Домашни по УИС:

Мартин Георгиев

Коледен едишън

Задача 1. За произволно свойство S, $Code(S)$ е полуразрешим тогава и само тогава, когато:

(1) Ако L е полуразрешим език в S и $L \subseteq L'$, за някакъв полуразрешим език L', то L' е в S

(2) Ако L е безкраен полуразрешим език в S, то съществува краен подезик L' на L, който е в S.

(3) Множеството от крайни езици(кодирани като стрингове) в S е полуразрешимо.[†]

Решение:
($\Rightarrow$) Първо ще докажем, че полуразрешимостта на $Code(S)$ влече (1). Това е същото като да докажем, че ако L е полуразрешим език в S и $L \subseteq L'$, за някакъв полуразрешим език L', и L' не е в S, то $Code(S)$ не е полуразрешим (контрапозиция). Нека си фиксираме полуразрешими езици L и L', такива че L е в S, $L \subseteq L'$ и L' не е в S. Ще направим свеждането $L_{diag} \leq Code(S)$. За целта ще изложим алгоритъм, реализиращ тотална функция f, таква че:

$$\mathcal{L}(M_{f(\omega)}) = \begin{cases} L & \omega \in L_{diag} \ (\iff w \notin \mathcal{L}(M_\omega)) \\ L' & \omega \notin L_{diag} \ (\iff w \in \mathcal{L}(M_\omega)) \end{cases}$$

Описание на работата на $M_{f(\omega)}$ върху входна дума α:

- $M_{f(\omega)}$ симулира едновременно $M_\omega(\omega)$ и $M_L(\alpha)$.
- Ако M_L завърши първа с q_{accept}, $M_{f(\omega)}$ завършва с q_{accept}.
- Иначе, ако M_ω завърши първа с q_{accept}, $M_{f(\omega)}$ симулира $M_{L'}(\alpha)$ и в случай, че симулацията завърши, завършва със същия резултат.

Ясно е, че ако ω е в L_{diag}, то M_ω никога няма да завърши, съответно $M_{f(\omega)}$ ще приеме точно думите, които M_L приема, и $\mathcal{L}(M_{f(\omega)})$ ще е точно L. За случая, в който ω не е в L_{diag}, разглеждаме два възможности. Ако α е от L', то тя или ще бъде приета от M_L (в случая, когато $M_L(\alpha)$ завършва успешно преди $M_\omega(\omega)$) или ще бъде приета от $M_{L'}$ (в случая, когато $M_\omega(\omega)$ завършва успешно преди $M_L(\alpha)$). Ако пък α не е от L', то тя няма да бъде приета от M_L, съответно $M_\omega(\omega)$ ще завърши първа с q_{accept} и симулацията на $M_{L'}$ няма да приеме α. Така, когато ω не е в L_{diag}, $\mathcal{L}(M_{f(\omega)})$ ще е точно L'. Оттук директно можем да заключим, че:

$$\omega \in L_{diag} \iff \mathcal{L}(M_{f(\omega)}) \in S \iff f(\omega) \in Code(S)$$

Щом L_{diag} се свежда до $Code(S)$, то $Code(S)$ е "поне толквоа труден" колкото L_{diag}, тоест $Code(S)$ няма как да е полуразрешим. □

Сега ще докажем, че полуразрешимостта на $Code(S)$ влече (2). Това е същото като да докажем, че ако L е безкраен полуразрешим език в S и никой краен подезик L' на L не е в S, то $Code(S)$ не е полуразрешим (контрапозиция). Нека си фиксираме безкраен полуразрешим език L в S, който няма крайни подезици в S. Както в предишното доказателство, ще направим свеждането $L_{diag} \leq Code(S)$. За целта ще изложим алгоритъм, реализиращ тотална функция f, такава че:

$$\mathcal{L}(M_{f(\omega)}) = \begin{cases} L & \omega \in L_{diag} \ (\iff w \notin \mathcal{L}(M_\omega)) \\ L' & \omega \notin L_{diag} \ (\iff w \in \mathcal{L}(M_\omega)) \end{cases}$$

където L' е някакъв краен подезик на L (все още не знаем какъв).
Описание на работата на $M_{f(\omega)}$ върху входна дума α:

[†] В учебника на Hopcroft и Ullman се говори за *enumerable* множества, тоест множества, които могат да бъдат генерирани от машина на Тюринг (чрез изброяване на елементите им на изходна лента). Доколкото разбирам, един език е полуразрешим точно тогава, когато е *enumerable*, затова съм формулирал твърдението с полуразрешимост.

- $M_{f(\omega)}$ симулира $M_\omega(\omega)$ за $|\alpha|$ на брой стъпки.
- Ако симулацията завърши с q_{accept}, $M_{f(\omega)}$ отхвърля α.
- Иначе, $M_{f(\omega)}$ пуска нова симулация на $M_L(\alpha)$ и, в случай, че тя завърши, $M_{f(\omega)}$ завършва със същия резултат.

Ясно е, че ако ω е в L_{diag}, то M_ω никога няма да завърши, съответно $M_{f(\omega)}$ ще приеме точно думите, които M_L приема, и $\mathcal{L}(M_{f(\omega)})$ ще е точно L. В случая, в който ω не е в L_{diag}, $M_\omega(\omega)$ завършва в q_{accept} за някакъв брой стъпки k. Тогава $M_{f(\omega)}$ няма да приеме никоя дума с дължина поне k, тоест ще приеме само краен брой думи. Нещо повече, тези думи ще са от L тъй като $M_{f(\omega)}(\alpha)$, завършва успешно само когато $M_L(\alpha)$ завършва успешно. Така, когато ω не е в L_{diag}, $\mathcal{L}(M_{f(\omega)})$ ще е точно L', за някакъв краен подезик L' на L. Оттук директно можем да заключим, че:

$$\omega \in L_{diag} \iff \mathcal{L}(M_{f(\omega)}) \in S \iff f(\omega) \in Code(S)$$

Щом L_{diag} се свежда до $Code(S)$, то $Code(S)$ е "поне толквоа труден" колкото L_{diag}, тоест $Code(S)$ няма как да е полуразрешим. □

Остава да докажем, че полуразрешимостта на $Code(S)$ влече (3). За целта е достатъчно да построим машина на Тюринг M, такава че $\mathcal{L}(M)$ е множеството от крайни езици в S. Нека $M_{Code(S)}$ е машина на Тюринг, която разпознава $Code(S)$.

