- jeśli puzzel v znajduje się gdzieś bezpośrednio na prawo od puzzla u, to $(u, v) \in H$;
- w ostatnim wierszu użyty został puzzel r.

W problemie pytamy, czy Ewa może wygrać, niezależnie od ruchów Adama.[3]

Nietrudno zobaczyć, że powyższy problem jest n-EXPTIME-trudny. W tym celu musimy wiedzieć, że n-EXPTIME = A-$(n-1)$-EXPSPACE. Ta druga klasa zawiera problemy rozwiązywalne przez alternujące maszyny Turinga w pamięci $(n-1)$-krotnie wykładniczej. Działanie takiej maszyny możemy zakodować za pomocą problemu układania

[1] Choć będziemy pracować z automatem z ko-trywialnym warunkiem akceptacji, automatycznie dostajemy także trudność naszego problemu dla automatów z trywialnym warunkiem akceptacji. Wykona to z faktu, że dopełnienie L_{tt} jest rozpoznawane przez automat z trywialnym warunkiem akceptacji, a klasa n-EXPTIME jest zamknięta na dopełnienie.

[2] Dla $n = 1$ jest tylko $|w|$ pól, więc np. ignorujemy ostatnią literę słowa wejściowego.

[3] Niniejszy problem jest n-EXPTIME trudny nawet wówczas, gdy na wejściu dane jest tylko słowo w, natomiast pozostałe parametry są ustalone. Tutaj nie będziemy jednak z tego korzystać.

puzzli. Wiersze prostokąta będą odpowiadały kolejnym konfiguracjom maszyny Turinga. W definicji alternującej maszyny Turinga również pojawia się gra między Adamem i Ewą, podobnie jak w naszym problemie układania puzzli. Dopracowanie szczegółów kodowania pozostawiamy jako ćwiczenie. Zwróćmy jednak uwagę na jeden szczegół: mówiąc o pamięci $(n-1)$-krotnie wykładniczej, mamy na myśli pamięć o rozmiarze $\exp_{n-1}(p(|w|))$, gdzie w to słowo wejściowe, a p jest pewnym ustalonym wielomianem. Natomiast w problemie układania puzzli mamy do dyspozycji jedynie $\exp_{n-1}(|w'|)$ pól, gdzie w' to słowo wejściowe (czyli tym razem bez wielomianu w wykładniku). Nie jest to jednak problemem: jako w', czyli jako słowo wejściowe do problemu układania puzzli, możemy podać słowo długości $p(|w|)$, zawierające na początku słowo w, a następnie odpowiednie symbole (blanki) – w ten sposób będziemy mogli zmieścić całą potrzebną pamięć w dostępnym prostokącie.

Redukcja dla rzędu 1

Chcemy zredukować grę układania puzzli rzędu 1 do gry osiągalności w automacie ze stosem rzędu 1. Mamy więc dane: zbiór puzzli T, wyróżnione puzzle $c_1, c_2, r \in T$, warunki $V, H \subseteq T \times T$ oraz słowo wejściowe w. Powinniśmy skonstruować automat ze stosem rzędu 1, który opisuje „reguły" gry osiągalności. Stworzony automat powinien być rozmiaru wielomianowego względem rozmiaru instancji gry osiągalności.

Gra osiągalności będzie przebiegała w dwóch fazach. W pierwszej fazie gracze symulują grę układania puzzli, zapisując stworzone ułożenie puzzli na stosie. A więc najpierw Adam odkłada na stos puzzel, który chce położyć na pierwsze pole pierwszego wiersza. Potem Ewa wstawia na stos kolejne puzzle układane w pierwszym wierszu, które muszą pochodzić ze słowa wejściowego. Następnie na stos wstawiany jest separator \$. Dla kolejnych wierszy robimy to samo, przy czym teraz już oczywiście Ewa nie wypisuje słowa wejściowego, tylko dowolnie wybiera wstawiane puzzle. Automat zapewnia, że pomiędzy kolejnymi symbolami \$ zostanie wyłożonych dokładnie $|w|$ puzzli (możemy mieć $|w|$ stanów w automacie). Dodatkowo, w każdym momencie sprawdzamy, czy spełniony jest warunek „poziomy", czyli czy każda para kolejnych puzzli należy do zbioru H. Warunek V na razie ignorujemy. Jeśli w którymś wierszu Ewa wybierze puzzel r, to po zakończeniu wiersza przechodzimy do drugiej fazy.

