Úplné problémy pro další třídy složitosti

PSPACE-úplné problémy, EXPTIME-úplné problémy, . . .

- Problém A je PSPACE-**těžký**, jestliže je každý problém A' z PSPACE polynomiálně převeditelný na problém A.
- Problém A je PSPACE-**úplný**, jestliže je PSPACE-těžký a navíc sám patří do třídy PSPACE.

- Problém A je EXPTIME-**těžký**, jestliže je každý problém A' z EXPTIME polynomiálně převeditelný na problém A.
- Problém A je EXPTIME-**úplný**, jestliže je EXPTIME-těžký a navíc sám patří do třídy EXPTIME.

$\vdots$

PSPACE-úplné problémy, EXPTIME-úplné problémy, ...

Obecně pro libovolnou třídu složitosti $\mathcal{C}$ můžeme zavést třídy **$\mathcal{C}$-těžkých** a **$\mathcal{C}$-úplných** problémů:

Definice

- Problém A je **$\mathcal{C}$-těžký**, jestliže je každý problém A' ze třídy $\mathcal{C}$ polynomiálně převeditelný na problém A.
- Problém A je **$\mathcal{C}$-úplný**, jestliže je $\mathcal{C}$-těžký a navíc sám patří do třídy $\mathcal{C}$.

Kromě NP-úplných problémů tak máme PSPACE-úplné problémy, EXPTIME-úplné problémy, EXPSPACE-úplné problémy, 2-EXPTIME-úplné problémy, ...

Obecně se dá říci, že $\mathcal{C}$-úplné problémy jsou vždy ty nejtěžší problémy v dané třídě $\mathcal{C}$.

P-úplné problémy, NL-úplné problémy, ...

Poznámka: Výše uvedeným způsobem zavedené pojmy $\mathcal{C}$-těžkých a $\mathcal{C}$-úplných problémů, kdy byl v definici použit pojem **polynomiální převoditelnosti**, nedávají příliš smysl pro třídu P a další třídy, které jsou jejími podmnožinami (jako třeba NL).

Pro takové třídy se zavádí pojmy $\mathcal{C}$-těžké a $\mathcal{C}$-úplné problémy podobným způsobem jako v předchozích definicích, ale místo polynomiálních redukcí se používají, tzv. **logspace redukce**:

- algoritmus realizující daný převod musí být deterministický a mít logaritmickou prostorovou složitost

Tímto způsobem se zavádí například:

- P-úplné a P-těžké problémy
- NL-úplné a NL-těžké problémy

Třídy co-NP, co-NL, ...

U tříd složitosti, kde to, zda daný problém patří nebo nepatří do dané třídy, závisí na existenci nebo neexistenci **deterministického** algoritmu řešícího tento problém, platí, že:

- libovolný problém A patří do dané třídy právě tehdy, když doplňkový problém $\bar{A}$ problému A patří do dané třídy.

  Algoritmus řešící problém $\bar{A}$ dostaneme z algoritmu řešícího problém A tak, že prostě znegujeme odpověď, kterou by tento původní algoritmus vrátil jako výstup.

Toto ovšem nemusí být nutně pravda u tříd, které se vztahují k existenci **nedeterministických** algoritmů — jako jsou třeba třídy NP, NL nebo NEXPTIME.

Třídy co-NP, co-NL, ...

Z toho důvodu se zavádí třídy jako:

- co-NP — třída tvořená právě těmi problémy, které jsou doplňkovými problémy problémů ze třídy NP
- co-NL — třída tvořená právě těmi problémy, které jsou doplňkovými problémy problémů ze třídy NL
- ...

Například třída co-NP je tak tvořena právě těmi problémy, pro které:

- Existuje nedeterministický algoritmus s polynomiální časovou složitostí rozpoznávající právě ty vstupy, pro které je odpověď $\textsc{Ne}$.
- Pro vstupy, pro které je odpověď $\textsc{Ne}$ existují polynomiálně velcí svědci, které je možné ověřit deterministickým algoritmem v polynomiálním čase.

co-NP-úplné problémy

Analogickým způsobem, jako jsou definovány NP-těžké a NP-úplné problémy je možné definovat i co-NP-těžké a co-NP-úplné problémy.

Příklady co-NP-úplných problémů:

UNSAT

Vstup: Booleovská formule φ.

Otázka: Je formule φ nesplnitelná?

Doplňkový problém k problému IS (nezávislá množina)

Vstup: Neorientovaný graf G a číslo k.

Otázka: Platí, že v grafu G neexistuje nezávislá množina velikosti k ?

Poznámka: Redukce dokazující co-NP-obtížnost doplňkových problémů jsou přesně ty samé redukce, které dokazují NP-obtížnost původních problémů.

co-NP-úplné problémy

Příklad problému, který je přirozenější definovat ve formě, která patří do třídy co-NP:

TAUTOLOGY

Vstup: Booleovská formule φ.

Otázka: Je formule φ tautologií?

Opět se jedná o příklad co-NP-úplného problému.

Třída PSPACE

Třída PSPACE

Připomeňme si definici třídy PSPACE:

Rozhodovací problém A patří do třídy PSPACE právě tehdy, když existuje algoritmus s **polynomiální prostorovou** složitostí (tj. algoritmus s prostorovou složitostí $\mathcal{O}(f(n))$, kde $f(n)$ je polynom), který ho řeší.

Připomeňme si rovněž Savitchovu větu:

- Ke každému **nedeterministickému** algoritmu s prostorovou složitostí $f(n)$ je možné sestrojit **deterministický** algoritmus s prostorovou složitostí $\mathcal{O}(f(n)^2)$, který dává odpověď ANO právě pro ty vstupy, pro které existuje přijímající výpočet původního nedeterministického algoritmu.

Pro ukázání toho, že nějaký daný problém A patří do třídy PSPACE, proto stačí ukázat **nedeterministický** algoritmus s polynomiální prostorovou složitostí, který ho řeší.

Přechodové systémy

Řekněme, že máme dán nějaký systém, kde:

- Daný systém se může nacházet v libovolném stavu z nějaké množiny **stavů**.

  Označme množinu těchto stavů *States*.

- Mezi těmito stavy je možné přecházet pomocí nějak definovaných **přechodů**.

  Označme *Transitions* množinu těchto přechodů, tj. těch dvojic (α, β), kde $\alpha, \beta \in \mathit{States}$ a je možné přejít jedním krokem ze stavu α do stavu β.

  Zápisem

  $$\alpha \longrightarrow \beta$$

  označme to, že platí $(\alpha, \beta) \in \mathit{Transitions}$.

Přechodové systémy

- Řekneme, že stav α_n je **dosažitelný** ze stavu α_0, jestliže existuje nějaká konečná posloupnost $\alpha_0, \alpha_1, \ldots, \alpha_n$, kde $k \geq 0$, a kde

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \quad \cdots \quad \longrightarrow \alpha_{n-1} \longrightarrow \alpha_n$$

  To, že stav β je dosažitelný ze stavu α označme zápisem

$$\alpha \longrightarrow^* \beta$$

- Řekněme, že máme dán nějaký **počáteční stav** α_0 a nějaký **koncový** stav α_f.

  Můžeme se ptát, zda platí

$$\alpha_0 \longrightarrow^* \alpha_f$$

  (Obecněji bychom mohli uvažovat i případ, kdy máme dánu **množinu počátečních stavů** a **množinu koncových stavů**.)

Přechodové systémy

Na takový systém se tedy můžeme dívat jako na **orientovaný graf**:

- **vrcholy** — prvky množiny *States*
- **hrany** — přechody z množiny *Transitions*

Dosažitelnost odpovídá tomu, že mezi danými vrcholy existuje cesta.

Tj. $\alpha \longrightarrow^* \beta$ platí právě tehdy, když v daném grafu existuje cesta z α do β.

Přechodové systémy

Takovýto systém nemusí být dán explicitně, tj. vyjmenováním všech stavů a přechodů, ale může být popsán nějakým způsobem **implicitně**, např.:

- **Stavy** budou reprezentovány pomocí nějakých konečných objektů (např. pomocí slov nad nějakou abecedou Σ).

  Navíc budeme mít algoritmus, který pro daný objekt bude schopen určit, zda tento objekt reprezentuje stav daného systému nebo ne (např. jestli dané slovo je zápisem stavu z množiny *States* nebo ne).

- **Přechody** budou reprezentovány pomocí nějakého algoritmu, který pro každou dvojici $\alpha, \beta \in \mathit{States}$ určí, zda platí

$$\alpha \longrightarrow \beta$$

- **Počáteční stavy** budou reprezentovány pomocí algoritmu, který pro libovolný stav α určí, zda se jedná o počáteční stav nebo ne.

- **Koncové stavy** budou reprezentovány pomocí algoritmu, který pro libovolný stav α určí, zda se jedná o koncový stav nebo ne.

Přechodové systémy

Vezměme si nyní nějaký takový systém, kde navíc budeme předpokládat:

- Stavy je možné reprezentovat pomocí **polynomiálně** velkých objektů, u kterých je navíc možné algoritmem s polynomiální prostorovou složitostí ověřit, zda se jedná o stav z množiny *States* nebo ne.

  Stavů tedy obecně může být exponenciálně mnoho, ale na uložení jednoho stavu postačuje polynomiální množství paměti.
- Existují algoritmy s polymiální prostorovou složitostí, které pro libovolné stavy α, β umožňují otestovat:
  - zda α a β jsou jeden a tentýž stav, tj. zda platí $\alpha = \beta$
  - zda platí $\alpha \longrightarrow \beta$
  - zda je α počáteční stav
  - zda je α koncový stav.
- Existuje možnost jak s polynomiálním množstvím paměti postupně generovat všechny stavy z množiny *States*, tj. máme algoritmus s polymiální prostorovou složitostí, který dostane popis stavu α a vrátí jako výsledek popis následujícího stavu α'.

Přechodové systémy

Pro takový systém můžeme snadno sestrojit **nedeterministický** algoritmus s polynomiální prostorovou složitostí řešící problém, zda je nějaký koncový stav dosažitelný z nějakého počátečního stavu,
tj. zda existuje nějaký počáteční stav α_0 a nějaký koncový stav α_f,
pro které platí $\alpha_0 \longrightarrow^* \alpha_f$:

- Algoritmus si bude průběžně pamatovat:
  - Aktuální stav α
  - Hodnotu čítače c — kolik kroků je ještě možné udělat
    (čítač c bude reprezentován binárně)
- Na začátku bude α inicialializováno nějakým nedeterministicky zvoleným počátečním stavem α_0.

  Čítač bude inicializován nějakou hodnotou, která bude větší nebo rovna $|States| - 1$.

  Protože je stavů nanejvýš exponenciálně mnoho, bude stačit na reprezentaci hodnoty čítače polynomiální počet bitů.