Описание на работата на M върху входна дума α:

- За всяка от думите от езика с код α, M "строи" код на машина на Тюринг, разпознаваща само тази дума.
- M обединява всички машини в нова машина M_L с език обединението на езиците на машините.
- M симулира $M_{Code(S)}$ върху кода на M_L и в случай, че $M_{Code(S)}$ завърши, M завършва със същия резултат.

Ясно е, че M завършва успешно върху входна дума α тогава и само тогава, когато крайният език L, кодиран в думата α е в S, тъй като генерирането на машината M_L за езика L отнема крайно време и $M_{Code(S)}$ завършва успешно върху M_L точно тогава, когато $\mathcal{L}(M_L) = L$ е в S. □

($\Leftarrow$) В обратната посока, нека приемем, че имаме (1), (2) и (3). Трябва да докажем, че $Code(S)$ е полуразрешим. За целта е достатъчно да построим машина на Тюринг $M_{Code(S)}$, такава че $\mathcal{L}(M_{Code(S)}) = Code(S)$. Нека M_{finite} е машина за множествтото от крайни езици в S и нека $\omega_1, \omega_2, \ldots$ е произвлно изброяване на думите от Σ^*.

Описание на работата на $M_{Code(S)}$ върху входна дума α:

- $M_{Code(S)}$ паралелно[†] симулира $M_{finite}(\omega_1), M_{finite}(\omega_2), \ldots$ (като да кажем ги редува[†] на някакъв брой стъпки)
- Ако някога някоя от симулациите (да кажем за дума ω_i) завърши успешно, $M_{Code(S)}$ създава нови паралелни симулации(които продължава да редува с останалите) на M_α за всяка една от думите в ω_i(да не забравяме, че ω_i кодира език). Ако всяка една от тези нови симулации завърши успешно, $M_{Code(S)}$ завършва с q_{accept}.

Ясно е, че ако α е в $Code(S)$, то езикът $L \coloneqq \mathcal{L}(M_\alpha)$ ще е в S, съответно от (2) ще съществува краен подезик L' на L в S, който се кодира от крайна дума $\omega_{L'}$. По построение машината M все някога ще завърши успешно симулация на M_{finite} върху думата $\omega_{L'}$(тъй като $L' \in S$), след което ще завърши успешно симулация на M_α върху всички думи кодирани от $\omega_{L'}$(тъй като $L' \subseteq L = \mathcal{L}(M_\alpha)$) и ще приключи в приемащо състояние q_{accept}. Ако пък α не е в $Code(S)$, то езикът $L \coloneqq \mathcal{L}(M_\alpha)$ няма да е в S, съответно от контрапозицията на (1) никой краен подезик L' на L няма да е в S и M никога няма да завърши успешно никоя от симулациите на M_α(и съответно самата M няма да завърши успешно), тъй като тези симулации ще са върху думи от езици, които не са подезици на L. Двете разсъжденииия доказват, че $M_{Code(S)}$ разпознава точно езика $Code(S)$. □

Задача 2. (Теорема за линейна компресия) За всяка ограничена по памет от $S(n)$ машина на Тюринг M с k работни ленти и една *read-only* входна лента и за всяка константа $c > 0$ съществува такава ограничена по памет от $cS(n)$ машина на Тюринг M' с една работна лента и една *read-only* входна лента, че $\mathcal{L}(M) = \mathcal{L}(M')$.

Решение:

Броят работни ленти на M няма значение, тъй като винаги можем да трансформираме M в машина N, която използва точно една работна лента. Наистина, тази машина ще работи по-бавно, но броят клетки, които ще използва, ще е точно максимума от използваните клетки на различните работни ленти на M, съответно N също ще е ограничена по памет от $S(n)$. Щом броят ленти няма значение, можем да направим доказателство за машина на Тюринг M с една работна лента. Разделяме задачата на два случая:

[†] Под паралелно не се има предвид на различни ленти, ами по такъв начин, че симулацията на досегашните машини да не бъде замазана.

[†] Пример за такова редуване е 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, . . . Важно е редуването да се връща обратно към симулациите, които вече са стартирани - редуване тип 1, 2, 3, 4, . . . няма да проработи.

1 сл. Ако $c \geq 1$, $M' = M$ ще бъде ограничена по памет от $cS(n)$, тъй като M е ограничена по памет от $S(n)$, а $1 \leq S(n) \leq cS(n)$.

2 сл. Остава да разгледаме случая, когато $0 < c < 1$. Тъй като редицата с общ член $\frac{1}{n}$ клони към 0, винаги можем да намерим естествено число d, такова че $\frac{2}{d} < c$. Е, тогава можем да конструираме машина на Тюринг M', която мислено разбива лентата, която M използва, на равни интервали от по d клетки и кодира всеки от тези интервали в отделна клетка, използвайки азбука $\Gamma' = \Gamma^d \cup \Gamma$ и състояния $Q' = (Q \times I_d) \cup \{q_{accept}, q_{reject}\}$, където всяка буква от Γ^d кодира интервал от d клетки в M, а втората компонента на състоянията от $(Q \times I_d)$ кодира в коя клетка от текущия интервал е главата на M в момента. Началното състояние на M' ще е $q_{start'} = (q_{start}, 0)$, а функцията на преходите трябва да се дефинира така, че на всяка стъпка M' да променя само текущата буква от текущия интервал, да мести главата си само когато главата в M би преминала в нов интервал и да влиза в q_{accept}, q_{reject} само когато е в (q_{accept}, x) и (q_{accept}, x), за някакво x.

Ясно е, че подобна конструкция на M' ще разпознава точно $\mathcal{L}(M)$, използвайки $\lceil \frac{1}{d}S(n) \rceil$ памет. Ако $\lceil \frac{1}{d}S(n) \rceil > 1$, то $\frac{1}{d}S(n) > 1$ и съответно $\lceil \frac{1}{d}S(n) \rceil < \frac{1}{d}S(n) + 1 < \frac{2}{d}S(n)$. От избора ни на d, $\frac{2}{d}S(n) < cS(n)$, откъдето M' ще бъде ограничена по памет и от $cS(n)$. Ако пък $\lceil \frac{1}{d}S(n) \rceil = 1$, то M' тривиално ще бъде ограничена по памет от всяка функция†, включително $cS(n)$.

□

Задача 3. (Теорема за линейно ускорение) Ако един език L се приема от ограничена по време от $T(n)$ k-лентова Тюринг Машина M, $k > 1$ и $n \in o(T(n))$, то за всяко $c > 0$, L се приема от ограничена по време от $cT(n)$ k-лентова Тюринг Машина M'.