Faza druga odpowiedzialna jest za sprawdzenie warunku V. Powinno być tak, że jeśli warunek V jest spełniony, to wygra Ewa; natomiast jeśli warunek V nie jest spełniony (czyli w pewnym momencie Ewa oszukała), to wygra Adam. Własność tą można łatwo zapewnić: po prostu pozwalamy Adamowi wskazać miejsce na stosie, gdzie jego zdaniem jest błąd. Może on więc zdjąć ze stosu dowolnie wiele symboli i w pewnym momencie powiedzieć: tu jest błąd. Wówczas zapamiętujemy jaki puzzel jest na szczycie stosu, zdejmujemy ze stosu $|w|+1$ symboli i odczytujemy ze szczytu stosu jaki puzzel sąsiaduje w pionie ze wskazanym. Jeśli wskazana para spełnia warunek V, to wygrywa Ewa, wpp. wygrywa Adam.

Widzimy, że jeśli Ewa jest w stanie wygrać w grze układania puzzli, to w pierwszej fazie gry osiągalności może po prostu symulować tamtą rozgrywkę i w drugiej fazie Adam nie będzie w stanie wykryć błędu, Ewa wygra. Z kolei jeśli Adam wygrywa w grze układania puzzli, a Ewa mimo to doprowadziła do drugiej fazy w grze osiągalności (przy optymalnej grze Adama), to w którymś momencie na pewno musiał zostać dołożony puzzel niepasujący w pionie; Adam może więc wskazać błąd w drugiej fazie i tym sposobem wygrać.

Redukcja dla rzędu 2

Tym razem chcemy zredukować grę układania puzzli rzędu 2, która odbywa się na planszy o większej szerokości, konkretnie $2^{|w|}$. Jeśli wprost zastosowalibyśmy poprzednią konstrukcję do tego przypadku, dostalibyśmy automat rozmiaru wykładniczego. Niezbędne jest bowiem liczenie, jak dużo puzzli zostało już położone w danym wierszu, a także liczenie w fazie drugiej ile symboli zdjęliśmy ze stosu, gdy chcemy znaleźć tę samą kolumnę w kolejnym wierszu. Skoro jednak liczenie to nie może być realizowane przez automat, powierzymy je graczom.

Rozszerzymy w tym celu alfabet stosowy o dodatkowe symbole 0 i 1. W fazie pierwszej, przed wpisaniem każdego z puzzli na stos przez któregoś z graczy, Ewa wpisuje na stos licznik długości $|w|$, korzystając z cyfr 0 i 1. Jest przy tym sprawdzane przez automat, że pierwszy licznik w wierszu zawiera same zera, a ostatni przed dolarem – same jedynki. W „idealnej sytuacji" Ewa powinna przed kolejnymi puzzlami wypisywać kolejne wartości licznika, od 0 do $2^{|w|}-1$ (co zapewniłoby, że ułożony prostokąt ma szerokość $2^{|w|}$); nie jest to jednak na razie sprawdzane przez automat. W fazie pierwszej, tak jak poprzednio, korzystamy tylko z jednego stosu rzędu 1.

Po fazie pierwszej następuje faza druga, w której Adam ma prawo wykazania (jednego) błędu w wypisanym ułożeniu. Błędy mogą być tym razem dwóch rodzajów.

- Być może liczniki przypisane kolejnym puzzlom nie są kolejnymi liczbami naturalnymi. Zobaczmy, że zapisy kolejnych liczb binarnych są postaci $w01^k$ i $w10^k$ dla jakichś w i k. Zatem Adam po pierwsze wskazuje (wykonując pewną liczbę operacji pop_1), który licznik jego zdaniem nie jest większy o jeden od poprzedniego. Następnie może policzyć liczbę zer po ostatniej jedynce (ta liczba jest nie większa niż $|w|$, więc automat może ją zapamiętać) i

sprawdzić, że jest taka sama, jak liczba jedynek po ostatnim zerze w poprzednim liczniku. Alternatywnie, może wybrać jedną z pozycji przed ostatnią jedynką (znowu, automat może zapamiętać, która to pozycja) i sprawdzić, że w poprzednim liczniku na tej pozycji jest inna cyfra.