Přechodové systémy

Algoritmus bude v cyklu provádět následující:

- Pokud bude α koncový stav, skončí s odpovědí Ano.
- Pokud bude mít čítač c hodnotu 0, skončí s odpovědí Ne.
- Nedeterministicky zvolí nějaký stav α' takový, aby platilo $\alpha \longrightarrow \alpha'$.
- Nastaví hodnotu α na α'.
- Sníží hodnotu čítače c o 1.
- Pokračuje další iterací.

Algoritmus tedy nedeterministicky hádá příslušnou cestu v grafu, přičemž si vždy pamatuje jen aktuální vrchol.

Je zjevné, že tento algoritmus vystačí s polynomiálním množstvím paměti.

Přechodové systémy

Můžeme se odvolat na Savitchovu větu a vyvodit, že tedy bude existovat i **deterministický** algoritmus s polynomiální prostorovou složitostí řešící daný problém.

Alternativně je také možné rovnou sestrojit **deterministický** algoritmus, který bude založen na stejné myšlence, na které je založen důkaz Savitchovy věty.

Toto řešení se tedy na Savitchovu větu nebude odvolávat.

Základem tohoto algoritmu bude rekurzivní funkce

$$\textsc{Reach}(i, \alpha, \beta)$$

která bude vracet boolevskou hodnotu:

- vrátí $\textsc{True}$ právě tehdy, když bude existovat cesta z α do β délky nejvýše 2^i

Přechodové systémy

```
REACH (i, α, β):
    if i = 0 then
        if (α = β or α ⟶ β) then return True
    else
        for each γ ∈ States do
            if REACH(i − 1, α, γ) and REACH(i − 1, γ, β) then
                return True
    return False
```

Stačí tedy zavolat funkci $\textsc{Reach}(h, \alpha, \beta)$ pro všechny počáteční stavy α a všechny koncové stavy β, přičemž hodnota h bude číslo, které bude splňovat podmínku

$$2^h \geq |States| - 1$$

Přechodové systémy

Je zřejmé, že:

- Pro každé volání funkce $\textsc{Reach}$ bude stačit polynomiální množství paměti pro uložení hodnot jejích argumentů a lokálních proměnných.
- Počáteční hodnota proměnné i, tedy hodnota h, je v $\mathcal{O}(\log |States|)$.

  Hlouka zanoření rekurzivních volání funkce $\textsc{Reach}$ bude rovna h a bude tedy také nejvýše $\mathcal{O}(\log |States|)$.

  Tato hodnota je nejvýše polynomiální.

Vidíme tedy, že celý tento deterministický algoritmus má polynomiální prostorovou složitost.

Oblázková hra

Uvažujme následující **oblázkovou hru** (**pebble game**):

- Máme dán acyklický orientovaný graf $G = (V, E)$, ve kterém je jeden vrchol vyznačen jako cílový (označme tento vrchol t).
- Máme k dispozici k kamenů. Tyto kameny mohou být pokládány na vrcholy grafu.
- Na začátku je graf prázdný.
- Je možné provádět kroky podle následujících pravidel:
  - Pokud je vrchol v prázdný a na na všech předchůdcích vrcholu v (tj. na všech vrcholech v' takových, že existuje hrana z v' do v) leží kameny, je možné položit kámen na vrchol v.
  - Pokud je vrchol v prázdný a na na všech předchůdcích vrcholu v leží kameny, je možné posunout kámen z jednoho z těchto předchůdců na vrchol v.
  - Libovolný kámen ležící na nějakém vrcholu grafu G je možné odebrat.

Oblázková hra

- Cílem je najít posloupnost tahů takovou, aby na jejím konci byl položen kámen na cílový vrchol t, nebo zjistit, že žádná taková posloupnost neexistuje.

Poznámka: Nejde v pravém slova smyslu o hru, protože tady nejsou dva hráči, kteří by hráli proti sobě. Je to spíše něco jako hlavolam.

Můžeme zformulovat následující rozhodovací problém:

Oblázková hra

Vstup: Acyklický orientovaný graf G s vyznačeným cílovým vrcholem t a počet kamenů k.

Otázka: Existuje nějaká posloupnost tahů, kde na konci této posloupnosti bude položen kámen na vrchol t?

Oblázková hra

Na oblázkovou hru se můžeme dívat jako na přechodový systém, kde:

- **Stavy** odpovídají všem možnostem, jak může být rozmístěno 0 až k kamenů na n vrcholech grafu G.
- **Přechody** jsou dány pravidly určujícími, jak je možné pokládat, posouvat nebo odebírat kameny.
- **Počáteční stav** odpovídá situaci, kdy na grafu G nejsou položeny žádné kameny.
- **Koncové stavy** jsou ta rozmístění kamenů, kdy leží kámen na vrcholu t.

Oblázková hra

- Stavy je možné reprezentovat jako n-tice bitů, kde hodnota i-tého bitu udává, zda na vrcholu s číslem i leží nebo neleží kámen.

  (Předpokládáme, že vrcholy jsou označeny čísly $0, 1, \ldots, n-1$.)
- Je zřejmé, že následující testy je možné provést v čase úměrném velikosti grafu G:
  - test, zda jsou stavy α a β totožné (tj. zda platí $\alpha = \beta$)
  - test, zda je možné jedním krokem přejít ze stavu α do stavu β
  - test, zda α je počáteční stav
  - test, zda α je koncový stav

Vidíme tedy, že platí následující:

Věta

Problém „Oblázková hra“ je v PSPACE.

Ekvivalence nedeterministických konečných automatů

Ekvivalence NKA

Vstup: Nedeterministické konečné automaty $\mathcal{A}_1$ a $\mathcal{A}_2$.

Otázka: Je $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$?

Tento problém je možné řešit takto:

- k daným **nedeterministickým** konečným automatům $\mathcal{A}_1$ a $\mathcal{A}_2$ sestrojíme ekvivalentní **deterministické** konečné automaty $\mathcal{A}'_1$ a $\mathcal{A}'_2$ a pro tyto deterministické konečné automaty budeme hledat rozlišující slovo, které jeden z nich přijme a druhý ne.

  Pokud takové rozlišující slovo existuje, bude platit $\mathcal{L}(\mathcal{A}_1) \neq \mathcal{L}(\mathcal{A}_2)$ (a odpověď tedy bude NE), pokud ne, tak bude platit $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$ (a odpověď tedy bude ANO).

Ekvivalence nedeterministických konečných automatů

Při převodu nedeterministického konečného automatu na deterministický může dojít k exponenciálnímu nárůstu počtu stavů:

- Stavy deterministického automatu totiž odpovídají **podmnožinám** množiny stavů původního nedeterministického automatu a těch může být až 2^n, kde n je počet stavů původního automatu.

Výše uvedený postup má tedy exponenciální prostorovou složitost.

Ekvivalence nedeterministických konečných automatů

Exponenciálně velké deterministické automaty $\mathcal{A}'_1$ a $\mathcal{A}'_2$ však není třeba držet celé v paměti.

Navíc můžeme při hledání rozlišujícího slova využít nedeterminismus:

- Algoritmus si bude pamatovat vždy jen aktuální stav deterministických automatů $\mathcal{A}'_1$ a $\mathcal{A}'_2$
  (kde první z nich bude podmnožina množiny stavů automatu $\mathcal{A}_1$ a druhý podmnožina množiny stavů automatu $\mathcal{A}_2$).
- Algorimus bude nedeterministicky hádat jednotlivé symboly rozlišujícího slova.

  V rámci jednoho kroku vždy nedeterministicky uhodne následující symbol a odsimuluje jeho přečtení na stavech automatů $\mathcal{A}'_1$ a $\mathcal{A}'_2$.

Ekvivalence nedeterministických konečných automatů

Je zjevné, že tento nedeterministický algoritmus vystačí s polynomiálním množstvím paměti.

Použitím Savitchovy věty jej můžeme převést na deterministický algoritmus, který také bude mít polynomiální prostorovou složitost.

Dostáváme tedy následující:

Věta

Problém „Ekvivalence NKA“ je v PSPACE.

Poznámka: Alternativně je také možné se neodvolávat na Savitchovu větu a místo toho přímo sestrojit příslušný deterministický algoritmus s polynomiální prostorovou složitostí.

Ekvivalence nedeterministických konečných automatů

Z předchozího je zřejmé, že i následující problém (na který se můžeme dívat jako na speciální případ problému „Ekvivalence NKA“) patří do třídy PSPACE:

Univerzalita NKA

Vstup: Nedeterministický konečný automat $\mathcal{A}$.

Otázka: Je $\mathcal{L}(\mathcal{A}) = \Sigma^*$?

Tento problém je navíc PSPACE-**těžký** a tedy PSPACE-**úplný**.

Z toho okamžitě dostáváme, že i problém „Ekvivalence NKA“ je PSPACE-**úplný**.

Věta

Problémy „Ekvivalence NKA“ a „Univerzalita NKA“ jsou oba PSPACE-úplné.

Ekvivalence nedeterministických konečných automatů

Důkaz PSPACE-obtížnosti problému „Univerzalita NKA“ je možné provést tak, že se ukáže, jak pro každý problém A z třídy PSPACE sestrojit polynomiální redukci z A na doplňkový problém problému „Univerzalita NKA“:

- Předpokládejme, že problém A patří do třídy PSPACE.
- Existuje tedy jednopáskový deterministický Turingův stroj $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, který jej řeší, a polynom $p(n)$ takový, že když stroj $\mathcal{M}$ dostane jako vstup slovo w velikosti n, tak všechny konfigurace, kterými stroj $\mathcal{M}$ při výpočtu nad w projde, budou velikosti nejvýše $p(n)$.

  Tyto konfigurace můžeme zapisovat jako slova nad abecedou $\Delta' = \Gamma \cup (Q \times \Gamma)$ velikosti přesně $p(n)$.

Ekvivalence nedeterministických konečných automatů

- Výpočet stroje $\mathcal{M}$ nad slovem w pak můžeme reprezentovat jako slovo vzniklé zřetezením slov reprezentujících jednotlivé konfigurace, přičemž tyto konfigurace budou odděleny pomocí speciálního znaku #.
- Stroj $\mathcal{M}$ dá pro vstup w odpověď ANO právě tehdy, když jeho výpočet nad tímto slovem skončí v **přijímající** konfiguraci, kde stav řídící jednotky bude přijímající koncový stav q_{acc}.
- K danému Turingovu stroji $\mathcal{M}$ a slovu w tak můžeme zkonstruovat nedeterministický konečný automat $\mathcal{A}$, který příjme právě ta slova nad abecedou $\Delta = \Delta' \cup \{\#\}$, která **nejsou** zápisem přijímajícího výpočtu stroje $\mathcal{M}$ nad slovem w:
  - pokud $\mathcal{M}$ přijímá w, bude $\mathcal{L}(\mathcal{A}) \neq \Delta^*$
  - pokud $\mathcal{M}$ nepřijímá w, bude $\mathcal{L}(\mathcal{A}) = \Delta^*$

Ekvivalence nedeterministických konečných automatů

Slova nad abecedou Δ, která **nejsou** korektním zápisem přijímajícího výpočtu stroje $\mathcal{M}$ nad slovem w, kde každá konfigurace je zapsána slovem délky $p(n)$, vypadají následovně:

- nemají správný formát, tj. nemají podobu posloupnosti konfigurací, kde má každá konfigurace délku $p(n)$, nebo
- nezačínají počáteční konfigurací stroje $\mathcal{M}$ nad vstupem w, nebo
- obsahují nějaké dvě po sobě jdoucí konfigurace α_i a α_{i+1}, které neodpovídají tomu, jaký krok by udělal stroj $\mathcal{M}$ v konfiguraci α_i, nebo
- nekončí konfigurací, ve které by byl stav řídící jednotky q_{acc}.