Решение:
Както в предишната задача, разделяме теоремата на два случая:

1 сл. Ако $c \geq 1$, то $M' = M$ ще бъде ограничена по време от $cT(n)$, тъй като M е ограничена по време от $T(n)$, а $1 \leq T(n) \leq cT(n)$.

2 сл. Остава да разгледаме случая, когато $0 < c < 1$. Тъй като редицата с общ член $\frac{1}{n}$ клони към 0, винаги можем да намерим естествено число d, такова че $\frac{2}{d} < c$. Ще изложим конструкция на еднолентова машина на Тюринг M', симулираща друга, ограничена по време от $T(n)$, еднолентова машина на Тюринг M за време $\lceil \frac{1}{d}T(n) \rceil$.

Подобно на конструкцията в теоремата за линейна компресия, M' разбива лентата, която M използва, на равни интервали от по d клетки и кодира всеки от тези интервали в отделна клетка, но този път заедно със съседните му отляво и отдясно интервали, използвайки азбука $\Gamma' = \Gamma^d \times \Gamma^d \times \Gamma^d \cup \Gamma$ и състояния $Q' = (Q \times I_d \times \Gamma^d \times \Gamma^d \times \Gamma^d) \cup \{q_{accept}, q_{reject}\}$, където всяка буква от азбуката от вида $(\Gamma^d \times \Gamma^d \times \Gamma^d)$ кодира съответно текущия интервал от d клетки в M, левия интервал от d клетки в M и десния интервал от d клетки в M, втората компонента на състоянията от $(Q \times I_d \times (\Gamma^d \times \Gamma^d \times \Gamma^d))$ кодира в коя клетка от текущия интервал е главата на M в момента, а третата компонента държи "обновени стойности" на интервалите, над които е главата на M' в момента (M' кодира всеки от интервалите от d клетки в M на две места, съответно при намазване на едното място трябва да има механизъм, чрез който да се синхронизират промените).

Тъй като новите букви в M' могат да държат контекст от поне d клетки вляво и вдясно от клетката, върху която е главата на M, M' може да симулира d прехода в M за една стъпка, откъдето сложността по време на M' ще бъде точно $\lceil \frac{1}{d}T(n) \rceil$.

Лесно се вижда, че горната конструкцията може да се разшири до такава за много ленти, тоест всяка ограничена по време от $T(n)$ k-лентова машина на Тюринг M може да бъде трансформирана в константно по-бърза k-лентова машина на Тюринг M', която обаче работи константно по-бързо само за вече кодирана по начина описан по-горе входна дума. Тък като ние искаме да докажем, че съществува константно по-бърза машина на Тюринг за същия вход, M' ще трябва сама да извърши кодирането на входната дума. Това може да стане с едно сканиране надясно на входната лента и едно връщане наляво на главата върху лентата за кодиране за общо $n + \lceil \frac{n}{d} \rceil$ стъпки.

Така за входна дума с дължина n общият брой стъпки, които M' ще извърши, ще бъде:

$$n + \lceil \frac{n}{d} \rceil + \lceil \frac{1}{d}T(n) \rceil \leq n + \frac{n}{d} + \frac{1}{d}T(n) + 2$$

Тъй като $n \in o(T(n))$, за достатъчно големи n ще е изпълнено, че:

$$n + \frac{n}{d} + \frac{1}{d}T(n) + 2 < 3n + \frac{1}{d}T(n) < \frac{2}{d}T(n) < cT(n)$$

тоест M' ще бъде ограничена по време от $cT(n)$.

†В учебника на Hopcroft и Ullman под "ограничена по памет $S(n)$" се има предвид ограничена по памет от $max(1, \lceil S(n) \rceil)$. Ако ползваме тази дефиниция, M' наистина ще бъде ограничена по памет от всяка функция, тъй като е ограничена по памет от $\lceil \frac{1}{d}S(n) \rceil = 1$.

□

Задача 4. (Теорема за ускорението на Блум) За всяка тотална изчислима функция $r(n)$ съществува разрешим език L, такъв че за произволна машина на Тюринг M_i, приемаща L, съществува, машина на Тюринг M_j, приемаща L, такава че $r(S_j(n)) \leq S_i(n)$ за почти всички n.

Решение:

Забележка. Доказателството е пренаписана от мен версия на доказателството на теорема 12.14 от [1].

Без ограничение на общността нека вземем r да е напълно-построима по памет† монотонно растяща функция, такава че $r(n) \geq n^2$.† Дефинираме си функция h по следния начин:

$$h(n) = \begin{cases} 2 & n = 1 \\ r(h(n-1)) & n \geq 2 \end{cases}$$

Очевидно щом $r(n) \geq n^2$, то $h(n) \geq 2^{2^n}$.

Нека сега разгледаме изброяване $M_1, M_2, ...$ на всички машини на Тюринг с *read-only* входна лента, такова че кодът на машина M_i е с дължина най-много $\log_2 i$ (очевидно такова изброяване съществува). Ще построим език L, такъв че:

(1) Ако $\mathcal{L}(M_i) = L$, то $S_i(n) \geq h(n-i)$ за почти всички n.

(2) За всяко k съществува машина на Тюринг M_j, такава че $\mathcal{L}(M_j) = L$ и $S_j(n) \leq h(n-k)$ за почти всички n.

Ясно е, че ако намерим разрешим език L, за който горните две твърдения са изпълнени, то за произволна машина на Тюринг M_i, приемаща L, ще съществува машина на Тюринг M_j, приемаща L, такава че $r(S_j(n)) \leq S_i(n)$, защото от (1) $S_i(n) \geq h(n-i)$, а от (2) можем да вземем M_j да е такава, че:

$$r(S_j(n)) \leq r(h(n-i-1)) = h(n-i) \leq S_i(n) \text{за почти всички } n.$$

Ще изложим алгоритъм за конструкцията на език $L \subseteq 0^*$, който удовлетворява (1) и (2). Алгоритъмът ще разглежда последователно $n = 0, 1, 2, \dots$, и за всяко n ще определя дали думата 0^n е в L. По време на самото итериране, някои от машините на Тюринг в изброяването, което си фиксирахме ще бъдат "отхвърляни", като тези машини няма да приемат L. Нека си дефинираме $\sigma(n)$ като най-малкият индекс $j \leq n$, такъв че $S_j(n) < h(n-j)$, и M_j не е отхвърлена за итерациите $0, 1, \dots, n-1$. Ако $\sigma(n)$ съществува†, то алгоритъмът отхвърля машината $M_{\sigma(n)}$, приемайки 0^n точно тогава, когато $M_{\sigma(n)}$ не приема 0^n.