- Być może warunek V nie jest spełniony: puzzle w kolejnych wierszach nie pasują do siebie. Aby to wykazać, Adam wykonuje pewną liczbę operacji pop_1, odsłaniając puzzel, który jego zdaniem nie pasuje do puzzla w poprzednim wierszu. Cały stos jest teraz kopiowany operacją push_2, a następnie zadaniem Ewy jest wskazanie puzzla znajdującego się o jedno pole w górę. Ewa wykonuje w tym celu pewną liczbę operacji pop_1, przechodząc przy tym przez dokładnie jeden symbol \$. Odsłania w ten sposób pewien puzzel w poprzednim wierszu (jednak nie wiemy na razie, czy w odpowiedniej kolumnie). Jeśli wskazane puzzle nie pasują do siebie w pionie, to oczywiście jest błąd, Adam wygrywa. Jeśli puzzle pasują, to Adam ma jeszcze możliwość wykazania, że wartości licznika pod wskazanymi puzzlami różnią się. W tym celu wybiera dowolną pozycję licznika znajdującego się na szczycie stosu (zapamiętujemy w stanie, która to pozycja i jaka cyfra się na niej znajduje). Wykonujemy teraz pop_2, odsłaniając wcześniej wskazany puzzel i licznik znajdujący się pod nim, przechodzimy w nim na tę samą pozycję i sprawdzamy, czy cyfra się zgadza. Jeśli się zgadza, to wygrywa Ewa, wpp. Adam.

Podobnie jak poprzednio, powinno być jasne, że Ewa może wygrać w grze układania puzzli wtedy i tylko wtedy, gdy może wygrać w skonstruowanej grze osiągalności.

Redukcja dla rzędu 3 i wyższych

Naszkicujmy jeszcze rozwiązanie dla rzędu 3. Tym razem szerokość prostokąta to $\exp_2(|w|) = 2^{2^{|w|}}$. Widzimy, że długość licznika, który byłby w stanie pomieścić taką liczbę, to $2^{|w|}$. Musimy więc także numerować pozycje liczników. W tym celu mamy dwa rodzaje cyfr: $0_1, 1_1$ i $0_2, 1_2$; nazwijmy je cyframi rzędu jeden i cyframi rzędu dwa. Główny licznik korzysta z cyfr rzędu 2. Jednak przed każdą cyfrą rzędu 2 mamy $|w|$ cyfr rzędu jeden. Docelowo licznik rzędu 1 numeruje pozycje licznika rzędu 2, zapewniając, że jest ich dokładnie $2^{|w|}$.

W pierwszej fazie to znowu Ewa konstruuje te liczniki i zapisuje na stosie pod każdym puzzlem (jak poprzednio, używamy tylko jednego stosu rzędu 1). W drugiej fazie Adam ma możliwość wskazania, że liczniki zostały przez Ewę zapisane nieprawidłowo.

- Adam może wskazać, że kolejne liczniki rzędu 1 nie zawierają kolejnych liczb; odbywa się to jak poprzednio.
- Adam może wskazać, że kolejne liczniki rzędu 2 nie zawierają kolejnych liczb. Jest to bardzo podobne do sprawdzania w poprzedniej konstrukcji, czy kolejne wiersze puzzli pasują do siebie zgodnie z warunkiem V.
- Ponadto, Adam może wskazać, że puzzle znajdujące się nad sobą nie spełniają warunku V. W tym celu wybiera jeden puzzel i robi push_2, po czym Ewa wskazuje puzzel w poprzednim wierszu, który jej zdaniem znajduje się w tej samej kolumnie. Adam może teraz wskazać jedną z pozycji licznika rzędu 2, która jego zdaniem nie jest taka sama, jak w liczniku pod wskazanym przez niego puzzlem. Robimy po tym push_3 i pop_2, odsłaniając pierwszy puzzel, w którym Ewa może wskazać tę samą pozycję licznika rzędu 2. Nadal nie wiemy, czy obaj gracze wskazali te same pozycje licznika rzędu 2, musimy porównać liczniki rzędu 1. Pozwalamy więc Adamowi wskazać jedną pozycję licznika rzędu 1, na której, jego zdaniem, liczniki różnią się między sobą. Teraz już możemy zapamiętać w stanie automatu, która to pozycja. Wykonujemy więc pop_3, przechodząc z powrotem do drugiego z liczników rzędu 1 i porównujemy cyfrę na tej zapamiętanej pozycji.