Všechno jsou to vlastnosti, které je možno snadno rozpoznávat nedeterministickým konečným automatem $\mathcal{A}$, jehož velikost je vzhledem k délce slova w (které je vstupem Turingova stroje $\mathcal{M}$) polynomiální (za předpokladu, že $p(n)$ je polynom).

Ekvivalence regulárních výrazů

Velice podobným způsobem je možné dokázat PSPACE-úplnost následujících dvou problémů.

Ekvivalence regulárních výrazů

Vstup: Regulární výrazy α_1 a α_2.

Otázka: Je $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Univerzalita regulárních výrazů

Vstup: Regulární výraz α.

Otázka: Je $\mathcal{L}(\alpha) = \Sigma^*$?

Ekvivalence regulárních výrazů

- Pro důkaz toho, že tyto problémy jsou v PSPACE, si stačí uvědomit, že k danému regulárnímu výrazu α je možné v polynomiálním čase (a tedy i v polynomiálním prostoru) sestrojit nedeterministický konečný automat $\mathcal{A}$ takový, že $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\alpha)$.

  Stačí tedy převést dané výrazy na nedeterministické konečné automaty a použít příslušné dříve popsané algoritmy, které pracují s nedeterministickými konečnými automaty a mají polynomiální prostorovou složitost.

- Při důkazu PSPACE-obtížnosti se konstruuje regulární výraz, který popisuje jazyk tvořený právě těmi slovy, která nejsou korektním zápisem přijímajícího výpočtu Turingova stroje $\mathcal{M}$ nad slovem w.

Ekvivalence regulárních výrazů

Regulární výrazy s mocněním jsou definovány podobně jako běžné regulární výrazy, ale kromě operátorů $+$, $\cdot$ a * mohou navíc obsahovat unární operátor ^{2} s následujícím významem:

- α^2 je zkratkou pro $\alpha \cdot \alpha$.

Následující dva problémy jsou EXPSPACE-úplné:

Ekvivalence regulárních výrazů s mocněním

Vstup: Regulární výrazy s mocněním α_1 a α_2.

Otázka: Je $\mathcal{L}(\alpha_1) = \mathcal{L}(\alpha_2)$?

Univerzalita regulárních výrazů s mocněním

Vstup: Regulární výraz s mocněním α.

Otázka: Je $\mathcal{L}(\alpha) = \Sigma^*$?

Ekvivalence regulárních výrazů

- To, že jsou tyto problémy ve třídě EXPSPACE se dá ukázat následovně:

  V regulárním výrazu s mocněním je možné podvýrazy tvaru α^2 nahradit ekvialentním výrazem $\alpha \cdot \alpha$.
  Tato úprava zvětší velikost výrazu nanejvýš exponenciálně.

  Pokud tedy použijeme dříve popsané algoritmy s polynomiální prostorovou složitostí na exponenciálně velké regulární výrazy vzniklé touto transformací, dostaneme algoritmus s exponenciální prostorovou složitostí.

Ekvivalence regulárních výrazů

- Důkaz EXPSPACE-obtížnosti je velmi podobný jako důkaz PSPACE-obtížnosti pro regulární výrazy bez mocnění.

  Ukáže se, jak vytvořit redukci pro každý problém A ze třídy EXPSPACE.

  Pokud je problém A v EXPSPACE, existuje jednopáskový deterministický Turingův stroj $\mathcal{M}$, který ho řeší, a který má prostorovou složitost omezenou shora funkcí tvaru $2^{p(n)}$, kde $p(n)$ je nějaký polynom.

  Výpočet tohoto stroje $\mathcal{M}$ nad slovem w délky n je možné popsat jako posloupnost konfigurací, kde každá konfigurace může být zapsána jako slovo nad abecedou Δ délky $2^{p(n)}$.

  Opět se vytvoří regulární výraz (tentokrát s mocněním) popisující právě ta slova, která nejsou zápisem přijímajícího výpočtu stroje $\mathcal{M}$ nad slovem w.

Ekvivalence regulárních výrazů

Polynomiálně velkými podvýrazy tvaru

$$((\cdots(((\alpha^2)^2)^2)\cdots)^2)^2$$

je možné popisovat exponenciálně dlouhé podúseky znaků vyskytující se mezi dvojicemi vzájemně si odpovídajících trojic znaků ve dvou po sobě jdoucích konfiguracích.

Kvantifikované booleovské formule (QBF)

Kvantifikované booleovské formule (**Quantified Boolean Formulas** — **QBF**) jsou zobecněním běžných booleovských formulí.

Tyto formule jsou definovány následovně:

- Formule tvaru x, kde x je booleovská proměnná, je dobře utvořená formule.
- Formule tvaru $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$, kde φ_1 a φ_2 jsou dobře utvořené formule, jsou dobře utvořené formule
- Formule tvaru $\exists x.\varphi_1$ a $\forall x.\varphi_1$, kde x je booleovská proměnná a φ_1 je dobře utvořená formule, jsou dobře utvořené formule.

Příklad:

$$\exists x_1.\forall x_2.\exists x_3.((\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_3))$$

Kvantifikované booleovské formule (QBF)

- Booleovské proměnné nabývají hodnot z množiny $\mathrm{Bool} = \{0, 1\}$.
- Zápis $\exists x.\varphi$ je možné chápat jako zkrácený způsob, jak vyjádřit formuli tvaru

$$\varphi[0/x] \vee \varphi[1/x]$$

  a podobně zápis $\forall x.\varphi$ je možné chápat jako zkrácený způsob, jak vyjádřit formuli tvaru

$$\varphi[0/x] \wedge \varphi[1/x]$$

  kde zápis $\varphi[b/x]$ označuje formuli, ve které jsou volné výskyty proměnné x nahrazeny booleovskou konstantou b.
- Význam logických spojek $\neg$, $\wedge$, $\vee$, $\rightarrow$ a $\leftrightarrow$ je stejný jako ve výrokové logice.

Kvantifikované booleovské formule (QBF)

- Na kvantifikované booleovské formule se také můžeme dívat jako na speciální případ formulí predikátové logiky, kde proměnné nabývají hodnot z univerza $\mathrm{Bool} = \{0, 1\}$.
- Formule φ je **uzavřená**, jestliže neobsahuje žádné volné proměnné, tj. jestliže každý výskyt booleovské proměnné ve formuli je vázán kvantifikátorem.
- Pravdivostní hodnota formule φ závisí na přiřazení pravdivostních hodnot všem booleovským proměnným, které mají ve φ volné výskyty.

  V případě uzavřených formulí je tedy jejich pravdivostní hodnota nezávislá na aktuálním přiřazení pravdivostních hodnot proměnným.
- O uzavřené formuli φ řekneme, že je **pravdivá**, jestliže nabývá pravdivostní hodnoty 1.

Kvantifikované booleovské formule (QBF)

Uvažujme následující rozhodovací problém:

Problém kvantifikovaných booleovských formulí (QBF)

Vstup: Kvantifikovaná booleovská formule φ.

Otázka: Je formule φ pravdivá?

Poznámka: V literatuře se tento problém také někdy označuje následujícími názvy:

- **TQBF** — True Quantified Boolean Formulas
- **QSAT** — Quantified Satisfiability

Kvantifikované booleovské formule (QBF)

- Na problém **splnitelnosti** booleovských formulí (**SAT**) se můžeme dívat jako na speciální případ problému QBF:

  Řekněme, že φ je obyčejná booleovská formule bez kvantifikátorů, a že x_1, x_2, ..., x_n jsou všechny booleovské proměnné, které se ve formuli φ vyskytují.

  Je zjevné, že formule φ je splnitelná právě tehdy, když je pravdivá následující kvantifikovaná booleovská formule:

  $$\exists x_1.\exists x_2.\cdots.\exists x_n.\varphi$$

- Podobně to, že (obyčejná) booleovská formule φ je **tautologií**, je možné vyjádřit následující kvantifikovanou booleovskou formulí:

  $$\forall x_1.\forall x_2.\cdots.\forall x_n.\varphi$$

Kvantifikované booleovské formule (QBF)

Není těžké si rozmyslet, že problém QBF patří do třídy PSPACE:

Jádrem algoritmu je rekurzivně definovaná funkce

$$\textsc{Eval}(\varphi, \nu)$$

která jako argumenty dostane:

- φ — kvantifikovanou booleovskou formuli, která se má vyhodnotit
- ν — pravdivostní ohodnocení, které přiřazuje hodnoty všem booleovským proměnným, které se mohou vyskytovat jako volné proměnné ve formuli φ

Poznámka: Zápisem

$$\nu[x \mapsto b]$$

budeme označovat pravdivostní ohodnocení, které je stejné jako pravdivostní ohodnocení ν, až na to, že proměnné x je přiřazena pravdivostní hodnota b.

Kvantifikované booleovské formule (QBF)

Funkce $\textsc{Eval}(\varphi, \nu)$ rozebírá jednotlivé případy na základě toho, jakého tvaru je formule φ.

Pokud je φ tvaru:

- x: **return** $(\nu(x))$
- $\neg\varphi_1$: **return** (**not** $\textsc{Eval}(\varphi_1, \nu)$)
- $\varphi_1 \wedge \varphi_2$: **return** ($\textsc{Eval}(\varphi_1, \nu)$ **and** $\textsc{Eval}(\varphi_2, \nu)$)
- $\varphi_1 \vee \varphi_2$: **return** ($\textsc{Eval}(\varphi_1, \nu)$ **or** $\textsc{Eval}(\varphi_2, \nu)$)
- $\varphi_1 \rightarrow \varphi_2$: **return** ((**not** $\textsc{Eval}(\varphi_1, \nu)$) **or** $\textsc{Eval}(\varphi_2, \nu)$)
- $\varphi_1 \leftrightarrow \varphi_2$: **return** ($\textsc{Eval}(\varphi_1, \nu)$ **iff** $\textsc{Eval}(\varphi_2, \nu)$)
- $\exists x.\varphi_1$: **return** ($\textsc{Eval}(\varphi_1, \nu[x \mapsto 0])$ **or** $\textsc{Eval}(\varphi_1, \nu[x \mapsto 1])$)
- $\forall x.\varphi_1$: **return** ($\textsc{Eval}(\varphi_1, \nu[x \mapsto 0])$ **and** $\textsc{Eval}(\varphi_1, \nu[x \mapsto 1])$)

Kvantifikované booleovské formule (QBF)

- Je zjevné, že hloubka zanoření rekurzivních volání funkce EVAL je shora omezená velikostí formule φ.
- Celkové množství informací, které si musí algoritmus během výpočtu pamatovat, je očividně omezeno polynomem.