Ще покажем, че L отговаря на (1) с допускане на противното - нека приемем, че съществува машина на Тюринг M_i, такава че $\mathcal{L}(M_i) = L$ и $S_i(n) < h(n-i)$ за безкраен брой остойностявания на n. Тогава за безкраен брой итерации в алгоритъма конструиращ L, ще бъде изпълнено, че $S_i(n) < h(n-i)$, и за още най-много краен брой от тях M_i ще стане най-малката неотхвърлена машина с това свойство и ще бъде отхвърлена. Но ние знаем, че отхвърлените машини не разпознават L, следователно M_i няма да разпознава L, което е противоречие.

Остава да покажем, че L отговаря на (2). Нека вземем произволно k. Ще построим машина на Тюринг $M = M_j$, която разпознава L и работи за памет $S_j(n) \leq h(n-k)$ за почти всички n. При определянето дали 0^n принадлежи на L, M трябва да симулира $M_{\sigma(n)}$ върху 0^n. За изчисляването на $\sigma(n)$ обаче, M трябва да "знае" кои от машините на Тюринг са били отхвърлени за първите n итерации в алгоритъма, конструиращ L. Симулирането на тези итерации би отнело памет $\max\{h(l-i) \mid 0 \leq l \leq n \ \& \ 1 \leq i \leq l\}$, което в общия случай е повече от $h(n-k)$.

За да се справим с този проблем, забелязваме, че съществува итерация с някакъв най-малък номер n_1, от която нататък никоя от машините $M_1, M_2, \dots M_k$ не бива отхвърлена. Тогава можем да добавим към M вградени правила за думите от вида $0^{<n_1}$, принадлежащи на L, и вграден "списък" с номерата на машините на Тюринг, които алгоритъмът, конструиращ L, би отхвърлил за $n_1 - 1$ итерации†. По този начин, ако $n < n_1$, M няма да използва никаква допълнителна памет, а ако $n \geq n_1$, M ще трябва да симулира итерации с номер $n_1 \leq l \leq n$ за машини с индекс $k < i \leq n$, използвайки най-много $h(n-k+1)$ памет.

Тъй като кодът на M_i е по-малък от $\log_2 n$ за $i \leq n$, списъкът от отхвърлени машини на Тюринг, които M трябва да "помни" ще заема памет $n \log_2 n$, а за симулациите на машините от $M_{k+1}, M_{k+2}, ..., M_n$, ще необходима памет $\log_2 n \ h(n-k+1)$. От наблюдението, че $h(n) \geq 2^{2^n}$, за почти всички n ще бъде в сила, че:

$$h(n-k) = h(n-k-1)^2 \geq 2^{2^{n-k-1}} h(n-k-1) \geq n \log_2 n \ h(n-k-1) \geq \log_2 n \ h(n-k-1) + n \log_2 n$$

тоест машината M ще работи за памет $S_j(n) \geq h(n-k)$ върху почти всички думи. □

†Под "напълно-построима по памет функция $r(n)$" разбираме функция $r(n)$, за която съществува машина на Тюринг, която използва точно $r(n)$ клетки за **всяка** входна дума с дължина n.

†Причината да не се губи общност е, че $r(n)$ и n^2 са изчислими фукнции, тоест винаги можем да конструираме напълно-построима по памет монотонно растяща функция породена от $r(n)$, която да расте поне квадратично - например можем да си дефинираме функцията $r'(n)$ чрез машина на Тюринг M, която строи $r'(n)$, изчислявайки и построявайки $r'(n-1)$, $r(n)$ и n^2 на отделни ленти.

†Очевидно $\sigma(n)$ може да се изчисли чрез тестване на неотхвърлените машини с индекси $\leq n$ върху всяка дума с дължина n.

†Не е нужно да знаем колко е n_1, машината на Тюринг, съобразена с правилното n_1, съществува независимо от наблюденията ни.

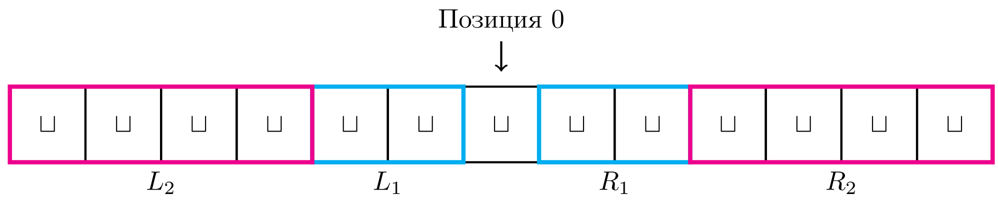
Задача 5. Съществува машина на Тюринг $\mathcal{U}$, такава че за всеки две думи $\omega, \alpha \in \{0,1\}^*$, $\mathcal{U}(\omega, \alpha) = M_\omega(\alpha)$. Нещо повече, ако $M_\omega(\alpha)$ спира за T стъпки, то $\mathcal{U}(\omega, \alpha)$ спира за $CT \log T$ стъпки, където C е някаква константа, зависеща единствено от големината на азбуката, състоянията и лентите на M_ω.

Решение:

Забележка. Идеята за доказателството е взета от [2].

Съществената част от задачата е да измислим как да симулираме една стъпка от изчислението на произволнолентова машина на Тюринг M чрез някакъв константен брой ленти на $\mathcal{U}$ за колкото се може по-малко време. Очевидно е, че $\mathcal{U}$ няма как да симулира различните ленти на M паралелно (или поне не всички), затова, за да направим последователната симулация на много ленти от M върху една лента на $\mathcal{U}$ ефективна, трябва смяната между две ленти на M да се осъществява колкото се може "по-плавно" (тоест без да се налага $\mathcal{U}$ да търси главата на следващата лента от M). За целта, $\mathcal{U}$ ще държи всяка от главите на лентите от M в началото на своята лента и, вместо да ги мести, ще "плъзга" всяка от лентите наляво и надясно под съответстващата ѝ глава. Естествено, плъзгането на една цяла лента би отнело твърде много време, затова $\mathcal{U}$ ще плъзга само определени сегменти от лентите.

Идеята за симулация върху машина на Тюринг M с единствена лента изглежда по следния начин. Представяме си, че началната позиция на главата на $\mathcal{U}$ е позиция 0 и мислено разделяме лявата част на лентата на зони $L_1, L_2, \ldots$ и дясната част на лентата на зони $R_1, R_2, \ldots$, където за всяко i, L_i и R_i съдържат по 2^i клетки както е илюстрирано на фигурата по-долу.