Powyższa konstrukcja w naturalny sposób uogólnia się na wyższe rzędy. Aby symulować grę układania puzzli rzędu n, potrzebujemy liczyć do $\exp_{n-1}(|w|)$, co można zrealizować za pomocą licznika rzędu $n-1$, którego pozycje są numerowane licznikami rzędu $n-2$, itd. Do sprawdzania, czy puzzle w kolejnych wierszach spełniają warunek V potrzebujemy n-krotnie przechodzić między dwoma miejscami na stosie rzędu 1 stworzonym w pierwszej fazie; jest to możliwe w automacie ze stosem rzędu n.

Osiągalność

Na wykładzie 7 pokazaliśmy, że przedstawiony powyżej problem można rozwiązać w klasie $(n-1)$-EXPTIME w szczególnym przypadku, gdy rozważany automat na drzewach z trywialnym warunkiem akceptacji jest deterministyczny z góry w dół. Pokażemy teraz odpowiadające temu ograniczenie dolne. Tak jak wcześniej, trudność udowodnimy dla ustalonego języka. Będzie to język podobny do L_{tt}, ale rozważany nad alfabetem bez symbolu AND. Zatem alfabet zawiera tylko symbole OR i tt, natomiast język L'_{tt} zawiera te drzewa, w których występuje symbol tt. Widzimy, że dopełnienie tego języka (w drzewie są same symbole OR) można rozpoznać deterministycznym (z góry w dół) automatem z trywialnym warunkiem akceptacji, o jednym stanie.

Nasz problem to innymi słowy problem osiągalności: czy niedeterministyczny automat ze stosem rzędu n może dojść to ustalonego stanu (w którym wypisałby tt). Jest to też to samo, co problem niepustości języka rozpoznawanego przez taki automat (czy mogę dojść do stanu akceptującego). My skupimy się na automacie, który nic nie wczytuje (nie wypisuje), a w którym tylko chcemy dojść do ustalonego stanu.

Aby udowodnić $(n-1)$-EXPTIME-trudność powyższego problemu dla automatów rzędu n (gdzie $n \geq 2$), zredukujemy do niego wcześniej rozważany problem gier osiągalności dla automatów rzędu $n-1$. Redukcja jest bardzo prosta: nasz nowy automat A' symuluje poprzedni automat A. Jeśli w A jesteśmy w stanie, w którym decyduje Ewa, to po prostu decydujemy za nią. Jeśli w A jesteśmy w stanie Adama i są np. dwa możliwe ruchy, to powinniśmy zapewnić, że po wykonaniu każdego z nich Ewa jest w stanie wygrać. Wykonujemy zatem pierwszy ruch na oryginalnym stosie rzędu $n-1$, a drugi na jego kopii stworzonej operacją push_n (dokładniej robimy tak: odkładamy opis pierwszego ruchu na stos, robimy push_n, zdejmujemy opis ruchu, wykonujemy drugi ruch; gdy później zobaczymy opis ruchu na szczycie stosu, to go wykonujemy). Z kolei jeśli uda się dojść do stanu docelowego, to wykonujemy pop_n. Akceptujemy, gdy stos rzędu n będzie pusty (dokładniej, gdy zobaczymy odpowiedni znacznik położony na początku). W ten sposób, zamiast rozgałęziać nasze obliczenie w stanach Adama, symulujemy wszystkie możliwości, odkładając na stos rzędu n rzeczy do zrobienia później (pamiętajmy, że symulujemy automat rzędu $n-1$). Powinno być jasne, że można dojść do stanu akceptującego w A' wtedy i tylko wtedy, gdy w oryginalnej grze Ewa może dojść do stanu docelowego przy wszystkich możliwych ruchach Adama. Korzystamy tutaj z faktu, że drzewo (o skończonym rozgałęzieniu), w którym każda gałąź jest skończona, ma skończenie wiele wierzchołków.