Problém QBF patří do třídy PSPACE.

Kvantifikované booleovské formule (QBF)

Ukažme nyní, že problém QBF je PSPACE-těžký:

- Je třeba ukázat, že pro každý problém A ze třídy PSPACE existuje polynomiální redukce problému A na QBF.
- Předpokládejme tedy, že A je problém ze třídy PSPACE, tj. že existuje nějaký algoritmus s polynomiální prostorovou složitostí, který ho řeší.
- Předpokládejme dále, že tento algoritmus je implentován ve formě nějakého stroje $\mathcal{M}$.

  Pro konkrétnost můžeme předpokládat například, že $\mathcal{M}$ je jednopáskový deterministický Turingův stroj.

  (Tato volba není příliš podstatná a algoritmus převodu by stejně dobře fungoval i pro jiné typy strojů.)
- Předpokládejme, že velikost konfigrací stroje $\mathcal{M}$ nad slovem w je omezena shora nějakým polynomem $p(n)$, kde n je délka slova w.

Kvantifikované booleovské formule (QBF)

- Konfigurace stroje $\mathcal{M}$ při výpočtu nad slovem w je možné „kódovat“ pomocí hodnot booleovských proměnných.

  Řekněme, že konfigurace α bude kódována pomocí hodnot booleovských proměnných

  $$x_1, x_2, \ldots, x_r,$$

  kde r je přirozené číslo, jehož hodnota je v $\mathcal{O}(p(n))$.

Kvantifikované booleovské formule (QBF)

- Jestliže je konfigurace α kódována pomocí proměnných $x_1, x_2, \ldots, x_r$, zápisy

$$\exists \alpha.\varphi \qquad\qquad \forall \alpha.\varphi$$

budeme chápat jako zkratky pro

$$\exists x_1.\exists x_2.\cdots.\exists x_r.\varphi \qquad\qquad \forall x_1.\forall x_2.\cdots.\forall x_r.\varphi$$

- Řekněme, že konfigurace α je kódována pomocí proměnných $x_1, x_2, \ldots, x_r$, a konfigurace β pomocí proměnných $y_1, y_2, \ldots, y_r$.

Formule $\mathrm{EQ}(\alpha, \beta)$ vyjadřuje, že α a β jsou jedna a tatáž konfigurace:

$$(x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \wedge \cdots \wedge (x_r \leftrightarrow y_r)$$

Kvantifikované booleovské formule (QBF)

Chování stroje $\mathcal{M}$ bude reprezentováno pomocí následujících tří formulí (kde konfigurace α je reprezentována např. pomocí proměnných $x_1, x_2, \ldots, x_r$ a konfigurace β pomocí proměnných $y_1, y_2, \ldots, y_r$)

— detaily toho, jak tyto formule vypadají, budou podobné jako například u důkazu NP-obtížnosti problému SAT:

- $\textsc{Step}(\alpha, \beta)$ — vyjadřuje, že stroj $\mathcal{M}$ může přejít jedním krokem z konfigurace α do konfigurace β
- $\textsc{Is-Init}(\alpha)$ — vyjadřuje, že α je počáteční konfigurace stroje $\mathcal{M}$ nad vstupem w
- $\textsc{Is-Acc}(\alpha)$ — vyjadřuje, že α je přijímající koncová konfigurace stroje $\mathcal{M}$.

Velikost všech těchto formulí bude polynomiální (nebo dokonce lineární) vzhledem k hodnotě $p(n)$.

Kvantifikované booleovské formule (QBF)

Jestliže předpokládáme, že konfigurace jsou kódovány r booleovskými proměnnými, je zřejmé, že počet navzájem různých konfigurací je nejvýše 2^r.

Během výpočtu stroje $\mathcal{M}$ se konfigurace nemohou opakovat — délka těchto výpočtů je tedy nejvýše 2^r.

Vytvoříme posloupnost formulí

$$\textsc{Reach}_0, \textsc{Reach}_1, \ldots, \textsc{Reach}_r$$

kde formule

$$\textsc{Reach}_i(\alpha, \beta)$$

kde $0 \leq i \leq r$, bude vyjadřovat, že stroj $\mathcal{M}$ může nejvýše 2^i kroky přejít z konfigurace α do konfigurace β.

Kvantifikované booleovské formule (QBF)

Tyto formule budou definovány induktivně:

- $\textsc{Reach}_0(\alpha, \beta)$: bude definována jako

$$\textsc{Eq}(\alpha, \beta) \vee \textsc{Step}(\alpha, \beta)$$

- $\textsc{Reach}_{i+1}(\alpha, \beta)$, kde $i \geq 0$:

  jednoduchá přímočará definice by mohla vypadat takto

$$\exists \gamma.(\textsc{Reach}_i(\alpha, \gamma) \ \wedge \ \textsc{Reach}_i(\gamma, \beta))$$

  Problém s tímto řešením je ten, že velikost formulí $\textsc{Reach}_i$ zde roste exponenciálně s hodnotou i.

  Ekvivalentně je možné vyjádřit stejnou věc následovně:

$$\exists \gamma.\forall \sigma_1.\forall \sigma_2.(\\ ((\textsc{Eq}(\sigma_1, \alpha) \wedge \textsc{Eq}(\sigma_2, \gamma)) \ \vee \ (\textsc{Eq}(\sigma_1, \gamma) \wedge \textsc{Eq}(\sigma_2, \beta))) \ \rightarrow \\ \textsc{Reach}_i(\sigma_1, \sigma_2))$$

Kvantifikované booleovské formule (QBF)

- Výsledná formule φ bude vypadat takto:

$$\exists\alpha.\exists\beta.(\textsc{Is-Init}(\alpha) \wedge \textsc{Is-Acc}(\beta) \wedge \textsc{Reach}_r(\alpha,\beta))$$

- Není těžké ověřit, že tato formule φ je pravdivá právě tehdy, když existuje nějaký přijímající výpočet stroje $\mathcal{M}$ nad slovem w.
- Rovněž není těžké ověřit, že velikost této formule je polynomiální vzhledem k délce slova w — tato velikost je $\mathcal{O}(p(n)^2)$, kde $n = |w|$.

  Tuto formuli je také možno snadno vyrobit v polynomiálním čase.

Vidíme tedy, že problém QBF je PSPACE-těžký.

Důkaz následujícího tvrzení je tedy hotov:

Věta

Problém QBF je PSPACE-úplný.

Kvantifikované booleovské formule (QBF)

Připomeňme, že pro dokazování NP-obtížnosti různých problémů se často používá problém SAT, kdy se NP-obtížnost problému A ukáže tak, že se ukáže polynomiální převod z problému SAT na A.

Podobně se problém QBF často používá pro dokazování PSPACE-**obtížnosti** dalších problémů:

- Abychom ukázali, že nějaký daný problém A je PSPACE-těžký, stačí ukázat polynomiální převod z QBF na A.

Kvantifikované booleovské formule (QBF)

Redukcí z QBF je například možné dokázat PSPACE-obtížnost (a tedy PSPACE-úplnost) dříve popsaného problému:

Oblázková hra

Vstup: Acyklický orientovaný graf G s vyznačeným cílovým vrcholem t a počet kamenů k.

Otázka: Existuje nějaká posloupnost tahů, kde na konci této posloupnosti bude položen kámen na vrchol t?

(Tato redukce je poměrně komplikovaná. Nebudeme ji zde proto detailně popisovat.)

Kvantifikované booleovské formule (QBF)

Při důkazech NP-obtížnosti se místo problému SAT (kde vstupem může být libovolná booleovská formule) často používá jeho varianta 3-SAT:

- předpokládá se, že formule je určitého specifického tvaru — je v konjuntivní normální formě, kde má každá klauzule právě 3 literály

Podobně, pokud se dokazuje PSPACE-obtížnost nějakého problému A tím způsobem, že se popisuje polynomiální redukce z QBF na A, tak se v daném důkaze často pro jednoduchost předpokládá, že formule, která je instancí QBF, je určitého specifického tvaru:

$$\mathcal{Q}_1\, x_1 .\, \mathcal{Q}_2\, x_2 .\, \cdots\, \mathcal{Q}_n\, x_n \,.\, \psi$$

- $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ jsou kvantifikátory ($\exists$, $\forall$), které se navíc pravidelně střídají, tj. $\mathcal{Q}_i = \exists$ pro liché i a $\mathcal{Q}_i = \forall$ pro sudé i
- podformule ψ neobsahuje žádné kvantifikátory a je v konjuntivní normální formě, kde má každá klauzule právě 3 literály

Kvantifikované booleovské formule (QBF)

Kvantifikované formule je možné pomocí **ekvivantních úprav** upravovat na ekvivalentní formule, které jsou nějakého specifického tvaru.

Například můžeme každou formuli φ převést na ekvivalentní formuli φ', která bude splňovat následující:

- z logických spojek obsahuje pouze spojky $\neg$, $\wedge$ a $\vee$ (tj. neobsahuje spojky $\leftrightarrow$ a $\rightarrow$):

$$\begin{aligned} A \leftrightarrow B &\iff (A \rightarrow B) \wedge (B \rightarrow A) \\ A \rightarrow B &\iff \neg A \vee B \end{aligned}$$

- negace ($\neg$) jsou aplikovány pouze na atomické formule, tj. na výrokové proměnné (např. $\neg x$):

$$\begin{aligned} \neg\exists x.A &\iff \forall x.\neg A \\ \neg\forall x.A &\iff \exists x.\neg A \end{aligned}$$

$$\begin{aligned} \neg\neg A &\iff A \\ \neg(A \wedge B) &\iff \neg A \vee \neg B \\ \neg(A \vee B) &\iff \neg A \wedge \neg B \end{aligned}$$

Kvantifikované booleovské formule (QBF)

Dále je možné danou kvantifikovanou formuli ϕ pomocí ekvivantních úprav transformovat do tzv. **prenexního tvaru**:

$$\mathcal{Q}_1\, x_1 .\, \mathcal{Q}_2\, x_2 .\, \cdots\, \mathcal{Q}_n\, x_n \,.\, \psi$$

kde $\mathcal{Q}_1$, $\mathcal{Q}_2$, ..., $\mathcal{Q}_n$ jsou kvantifikátory ($\exists$, $\forall$) a kde podformule ψ neobsahuje žádné kvantifikátory.