Ще казваме, че една зона е "празна", ако е запълнена единствено със специалния символ "$\smile$", "пълна", ако е запълнена единствено със символи различни от "$\smile$" и "наполовина пълна", ако само 2^{i-1}-те най-отдалечени от позиция 0 символи в зоната са "$\smile$".

Инвариант: В началото на стъпка x от симулацията на M в $\mathcal{U}$, за произволни зони L_i, R_i е изпълнено, че ако една от зоните е била достигната от главата на $\mathcal{U}$ на някоя предишна стъпка, то и двете зони са били достигнати и:

1. Или R_i е празна и L_i е пълна,
2. Или R_i е пълна и L_i е празна,
3. Или R_i е наполовина пълна и L_i е наполовина пълна.

Освен това лентата, която се получава след премахване на клетките, съдържащи символ "$\smile$", от лентата на $\mathcal{U}$, е идентична на лентата на на стъпка x от изчислението на M за същата дума.

Алгоритъмът за симулация на стъпка надясно в M е следния:

1. Главата на $\mathcal{U}$ започва да върви надясно, докато намери първата зона R_i, която не е празна.
2. Главата копира първия символ от R_i на позиция 0, а останалите $2^{i-1} - 1$ (или $2^i - 1$, ако R_i е била пълна) символи разпределя в първите половини на зони $R_1, R_2, \ldots, R_{i-1}$ (и R_i, ако R_i е била пълна). Клетките в R_i, които са останали без символи, биват запълнени със символа "$\smile$".
3. След като поправянето на дясната част от лентата е приключило, главата на $\mathcal{U}$ измества пълните зони $L_1, L_2, \ldots, L_{i-1}$ в първата половина на левите им зони $L_2, L_3, \ldots, L_i$, запълва вторите половини на зоните с "$\smile$" и, ако L_i е била наполовина пълна, измества тази половина в далечния край на L_i, освобождавайки място за новите символи, които ще дойдат от L_{i-1}. Накрая $\mathcal{U}$ копира досегашния символ от позиция 0 (тоест символа в началото на стъпката) в по-близката до позиция 0 половина на L_1, след което връща главата на $\mathcal{U}$ на позиция 0.

Стъпката наляво е аналогична, а когато главата на M не се мести, $\mathcal{U}$ не извършва никакви плъзгания.

С помощта на специална лента за копиране цялата процедура може да се извърши за време $O(2^i)$. Ключово е наблюдението, че след подобно плъзгане на i зони, клетките отляво и отдясно на позиция 0 се запълват по такъв начин, че повторно плъзгане на i зони би било възможно едва след изпразване или запълване на наполовина пълните зони $R_1, R_2, ..., R_{i-1}$, тоест след най-малко $2^{i-1}-1$ хода. Щом това е така, то броят плъзгания на i зони за цялата симулация на M ще бъде най-много $\frac{T}{2^{i-1}}$. Тъй като за време T $\mathcal{U}$ може да стигне най-много до зона $\lceil \log T \rceil$, общият брой стъпки, които $\mathcal{U}$ извършва в плъзгания на зони може да се изрази със следната сума:

$$\sum_{i=1}^{\lceil \log T \rceil} \frac{T}{2^{i-1}} O(2^i) = O(T \log T)$$

За да разширим горната конструкция до такава за k ленти, можем мислено да заменим позиция 0 с редица от k 0-леви позиции и да увеличим размера на всяка зона от вида L_i или R_i до $k2^i$ клетки. За всяка лента от M_ω ще извършваме изчислението единствено в заделените за нея части от зоните, като по този начин симулацията ще се забави приблизително k^2 пъти.

Остава да съобразим, че кодирането на азбуката Γ чрез $\{0,1\}$ би довело до допълнително забавяне от $\lceil \log_2 |\Gamma| \rceil$ ($\mathcal{U}$ може да кодира всяка клетка от M чрез $\lceil \log_2 |\Gamma| \rceil$ свои клетки), сканирането и презаписването на текущото състояние би довело до допълнително забавяне от порядъка на $\lceil \log_2 |Q| \rceil$, а търсенето на правило за текущата конфигурация във функцията на преходите на M би довело до допълнително забавяне от порядъка на $|Q||\Gamma|$ (в функцията на преходите има $|Q||\Gamma|$ правила).

Така, комбинирайки всички забавяния, получаваме, че $\mathcal{U}$ ще работи за време $CT \log T$, където C е константа зависеща единствено от големината на азбуката, състоянията и лентите на M_ω. ☐

Задача 6. (Теорема на Имерман-Селепчени) За всяка функция $S(n) \geq \log n$, $\mathrm{NSPACE}(S(n)) = \mathrm{co\text{-}NSPACE}(S(n))$.

Решение: Теоремата може да се докаже чрез следното наблюдение:

Наблюдение За всяка ограничена от $S(n) \geq \log n$ машина на Тюринг M съществува друга машина на Тюринг N, която строи графа[a] на M за входна дума ω, използвайки работни ленти с размер $S(n)$ и *write-only* изходна лента с глава, която може да се движи само надясно, и размер $p(c^{S(n)})$, където c е някаква константа, а p е някакъв полином.

[a] Под "Граф на машина на Тюринг M за входна дума ω" се има предвид граф с върхове конфигурациите на M, които могат да се получат при изчислението за вход ω, и ребра свързващи конфигурациите, между които има преходи.

Интуитивно, машината N ще генерира последователно всички конфигурации на M, записвайки ги на изходната си лента като върховете на графа G, след което за всяка от конфигурациите ще генерира всички нейни съседни конфигурации, записвайки наредените двойки от съседи като ребрата на графа G. За да изчисли броя възможни конфигурации на M за произволна входна дума ω, N ще използва разглеждания на лекции алгоритъм за налучкване на горна граница на използвани клетки за дадено изчисление. Тъй като този алгоритъм не увеличава големината на работната памет, N ще използва най-много $S(n)$ работна памет за генерирането на G, а големината на G (съответно и размерът на изходната лента на N) ще е най-много $p(c^{S(n)})$ за някаква константа c и някакъв полином p, понеже върховете на графа са ограничени от $c^{S(n)}$, а ребрата са най-много квадратично повече.