Pokud se ve formuli A nevyskytuje x jako volná proměnná, tak platí např.:

$$\begin{aligned} A \wedge \exists x.B &\iff \exists x.(A \wedge B) \\ A \vee \exists x.B &\iff \exists x.(A \vee B) \\ A \wedge \forall x.B &\iff \forall x.(A \wedge B) \\ A \vee \forall x.B &\iff \forall x.(A \vee B) \end{aligned}$$

(Výskyty proměnných je možné přejmenovat tak, aby se proměnné vázané různými kvantifikátory jmenovaly jinak.
Tím se zaručí, že se např. x nebude vyskytovat v A, jak je uvedeno výše.)

Kvantifikované booleovské formule (QBF)

Ve formuli v prenexním tvaru

$$\mathcal{Q}_1\, x_1 .\, \mathcal{Q}_2\, x_2 .\, \cdots\, \mathcal{Q}_n\, x_n\, .\, \psi$$

je možné podformuli ψ (která neobsahuje kvantifikátory) transformovat na podformuli tvaru

$$\exists y_1 . \exists y_2 . \cdots . \exists y_k . \psi'$$

kde:

- ψ' je v **konjuntivní normální formě**, kde každá klauzule bude obsahovat právě 3 literály
- $y_1, y_2, \cdots, y_k$ jsou nově zavedené pomocné proměnné odpovídající jednotlivým podformulím formule ψ

Udělá se to podobným způsobem jako při převodu problému SAT na problém 3-SAT.

Kvantifikované booleovské formule (QBF)

Formule v prenexním tvaru se dá také snadno upravit do tvaru, kdy se kvantifikátory **pravidelně střídají**:

- stačí přidat nové kvantifikátory s novými proměnnými, které nejsou nikde ve formuli použity

Příklad: Formuli tvaru

$$\forall x_1.\forall x_2.\forall x_3.\exists x_4.\exists x_5.\exists x_6.\forall x_7.\forall x_8.\,\psi$$

je možné převést na formuli

$$\exists y_1.\forall x_1.\exists y_2.\forall x_2.\exists y_3,\ \forall x_3.\exists x_4.\forall y_4.\exists x_5.\forall y_5\exists x_6.\forall x_7.\exists y_6.\forall x_8.\,\psi$$

kde $y_1, y_2, \ldots, y_6$ jsou nové dosud nepoužité proměnné.

Kvantifikované booleovské formule (QBF)

Všimněte si, že pro naprostou většinu těchto úprav platí, že se jejich provedením velikost dané formule příliš nemění:

- velikost výsledné formule φ' bude lineární vzhledem k velikosti původní formule φ
  (tj. pokud velikost původní formule φ je n, tak velikost výsledné formule φ' bude $\mathcal{O}(n)$)

Jedinou problematickou úpravou, kde by mohlo hrozit, že jejím opakovaným používáním by velikost formule mohla narůst až exponenciálně, je náhrada formule tvaru $A \leftrightarrow B$ formulí

$$(A \rightarrow B) \wedge (B \rightarrow A)$$

Ve formulích, které vznikají při konstrukci popsané v důkaze PSPACE-obtížnosti problému QBF, se logické spojka $\leftrightarrow$ vyskytovala pouze ve formulích $\mathrm{EQ}(\alpha, \beta)$, kde je však aplikována pouze na atomické formule. Výše uvedenou úpravou tedy velikost takových formulí vzroste také pouze lineárně.

Hry

Kvantifikované booleovské formule (QBF) jako hra

Na vyhodnocování kvantifikované booleovské formule φ je možné se dívat jako na určitý druh **hry**:

- Tuto hru hrají dva hráči — označme je **Hráč I** a **Hráč II**.
- *Hráč I* se snaží dokázat, že formule φ platí (tj. má pravdivostní hodnotu 1)
- *Hráč II* se snaží dokázat, že formule φ neplatí (tj. má pravdivostní hodnotu 0)

Jednotlivé **pozice** ve hře mají podobu dvojic (ψ, ν), kde:

- ψ — podformule původní formule φ
- ν — aktuální přiřazení booleovských hodnot (některým) proměnným

Kdo je v pozici (ψ, ν) na tahu a jaké tahy může provést, závisí na tvaru formule ψ.

Kvantifikované booleovské formule (QBF) jako hra

Předpokládejme, že formule φ je tvaru, kdy:

- z logických spojek může obsahovat pouze $\neg$, $\wedge$ a $\vee$ (tj. neobsahuje spojky $\rightarrow$ a $\leftrightarrow$)
- negace jsou aplikovány pouze na atomické formule, tj. negace se vyskytují pouze v podformulích tvaru $\neg x$, kde x je booleovská proměnná
- může obsahovat kvantifikátory $\exists$ a $\forall$

Příklad:

$$\forall x.\exists y.((\neg x \vee y) \wedge \forall z.((x \wedge (\neg y \vee z)) \vee \exists w.(\neg z \wedge \neg w)))$$

Poznámka: Ve skutečnosti by bylo možné nadefinovat i variantu hry, která by fungovala pro kvantifikované booleovské formule jakéhokoli tvaru. Bylo by to ale drobně technicky komplikovanější.
Pro jednoduchost se tedy omezíme na formule výše uvedeného tvaru.

Kvantifikované booleovské formule (QBF) jako hra

- Hra na formuli φ začíná v pozici (φ, ν_0), kde ν_0 je přiřazení, kde žádné proměnné není přiřazena hodnota.
- Pokud je v aktuální pozici (ψ, ν) formule ψ tvaru:
  - $\psi_1 \vee \psi_2$ — *Hráč I* zvolí jednu z formulí ψ_1 nebo ψ_2 a pokračuje se v pozici (ψ_1, ν) nebo (ψ_2, ν) podle této volby
  - $\psi_1 \wedge \psi_2$ — *Hráč II* zvolí jednu z formulí ψ_1 nebo ψ_2 a pokračuje se v pozici (ψ_1, ν) nebo (ψ_2, ν) podle této volby
  - $\exists x.\psi_1$ — *Hráč I* zvolí booleovskou hodnotu $b \in \{0, 1\}$, která se přiřadí proměnné x, a pokračuje se v pozici $(\psi_1, \nu[x \mapsto b])$
  - $\forall x.\psi_1$ — *Hráč II* zvolí booleovskou hodnotu $b \in \{0, 1\}$, která se přiřadí proměnné x, a pokračuje se v pozici $(\psi_1, \nu[x \mapsto b])$
  - x — hra končí: pokud $\nu(x) = 1$, vyhrál *Hráč I*, jinak vyhrál *Hráč II*
  - $\neg x$ — hra končí: pokud $\nu(x) = 0$, vyhrál *Hráč I*, jinak vyhrál *Hráč II*

Kvantifikované booleovské formule (QBF) jako hra

Strategie daného hráče je předpis, jak má tento hráč v dané hře hrát, tj. jaký konkrétní tah má zvolit v každé z pozic, kde je tahu.

Vyhrávající strategie daného hráče je taková strategie, která mu zaručí, že vždy vyhraje.
(Tj. při použití dané strategie vždy vyhraje bez ohledu na to, jaké tahy bude volit jeho protihráč.)

Chtěli bychom ukázat, že platí následující:

Tvrzení

Hráč I má ve výše popsané hře na formuli φ vyhrávající strategii právě tehdy, když je formule φ pravdivá.

Kombinatorické hry

Výše popsaná hra je speciálním případem **kombinatorické hry**.

Definice

Kombinatorickou hru je možné popsat jako čtveřici $\mathcal{G} = (Pos, Moves, lab, \alpha_0)$, kde:

- Pos — množina **pozic**
- $Moves \subseteq Pos \times Pos$ — množina **tahů**
- $lab : Pos \to \{\mathrm{I}, \mathrm{II}\}$ — přiřazení pozic hráčům
- $\alpha_0 \in Pos$ — počáteční pozice

Pro pozice $\alpha, \beta \in Pos$ to, že $(\alpha, \beta) \in Moves$, znamená, že v dané hře je možné provést tah z pozice α do pozice β. Budeme to označovat zápisem

$$\alpha \longrightarrow \beta$$

Kombinatorické hry

- Přiřazení pozic hráčům pomocí funkce *lab* určuje, který z hráčů je v dané pozici na tahu:
  - pokud $lab(\alpha) = \mathrm{I}$, je v pozici α na tahu *Hráč I*
  - pokud $lab(\alpha) = \mathrm{II}$, je v pozici α na tahu *Hráč II*
- Hra začíná v pozici α_0.
- Pozice, ve kterých neexistuje žádný možný tah (tj. pozice α, kde neexistuje žádné β takové, že $\alpha \longrightarrow \beta$) jsou **koncové pozice**.
- Jestliže pozice α není koncová (tj. pokud existuje alespoň jedno β takové, že $\alpha \longrightarrow \beta$), provede hráč, který je v pozici α na tahu, tah, který spočívá v tom, že vybere některé β takové, že $\alpha \longrightarrow \beta$.
  Hra pak pokračuje v pozici β.
- Hráč, který je na tahu v koncové pozici (a nemůže tedy táhnout), prohrál a jeho protihráč vyhrál. Hra tímto končí.

Kombinatorické hry

Zápisem $succ(\alpha)$ označme množinu všech pozic, do kterých je možný tah z pozice α, tj.

$$succ(\alpha) = \{ \beta \in Pos \mid \alpha \longrightarrow \beta \}$$

Pomocná označení některých důležitých množin pozic:

- $Pos_{\mathrm{I}} = \{\alpha \in Pos \mid lab(\alpha) = \mathrm{I} \wedge succ(\alpha) \neq \emptyset\}$
  — množina (nekoncových) pozic, kde je na tahu *Hráč I*
- $Pos_{\mathrm{II}} = \{\alpha \in Pos \mid lab(\alpha) = \mathrm{II} \wedge succ(\alpha) \neq \emptyset)\}$
  — množina (nekoncových) pozic, kde je na tahu *Hráč II*
- $Win_{\mathrm{I}} = \{\alpha \in Pos \mid lab(\alpha) = \mathrm{II} \wedge succ(\alpha) = \emptyset)\}$
  — množina koncových pozic, kde vyhrál *Hráč I*
- $Win_{\mathrm{II}} = \{\alpha \in Pos \mid lab(\alpha) = \mathrm{I} \wedge succ(\alpha) = \emptyset\}$
  — množina koncových pozic, kde vyhrál *Hráč II*

Kombinatorické hry

Partie je konečná nebo nekonečná posloupnost pozic

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots$$

kde:

- α_0 je počáteční pozice
- v každé z pozic α_i (s výjimkou poslední pozice v konečných partiích) vybral pozici α_{i+1} hráč určený hodnotou $lab(\alpha_i)$

V případě konečných partií je poslední pozice α_n koncová pozice:

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots \longrightarrow \alpha_{n-1} \longrightarrow \alpha_n$$

Vítěz takovéto konečné partie je určen podle toho, zda tato koncová pozice α_n patří do Win_{I} nebo do Win_{II}.

U nekonečných partií není vítězem žádný z hráčů.

Kombinatorické hry

Průběhy všech potenciálně možných partií je možné znázornit ve formě **stromu**:

- Kořen stromu je označen počáteční pozicí α_0.
- Pro všechny vrcholy stromu platí, že jestliže je vrchol označen pozicí α, tak potomci tohoto vrcholu jsou označeni pozicemi z množiny $succ(\alpha)$.
- Listy stromu jsou označeny koncovými pozicemi.
- Každá z větví tohoto stromu reprezentuje jednu možnost toho, jak může proběhnout partie v dané hře.

Poznámka: Z formálního hlediska bychom mohli nadefinovat tento strom následovně:

- Vrcholy stromu odpovídají neprázdným konečným posloupnostem tvaru:

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots \longrightarrow \alpha_{i-1} \longrightarrow \alpha_i$$

- Ve stromě vede hrana z vrcholu odpovídajícího posloupnosti

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots \longrightarrow \alpha_{i-1} \longrightarrow \alpha_i$$

do vrcholu odpovídajícího posloupnosti

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots \longrightarrow \alpha_{i-1} \longrightarrow \alpha_i \longrightarrow \alpha_{i+1}$$

právě tehdy, když $\alpha_i \longrightarrow \alpha_{i+1}$.

Kombinatorické hry

Zaměřme se nejprve na jednodušší případ, kdy všechny větve stromu jsou **konečné**:

- tj. když pravidla hry zaručují, že každá partie skončí po konečném počtu kroků vítězstvím jednoho z hráčů, bez ohledu na to, jaké tahy hráči volí.

Cílem je určit, které vrcholy stromu odpovídají situacím, které jsou:

- vítězné pro *Hráče I*
- vítězné pro *Hráče II*

Tj. ve kterých vrcholech už si daný hráč dokáže zajistit své vítězství bez ohledu na to, jak v následujících tazích bude hrát jeho protihráč.

Kombinatorické hry

Při této analýze můžeme postupovat od koncových pozic (tj. od listů stromu) směrem k počáteční pozici (tj. ke kořeni stromu):

- Koncové pozice, ve kterých vyhrál *Hráč* I, tj. pozice z množiny Win_{I}, jsou zcela jistě vítězné pro *Hráče* I.
- Koncové pozice, ve kterých vyhrál *Hráč* II, tj. pozice z množiny Win_{II}, jsou zcela jistě vítězné pro *Hráče* II.
- Řekněme, že máme dán vrchol v označený pozicí $\alpha \in Pos_{\mathrm{I}}$, tj. pozicí, která není koncová a kde je na tahu *Hráč* I.

  Řekněme dále, že pro všechny jeho potomky (označené pozicemi z množiny $succ(\alpha)$) už bylo určeno, zda jsou vítězné pro *Hráče* I nebo *Hráče* II.

  - Vrchol v je vítězný pro *Hráče* I právě tehdy, pokud má alespoň jednoho potomka, který je vítězný pro *Hráče* I.
    — *Hráč* I ve své strategii ve vrcholu v zvolí tohoto pro něj vítězného potomka.

Kombinatorické hry

- Uvažujme nyní případ, kdy vrchol v je označen pozicí α z množiny Pos_{II}, tj. pozicí, která není koncová a ve které je na tahu *Hráč II*.

  Každý takový vrchol bude vítězný pro *Hráče I* právě tehdy, pokud **všichni** jeho potomci budou vítězní pro *Hráče I*.

  Pokud jsou totiž všichni potomci vítězní pro *Hráče I*, *Hráč II* sice může zvolit tah, jaký chce, ale vždy se dostane do pozice, kde *Hráč I* zaručeně vyhraje.

  Naopak, pokud některý z potomků není vítězný pro *Hráče I*, *Hráč II* může zahrát právě do tohoto potomka.

- Analogicky můžeme určit, které pozice budou vítězné pro *Hráče II*.

Kombinatorické hry

Všimněme si několika věcí:

- To, zda je daný vrchol v vítězný pro daného hráče, závisí výhradně na podstromu, jehož kořenem je vrchol v.
- V daném stromě se může stejná pozice α vyskytovat vícekrát. Pokud si ale vezmeme libovolné dva vrcholy v a v' označené stejnou pozicí α, tak podstromy, jejichž kořeny budou v a v', budou zcela identické.
- To, zda je daný vrchol v vítězný pro daného hráče, tedy závisí čistě na pozici α, kterou je tento vrchol označen.

  Dává tedy smysl, mluvit o tom, že pozice α je **vítězná** pro *Hráče I* nebo pro *Hráče II*, resp. o tom, že daný hráč má v pozici α **vítěznou strategii**.
- Každý vrchol stromu bude vítězný pro právě jednoho z hráčů.

Kombinatorické hry

Vidíme tedy, že platí následující:

- Pozice $\alpha \in \mathit{Pos}_{\mathrm{I}}$ je vítězná pro *Hráče* I právě tehdy, když **existuje** nějaká pozice $\beta \in \mathit{succ}(\alpha)$, která je vítězná pro *Hráče* I.
- Pozice $\alpha \in \mathit{Pos}_{\mathrm{II}}$ je vítězná pro *Hráče* I právě tehdy, když **každá** pozice $\beta \in \mathit{succ}(\alpha)$ je vítězná pro *Hráče* I.

Kombinatorické hry

U kombinatorických her můžeme uvažovat problémy následujícího typu:

Vstup: Popis nějaké hry a nějaké pozice α.

Otázka: Má *Hráč I* (případně *Hráč II*) v pozici α vyhrávající strategii?

Jedna možnost je, že pozice a tahy mohou být popsány **explicitně**:

- jednotlivé pozice a tahy budou přímo vyjmenovány

Často ale mohou být pozice a tahy dány **implicitně**:

- máme dán obecný popis pravidel dané hry a popis toho, jak mohou vypadat jednotlivé pozice a tahy

Kombinatorické hry

Aby bylo možné při takovém **implicitním** způsobu popisu dané hry výše uvedený problém řešit algoritmicky, předpokládáme zde následující:

- Že jednotlivé pozice je možné reprezentovat nějakým konečným způsobem.
- Že pro každou pozici se dá nějak snadno algoritmicky určit, zda patří do Pos_{I}, Pos_{II}, Win_{I} nebo Win_{II}.
- Že počet prvků v $succ(\alpha)$ je pro libovolnou pozici α vždy konečný a že tyto prvky je možné algoritmicky generovat.

Poznámka: Připomeňme také, že se zde zatím omezujeme na hry, kde platí, že všechny partie jsou **konečné**.

Za výše uvedených podmínek z toho vyplývá, že strom reprezentující možné průběhy partií je konečný.

Kombinatorické hry

Dříve uvedené úvahy vedou k následujícímu jednoduchému rekurzivnímu algoritmu, který pro pozici α určí, zda je vítězná pro *Hráče* I:

Algoritmus: Určení pozic vítězných pro *Hráče* I

```
F(α):
    if α ∈ Win_I then return True
    else if α ∈ Win_II then return False
    else if α ∈ Pos_I then
        for each β ∈ succ(α) do
            if F(β) then return True
        return False
    else (tj. když α ∈ Pos_II)
        for each β ∈ succ(α) do
            if not F(β) then return False
        return True
```

Kombinatorické hry

Pokud množství paměti nutné pro uložení jedné pozice bude $\mathcal{O}(f(n))$ a délka všech větví stromu bude $\mathcal{O}(g(n))$, tak prostorová složitost výše uvedeného algoritmu bude $\mathcal{O}(f(n) \cdot g(n))$.

Uvažujme speciální případ, kdy bude navíc platit následující:

- Množství paměti potřebné pro uložení jedné pozice bude polynomiální.
- Na zjištění typu pozice a na generování všech prvků z $succ(\alpha)$ bude postačovat polynomiální množství paměti.
- Délka všech větví stromu (tj. délka všech partií) bude shora omezena nějakým polynomem.

Je očividné, že za těchto předpokladů bude prostorová složitost výše uvedeného algoritmu **polynomiální**.

Příklady PSPACE-úplných problémů týkajících se her

Uvedeme si tři příklady her, kde se dá takovýto algoritmus použít k tomu, aby se ukázalo, že výše popsaný problém určit, zda má daný hráč v dané pozici vyhrávající strategii, patří do třídy PSPACE.

Poznámka: Ve všech těchto třech příkladech her je daný problém navíc i PSPACE-těžký.
Všechno to tedy budou příklady PSPACE-úplných problémů.

Kvantifikované booleovské formule (QBF) jako hra

První příklad takové hry už jsme viděli:

QBF jako hra

Vstup: Kvantifikovaná booleovská formule φ.

Otázka: Má *Hráč* I ve hře na formuli φ vítěznou strategii?

- Předpokládáme, že formule φ splňuje dříve uvedená omezení (tj. že z logických spojek mohou být použity pouze $\neg$, $\wedge$ a $\vee$, a že negace jsou aplikovány pouze na atomické formule).

  Pravidla hry by bylo možné upravit tak, aby toto omezení nebylo potřeba. Bylo by to ale drobně technicky komplikovanější.

- Indukcí podle struktury formule ψ se snadno ukáže následující: Formule ψ má při ohodnocení ν pravdivostní hodnotu 1 právě tehdy, když má *Hráč* I v pozici (ψ, ν) vyhrávající strategii.

Kvantifikované booleovské formule (QBF) jako hra

Řekněme, že velikost vstupní formule φ je n:

- Je zjevné, že množství paměti, které postačuje pro uložení jedné pozice (ψ, ν), je $\mathcal{O}(n)$.
- Je rovněž zřejmé, že každá partie je konečná, a délka této partie je nejvýše n tahů.

Z toho vidíme, že dříve uvedený algoritmus má polynomiální prostorovou složitost ($\mathcal{O}(n^2)$), a daný problém tedy patří do třídy PSPACE.

PSPACE-obtížnost problému QBF jsme už ukázali dříve.

Dostáváme tedy následující:

Věta

Zjistit, zda ve hře na kvantifikované booleovské formuli má *Hráč I* vyhrávající strategii, je PSPACE-úplný problém.

Generalized Geography (GG)

Uvažujme následující hru, kterou hrají dva hráči na orientovaném grafu G:

- Hráči střídavě přesunují po vrcholech grafu G jeden hrací kámen.
- Při tazích se označují vrcholy, které již byly kamenem navštíveny.
- Začíná se na specifikovaném vrcholu v_0.
- Řekněme, že kámen je momentálně na vrcholu v. Hráč, který je na tahu, vybere vrchol v' takový, že existuje hrana z v do v' a vrchol v' nebyl dosud navštíven.
- Hráč, který nemůže táhnout, prohrál a jeho protihráč vyhrál.

Generalized Geografy (GG)

Vstup: Orientovaný graf G s vyznačeným počátečním vrcholem v_0.

Otázka: Má hráč, který táhne jako první, vyhrávající strategii ve hře hrané na grafu G, kde se začíná ve vrcholu v_0 ?