($\Rightarrow$) Нека сега вземем произволен език L от $\mathrm{NSPACE}(S(n))$ и нека фиксираме произволна недетерминирана машина на Тюринг M, която разпознава L и също така има единствена финална конфигурация κ_{accept}[†]. Ще докажем, че L принадлежи на $\mathrm{co\text{-}NSPACE}(S(n))$, което е същото като $\overline{L}$ да принадлежи на $\mathrm{NSPACE}(S(n))$. За целта конструираме недетерминитана машина на Тюринг R за $\overline{L}$, която комбинира машината N с ограничена по памет от $\log n$ недетерминирана машина на Тюринг $M_{\overline{PATH}}$ с единствена приемаща конфигурация за езика $\overline{PATH}$ - R ще симулира $M_{\overline{PATH}}$ върху графа кодиращ изчислението на M за входна дума ω, стартов връх $init(\omega)$ и финален връх κ_{accept}. За да спести памет, R няма да генерира самия граф, а вместо това ще използва специална лента брояч с размер $\log |G| \leq \log p(c^{S(n)}) = O(S(n))$, за запомняне на текущия индекс на клетката в, която трябва да се намира главата на лентата на G. Всеки път, когато симулацията на $M_{\overline{PATH}}$ иска да достъпи клетката под главата, R ще изчислява клетката използвайки работните ленти на N и още една допълнителна лента брояч(отново с размер $O(S(n))$) - вместо да записва на изходната си лента графа G, N ще инкрементира брояча докато стигне до индекса, от който се интересува $M_{\overline{PATH}}$.

По този начин R ще приема дума ω точно когато $\omega \in \overline{L}$, тъй като $\omega \in \overline{L}$ тогава и само тогава, когато не съществува път от $init(\omega)$ до κ_{accept} в графа на M за входна дума ω. Нещо повече R ще е ограничена по памет от $S(n)$, тъй като N и $M_{\overline{PATH}}$ работят за $S(n)$ памет.

($\Leftarrow$) В обратната посока, нека вземем произволен език L от $\mathrm{co\text{-}NSPACE}(S(n))$. От дефиницията на $\mathrm{co\text{-}NSPACE}(S(n))$, щом L е в $\mathrm{co\text{-}NSPACE}(S(n))$, то $\overline{L}$ е в $\mathrm{NSPACE}(S(n))$. Е, от първата част на доказателството знаем, че щом, $\overline{L}$ е в $\mathrm{NSPACE}(S(n))$, то $\overline{\overline{L}}$ е в $\mathrm{NSPACE}(S(n))$, тоест L ще е в $\mathrm{NSPACE}(S(n))$. ☐

Задача 7. (2-SAT) Съществува полиномиален алгоритъм, който по дадена КНФ ϕ, в която всяка от дизюнктивните клаузи е с единствен дизюнкт на два литерала, определя дали съществува остойностяване на променливите, за което ϕ е истина.

[†] Очевидно е, че всяка машина на Тюринг може да се сведе до такава с единствена финална конфигурация без новата машина да използва допълнително памет

Решение:
Първо ще въведем няколко дефиниции:

- Импликационен вид на формула $\phi = (\lambda_1 \vee \lambda_2) \wedge (\lambda_3 \vee \lambda_4) \wedge \cdots \wedge (\lambda_{n-1} \vee \lambda_n)$ е формулата

$$\psi = (\neg\lambda_1 \to \lambda_2) \wedge (\neg\lambda_2 \to \lambda_1) \wedge (\neg\lambda_3 \to \lambda_4) \wedge (\neg\lambda_4 \to \lambda_3) \wedge \cdots \wedge (\neg\lambda_{n-1} \to \lambda_n) \wedge (\neg\lambda_n \to \lambda_{n-1})$$

- Импликационният граф на формула ϕ е графът с върхове литералите в импликационния вид ψ на ϕ и ориентирани ребра импликациите между литералите в ψ.

Лесно се вижда, че ако ϕ е изпълнима, то и импликационният ѝ вид ψ е изпълним. Освен това за всяко остойностяване на променливите във ϕ, ϕ е истина тогава и само тогава, когато в импликационния ѝ граф не съществува ребро (λ_1, λ_2), за което $\lambda_1 \equiv T \wedge \lambda_2 \equiv F$.

Обратно към задачата, ще изложим алгоритъм, който решава 2-SAT:

```
0  SOLVE–2–SAT(ϕ: булева формула)
1    G ← BuildImplicationGraph(ϕ)
2    Toposort(SCC(G))
3    for x ∈ vars(ϕ):
4      if SCC(x) = SCC(¬x):
5        return False
6      else if SCC(x) < SCC(¬x):
7        x ← False
8      else:
9        x ← True
10   return True
```

SOLVE-2-SAT строи импликационния граф G на формулата ϕ, след което извършва топологично сортиране на силно свързаните компоненти на G. Ако съществува променлива, която е в една компонента с отрицанието си, то алгоритъмът връща лъжа като индикация, че ϕ не е изпълнима. В противен случай SOLVE-2-SAT остойностява x като лъжа, ако в топологичната сортировка силно свързаната компонента на x е с по-малък индекс от тази на $\neg x$, или като истина, ако обратното е изпълнено.

> **Твърдение:** SOLVE-2-SAT връща True тогава и само тогава, когато ϕ е изпълнима. Нещо повече, в случай, че ϕ е изпълнима, SOLVE-2-SAT остойностява променливите във ϕ, така че ϕ да е истина.

Доказателство:
($\Rightarrow$) Нека допуснем, че SOLVE-2-SAT връща True и грешно остойностяване за някоя формула ϕ, която не е изпълнима. Щом ϕ не е изпълнима, то в импликационният граф G на ϕ съществува ребро (x, y), за което SOLVE-2-SAT остойностява $x \equiv T$ и $y \equiv F$. Ще разгледаме това ребро, както и неговото обратно ребро $(\neg y, \neg x)$ (от конструкцията на G щом $(x, y) \in E(G)$, то и $(\neg y, \neg x) \in E(G)$). Факта, че тези две ребра съществуват влече, че:

$$\mathrm{SCC}(x) \leq \mathrm{SCC}(y)$$
$$\mathrm{SCC}(\neg y) \leq \mathrm{SCC}(\neg x)$$

Освен това, тъй като $x \equiv T$, от начина, по-който работи SOLVE-2-SAT, ще бъде изпълнено, че:

$$\mathrm{SCC}(\neg x) < \mathrm{SCC}(x)$$

Комбинирайки неравенствата получаваме:

$$\mathrm{SCC}(\neg y) \leq \mathrm{SCC}(\neg x) < \mathrm{SCC}(x) \leq \mathrm{SCC}(y)$$