Generalized Geography (GG)

Pozice v této hře jsou dány:

- Informací, na kterém vrcholu se momentálně nachází kámen.
- Informací, které vrcholy již byly navštíveny.

Na uložení jedné pozice zjevně postačuje $\mathcal{O}(n)$ bitů, kde n je počet vrcholů grafu G.

Je také zjevné, že každá partie skončí nejvýše po n tazích, protože žádný vrchol nemůže být navštíven dvakrát.

Je tedy zjevné, že problém „*Generalized Geography*“ patří do třídy PSPACE.

Generalized Geography (GG)

PSPACE-obtížnost problému „*Generalized Geography*“ se dá ukázat redukcí z problému QBF.

Popíšeme si polynomiální algoritmus, který:

- Dostane jako vstup kvantifikovanou booleovskou formuli φ.
- Vygeneruje jako výstup orientovaný graf G s vyznačeným počátečním vrcholem v_0.
- Bude platit, že *Hráč I* (který v dané hře začíná), bude mít vyhrávající strategii právě tehdy, když je formule φ pravdivá.

Generalized Geography (GG)

Navíc budeme předpokládat, že formule φ je specifického tvaru:

- Je v prenexním tvaru, kde se kvantifikátory pravidelně střídají:

$$\exists x_1.\forall x_2.\exists x_3.\forall x_4.\ \cdots \exists x_{n-2}.\forall x_{n-1}.\exists x_n.\ \psi$$

- První a poslední kvantifikátor je existenční.
- Podformule ψ je v konjunktivní normální formě.

K dané formuli φ se vytvoří graf G takovým způsobem, že při hře „*Generalized Geography*“ na tomto grafu budou hráči de facto hrát hru na kvantifikované booleovské formuli φ.

Tam, kde by v původní QBF hře hráči vybírali přiřazení booleovských hodnot proměnným nebo vybírali jednu z podformulí dané formule, budou ve hře na grafu realizovat tyto volby tím, že vyberou z několika možností následující vrchol, kam se přesune v daném tahu kámen.

Generalized Geography (GG)

Hra na grafu G bude v tomto smyslu „simulovat“ hru na formuli φ:

- Nejprve budou hráči střídavě přiřazovat pravdivostní hodnoty proměnným v pořadí $x_1, x_2, \ldots, x_n$:
  - *Hráč I* bude mít na výběr ze dvou možností u proměnných s **existenčním** kvantifikátorem.
  - *Hráč II* bude mít na výběr ze dvou možností u proměnných s **univerzálním** kvantifikátorem.
- Poté *Hráč II* vybere jednu z **klauzulí** formule ψ.
- Následně *Hráč I* vybere jeden **literál** z této klauzule.
- V této chvíli bude na tahu *Hráč II*— bude nebo nebude moci táhnout podle toho, zda tento vybraný literál bude při daném ohodnocení pravdivý nebo nepravdivý.

Generalized Geography (GG)

- Pokud v první fázi hry byla dané proměnné v literálu přiřazena hodnota, při které je tento literál pravdivý, bude jediný možný tah *Hráče II* vést do vrcholu grafu, který byl v této chvíli již navštíven.

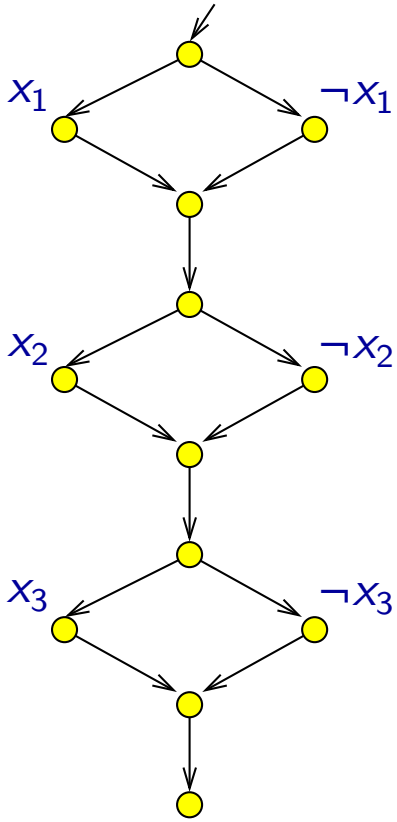
  Hráč II tedy prohrál a vyhrává *Hráč I*.

- Pokud bude daný literál nepravdivý, bude *Hráč II* moci táhnout. Dostane se do situace, kdy je na tahu *Hráč I*, přičemž ale jediná hrana z daného vrcholu povede do již dříve navštíveného vrcholu.

  Hráč I tedy v tomto případě prohrál a vyhrává *Hráč II*.

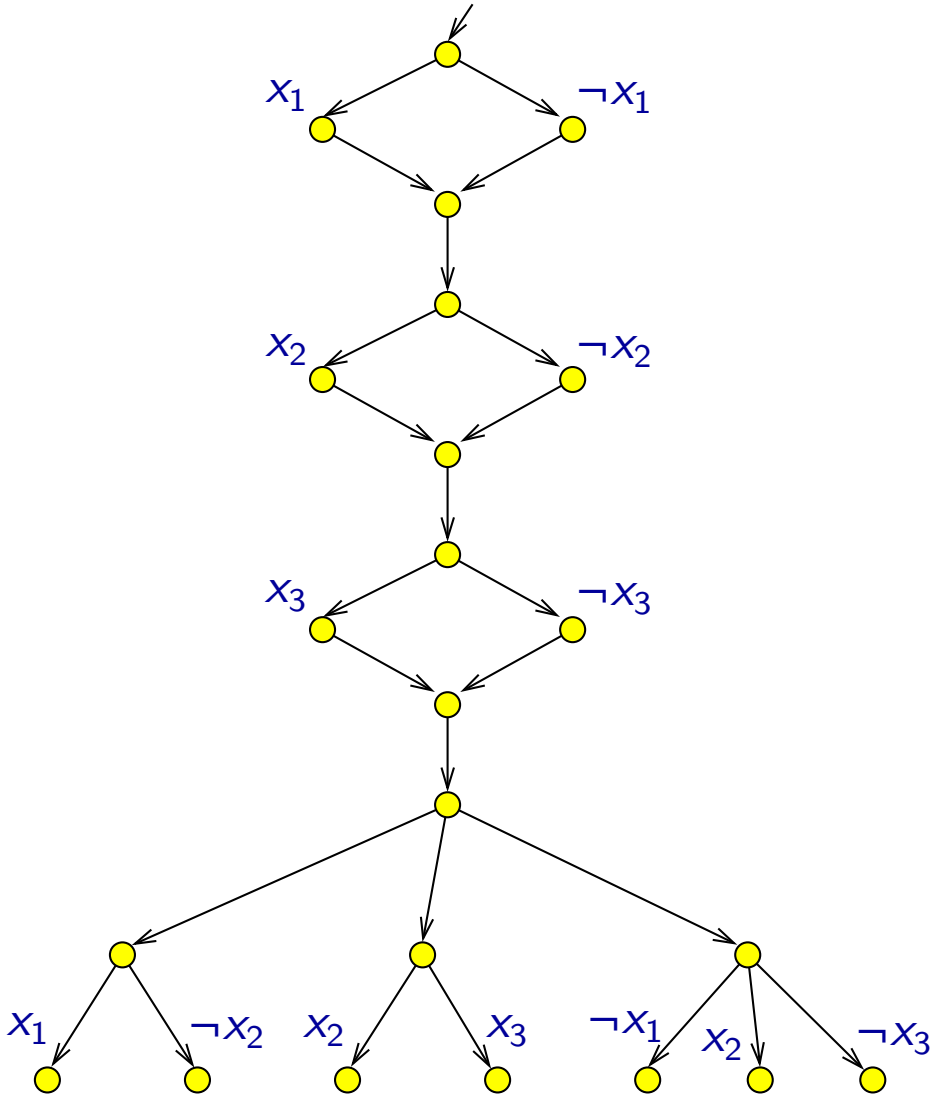
Příklad: Graf sestrojený pro formuli

$\exists x_1.\forall x_2.\exists x_3.((x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3))$



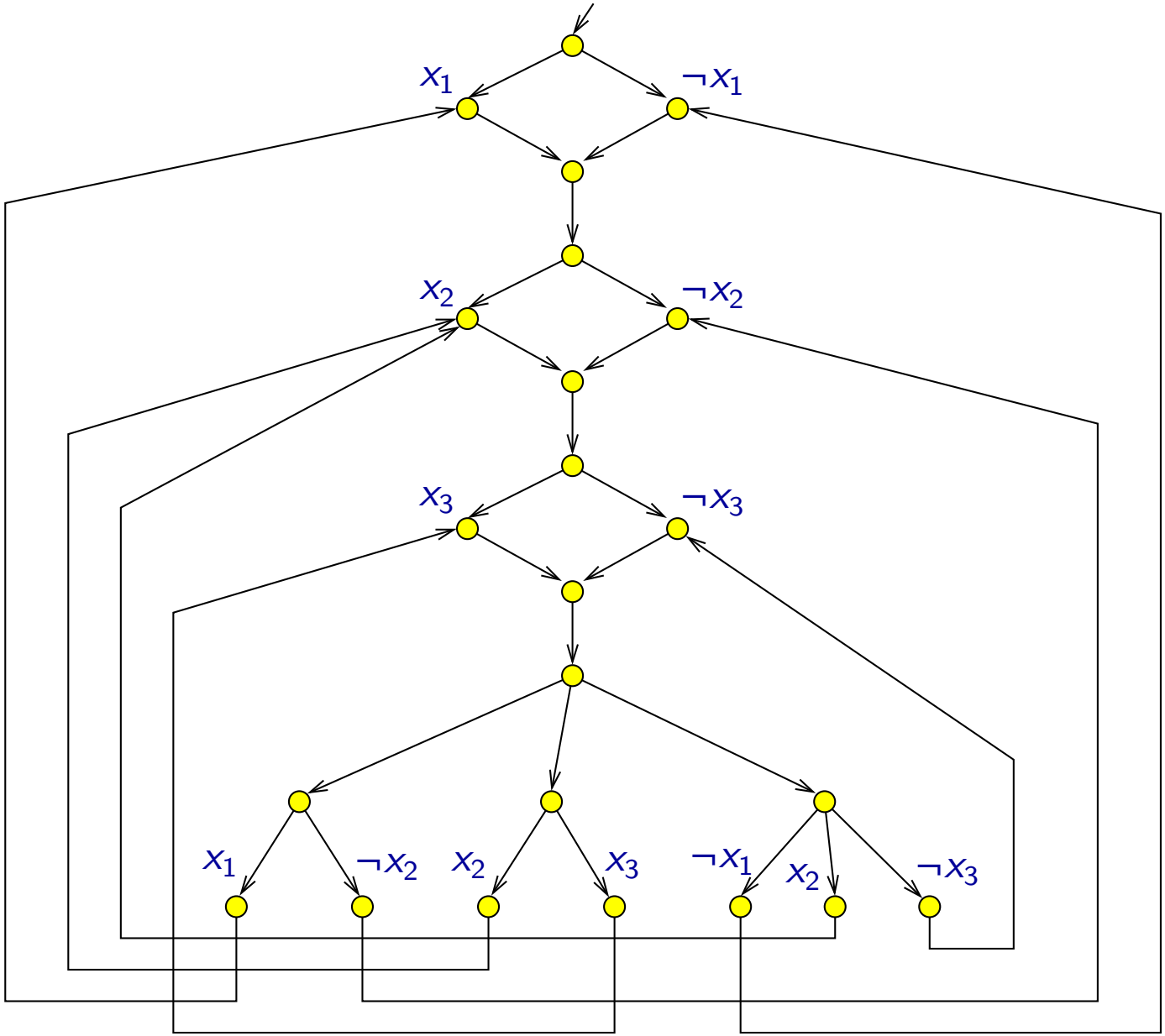
Příklad: Graf sestrojený pro formuli

$\exists x_1.\forall x_2.\exists x_3.((x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3))$



Příklad: Graf sestrojený pro formuli

$\exists x_1.\forall x_2.\exists x_3.((x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3))$



Generalized Geography (GG)

Není těžké ověřit, že ve hře na grafu G vytvořeném k formuli φ má *Hráč I* vyhrávající strategii právě tehdy, když má vyhrávající strategii ve hře na formuli φ, tj. právě tehdy, když je formule φ pravdivá.