В същото време, тъй като $y \equiv F$, от начина на работа на SOLVE-2-SAT би трябвало да е в сила $\mathrm{SCC}(y) < \mathrm{SCC}(\neg y)$, но последното противоречи на изведеното. □

($\Leftarrow$) В обратната посока, нека допуснем, че SOLVE-2-SAT връща False за някоя изпълнима формула ϕ. Тогава SOLVE-2-SAT е влязал в `if`-а на ред 4, тоест съществува променлива x, която е в една и съща силно свързана компонента с отрицанието си $\neg x$. Щом x и $\neg x$ са в една и съща свързана компонента, то в G ще съществува път $(x, v_1, v_2, \ldots, v_k, \neg x)$ от x до $\neg x$. Нека БОО в остойностяването на ϕ x е истина[†]. Тогава все някъде по пътя ще има ребро от връх със стойност истина към връх със стойност лъжа, тоест ϕ няма да е изпълнима, което е противоречие. □

[†]ако x е лъжа, то можем да разгледаме път $(\neg x, u_1, u_2, \ldots, u_k, x)$ от $\neg x$ до x

Анализ на сложността: За построяването на импликационния граф G на ϕ е нужно полиномиално време. Нещо повече, големината на графа G е $O(|\phi|)$, откъдето алгоритмите за топологично сортиране и намиране на свързани компоненти също ще работят за полиномиално време (такива алгоритми обикновено се реализират за време $O(n+m)$).

Задача 8. Докажете, че $\mathrm{NTIME}(T(n)) \subseteq \mathrm{DSPACE}(T(n))$.

Решение: Нека L е произволен език от $\mathrm{NTIME}(T(n))$ и нека M_L е ограничена по време от $T(n)$ недетерминирана машина на Тюринг, която разпознава L. Ако си мислим за изчислението на M_L върху дума ω като за дърво с корен $init(\omega)$, то това дърво ще има дълбочина $T(n)$ и максимална разклоненост от $3|Q||\Gamma|$ (при прочитане на буква M_L може да си смени състоянието по $|Q|$ начина, да напише нова буква по $|\Gamma|$ начина и да премести главата си по 3 начина). Така листата на дървото ще бъдат най-много $(3|Q||\Gamma|)^{T(n)}$ и съответно пътищата от корена до всяко листо ще могат да се изброят от ограничен по памет от $T(n)$ логаритмичен брояч с база $3|Q||\Gamma|$ (във всяка клетка броячът ще държи по кое разклонение поема пътя).

Оттук лесно можем да конструираме ограничена по памет от $T(n)$ детерминирана машина на Тюринг M, която разпознава L - на една от лентите си M ще използва логаритмичния брояч, за да итерира през всички възможни пътища в изчислението на M_L. За всеки от пътищата M ще проверява дали пътят завършва в приемащо състояние. Ако съществува поне един такъв път, то M завършва успешно. □

Задача 9. Докажете, че проблемът за съществуването на Хамилтонов път в граф е NP-пълен.

Забележка. Идеята за доказателството е взета от теорема 7.46 от [3].

Първо ще докажем. че проблемът HamiltonianPath $= \{\lceil G\rceil \mid G$ е граф, в който има Хамилтонов път$\}$ е в класа NP. За целта представяме следния недетерминиран полиномиален алгоритъм:

```
HasHamiltonianPath(G: ориентиран граф)
  n ← |V(G)|
  visited[1 .. n] ← [False; n]
  v ← Choose(V(G))
  visited[v] ← True
  for i ← 2 .. n:
    u ← Choose(V(G))
    if (v, u) ∉ E(G) or visited[u]:
      return False
    v ← u
    visited[v] ← True
  return True
```

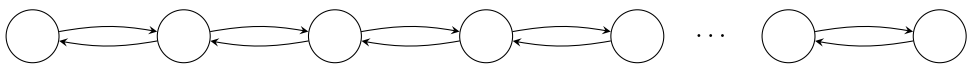
HasHamiltonianPath генерира път с дължина $n = |V(G)|$ като недетерминирано избира (или с други думи налучква) върховете от пътя. Ако в някое изчисление не съществува ребро между досегашния и новоизбрания връх или новоизбрания връх е вече част от пътя, то това изчисление на HasHamiltonianPath се проваля. Така се гарантира, че HasHamiltonianPath връща истина точно тогава, когато в G има път $(v_1, v_2, \ldots, v_n)$, такъв, че:

$$(\forall i \in \{1, 2, \ldots, n-1\}\ (v_i, v_{i+1}) \in E(G)) \land (\forall i, j \in \{1, 2, \ldots, n\}\ v_i = v_j \to i = j)$$

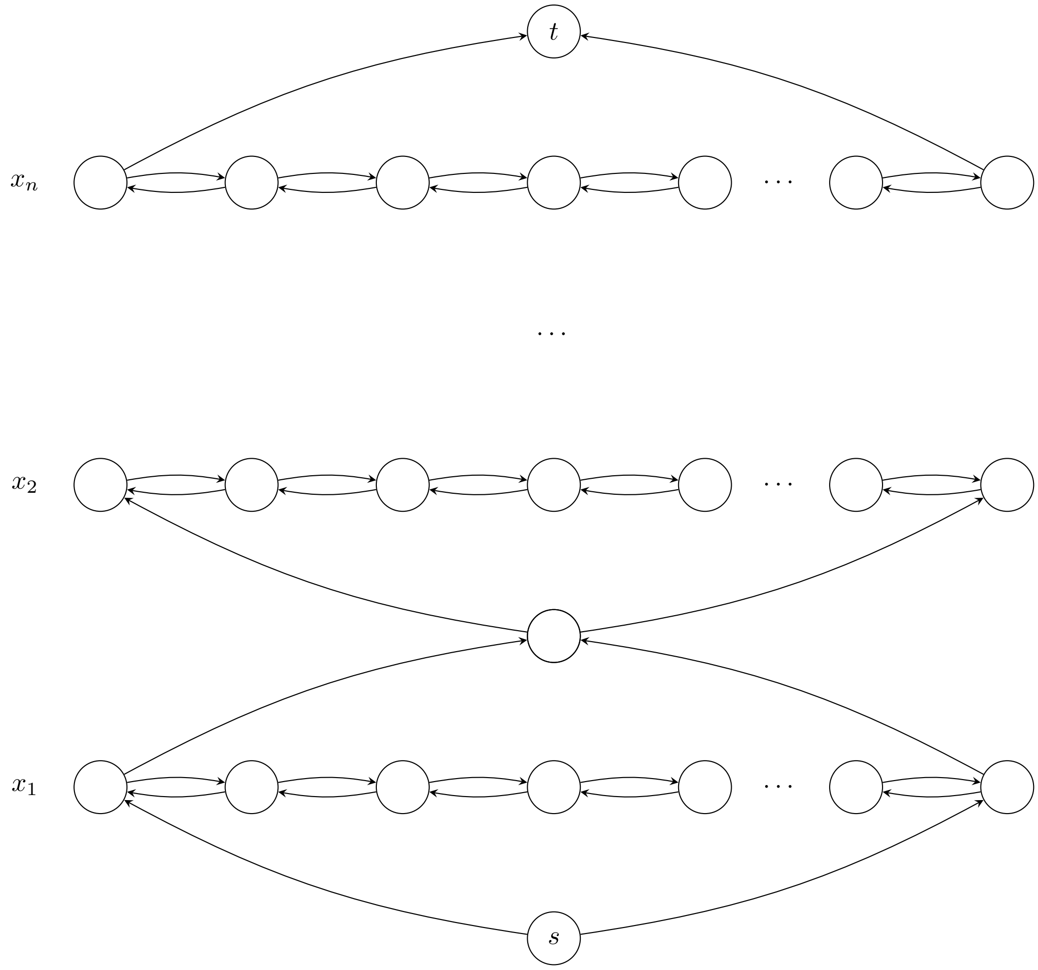
Тъй като последното е необходимо и достатъчно условие в G да има Хамилтонов път, HasHamiltonianPath работи коректно.

Сега ще покажем, че всеки NP проблем може да се реши чрез HamiltonianPath. За целта е достатъчно да направим свеждането 3-SAT $\leq^P$ HamiltonianPath. Един свеждащ алгоритъм може да работи по следния начин. Нека ϕ е произволна КНФ с променливи $x_1, x_2, \ldots, x_k$ и дизюнктивни клаузи $c_1, c_2, \ldots c_l$. За всяка от променливите свеждащият алгоритъм създава $3l+3$ нови върха в графа, свързани по следния начин:

x_i

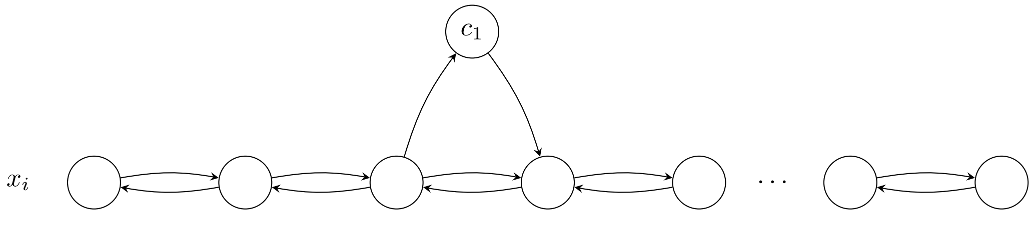


Връзките между веригите за различките променливи ще се осъществяват чрез специални междинни върхове свързани към краищата на веригите, както е показано по-долу:

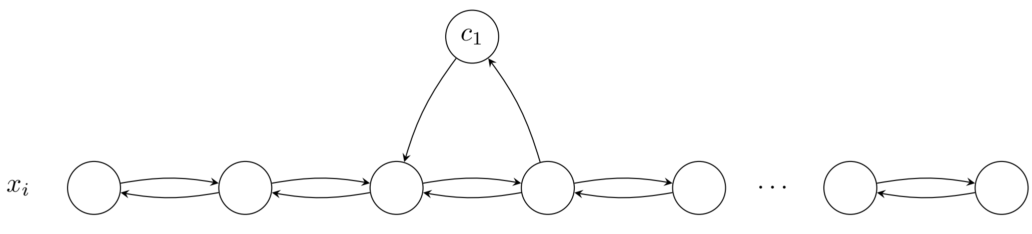


Най-долният връх s е източник, а най-горният връх t - сифон. Така се форсира последователно остойностяване на променливите $x_1, x_2, \dots x_n$ (Ако в G има Хамилтонов път той задължително започва от s, след което последователно преминава през веригите на $x_1, x_2, \dots, x_n$ и завършва в t.) Самото остойностяване на променливите ще зависи от посоката, на влизане във веригата - при влизане отялво променливата ще е истина, в противен случай - лъжа. Остава да измислим как да "отмятаме" клаузи, които са станали истина в резултат на остойностяването. Нека клауза c_j би станала истина, ако литералът $\lambda_i \in \{x_i, \overline{x_i}\}$ бъде остойностен като истина.

1 сл. $\lambda_i = x_i$. Тогава добавяме ребро между $(3j)$-тия връх във веригата на x_i и върха за c_j и добавяме ребро между върха за c_j и $(3j+1)$-вия връх във веригата:



2 сл. $\lambda_i = \overline{x_i}$. Тогава добавяме ребро между $(3j+1)$-вия връх във веригата на x_i и върха за c_j и добавяме ребро между върха за c_j и $3j$-вия връх във веригата (тоест правим същото като в горния случай, но с обърнати посоки на ребрата):



Забележете, че върховете на позиции от вида $3j + 2$ не получават нови ребра. Това е необходимо, за да се гарантира, че след преминаването през някоя клауза Хамилтоновия път задължително се връща обратно към веригата, от която е влязъл в клаузата.

Лесно се вижда, че в G винаги има път минаващ през s, t и върховете за променливите. Остава да забележим, че ако ϕ е удовлетворима, то всеки връх клауза в G може да бъде покрит, чрез отклонение по някои от веригите за променливите и непосредствено връщане обратно към веригата, от която е започнала отклонението. В обратната посока, ако в G има Хамилтонов път, то той със сигурност започва от s, минава през реда за x_1, "активирайки" някои клаузи (непосредствено връщайки се на реда след активирането), минава през реда за x_2, активирайки други клаузи и т.н. докато не мине през всички редове и не завърши в t. Тъй като пътят е Хамилтонов, всяка клауза ще бъде активирана от някоя променлива съответно, ϕ ще бъде истина при остойностяване на променливите съобразено с посоките на влизане на пътя във веригите. $\square$

Препратки:

[1] John Hopcroft и Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation*. 1979.

[2] Valentine Kabanets. *Universal Turing machines*. 2017. URL: `https://www2.cs.sfu.ca/~kabanets/407/lectures/lec3.pdf`.

[3] Michael Sipser. *Introduction to the Theory of Computation*. 2013.