Vidíme tedy, že platí následující:

Věta

Problém „*Generalized Geography*“ je PSPACE-úplný.

Generalized Geography (GG) — planární graf

Ve skutečnosti je problém „*Generalized Geography*“ PSPACE-úplný i ve speciálním případě, kdy je graf G **planární**.

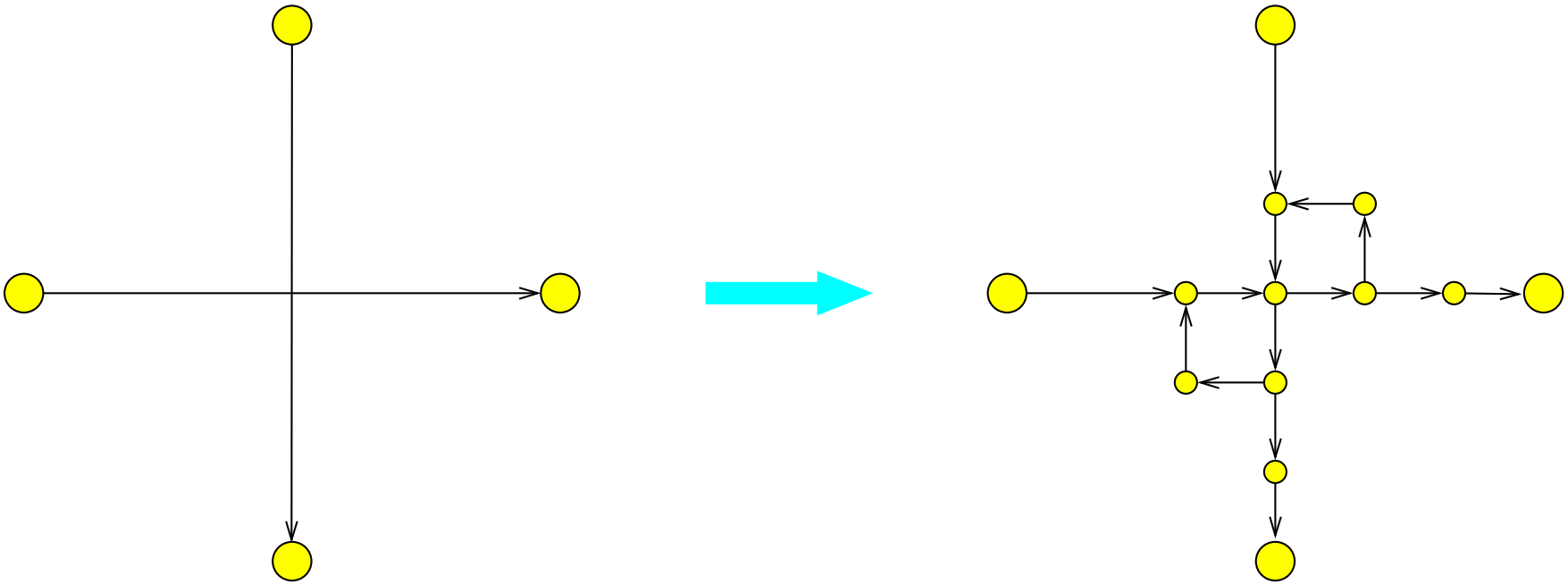
Graf je **planární**, jestliže je možné ho nakreslit v rovině takovým způsobem, že se žádné dvě hrany spolu nekříží.
(Hrany je možné kreslit jako libovolné křivky, ne nutně jako úsečky.)

Generalized Geography (GG) — planární graf

- V grafu sestrojeném výše uvedenou konstrukcí v důkaze PSPACE-obtížnosti problému GG se některé hrany kříží.
- Graf z této konstrukce se dá nakreslit tak, že jediné hrany, které se kříží, jsou hrany, které vedou z vrcholu odpovídajícího vybranému literálu x_i nebo $\neg x_i$ do odpovídajícího vrcholu v části grafu pro příslušný kvantifikátor $\exists x_i$ nebo $\forall x_i$.
- Je zřejmé, že v každé partii se projde právě jedna ze všech těchto hran.
- Pro každou dvojici křížících se hran tak platí, že se bude během každé partie procházet nejvýše jednou z těchto dvou hran.

Generalized Geography (GG) — planární graf

V místech, kde se nějaké dvě hrany kříží, je možné nahradit tato křížení pomocí následující konstrukce:

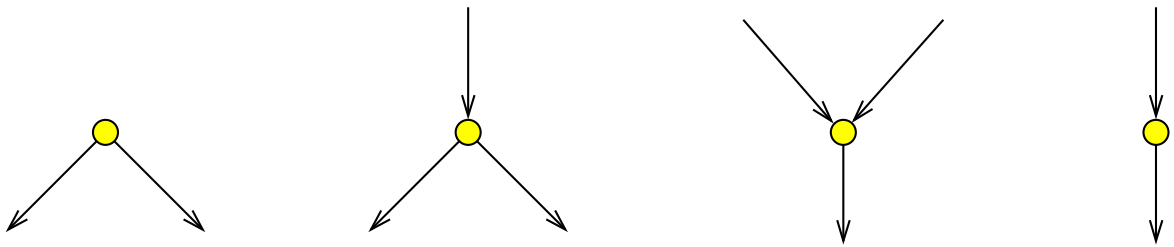


Generalized Geography (GG) — planární graf

Není těžké si také rozmyslet, že za cenu přidání dalších pomocných vrcholů a hran je možné graf upravit tak, aby pro každý vrchol platilo následující:

- do vrcholu vstupují nejvýše dvě hrany a vystupuje z něj z nejvýše jedna hrana, nebo
- do vrcholu vstupuje nejvýše jedna hrana a vystupují z něj z nejvýše dvě hrany

Graf pak bude obsahovat vrcholy jen několika málo typů:



Tuto úpravu je navíc možné provést tak, že pokud byl graf planární, tak bude planární i po této úpravě.

Generalized Geography (GG) — planární graf

Výsledný planární graf, kde jsou všechny vrcholy jen výše uvedených typů, je navíc možné nakreslit do roviny tak, aby všechny hrany vedly jen ve **vodorovném** a **svislém** směru — tj. rovnoběžně s osami souřadnic.

Příslušnou redukci je možné upravit tak, aby generovala nejen popis výsledného grafu, ale i informace o nakreslení grafu v rovině — tj. souřadnice jednotlivých vrcholů a hran.

Problém zůstává PSPACE-úplný i pro takto nakreslené grafy, což je možné využívat při důkazech PSPACE-obtížnosti dalších problémů pomocí redukce z této varianty problému.

Zobecněná varianta hry Hex

Uvažujme následující hru:

- Hrají dva hráči — *Hráč I* a *Hráč II* na neorientovaném grafu, ve kterém jsou dva vrcholy s a t vyznačeny jako speciální.
- Hráči střídavě pokládají své kameny na vrcholy grafu:
  - *Hráč I* má kameny jedné barvy — např. zelené
  - *Hráč II* má kameny druhé barvy — např. červené

  Oba hráči mají neomezenou zásobu kamenů své barvy.
- Na začátku hry je graf prázdný — neleží na něm žádné kameny.
- Jako první táhne *Hráč I*.
- Tah hráče spočívá v tom, že vybere libovolný vrchol, na kterém dosud neleží žádný kámen, a položí kámen své barvy na tento vrchol.

  Na speciální vrcholy s a t se kameny nesmí pokládat.

Zobecněná varianta hry Hex

- Cílem *Hráče* I je vytvořit ze svých kamenů **cestu** z vrcholu s do vrcholu t.
- Cílem *Hráče* II je mu v tom zabránit.

Partie končí jedním ze dvou způsobů:

- Na některé cestě z s do t leží samé zelené kameny:

  Vyhrál *Hráč* I.
- Na každé cestě z s do t leží alespoň jeden červený kámen:
  (*Hráč* I už tedy určitě nebude schopen vytvořit ze svých kamenů cestu z s do t.)

  Vyhrál *Hráč* II.

Zobecněná varianta hry Hex

Jedná se o určité zobecnění deskové hry **Hex**.

Hra Hex se standardně hraje na pravidelné hrací ploše tvořené šestiúhelníky:

- tyto šestiúhelníky si můžeme představit jako vrcholy grafu
- vrcholy odpovídající šestiúhelníkům jsou spojeny hranou, jestliže spolu dané šestiúhelníky sousedí
- vrcholy s a t si můžeme představit jako speciální vrcholy mimo hrací plochu, které jsou spojeny hranami se všemi šestiúhelníky na jedné a druhé straně hrací plochy

Zobecnění zde spočívá v tom, že místo standardní podoby se šestiúhelníky uvažujeme hru na libovolném grafu.
(Tento graf nemusí být ani planární.)

Zobecněná varianta hry Hex

Vezměme si následující problém:

Zobecněná varianta hry Hex

Vstup: Neorientovaný graf G se dvěma speciálními vyznačenými vrcholy s a t.

Otázka: Má *Hráč* I ve hře na grafu G vyhrávající strategii?

Řekněme, že graf G má n vrcholů:

- Pro uložení jedné pozice postačuje $\mathcal{O}(n)$ bitů.
- Každá partie má nejvýše n tahů.

Dříve popsaný rekurzivní algoritmus procházející systematicky strom všech možných partií bude mít proto prostorovou složitost $\mathcal{O}(n^2)$.

Výše uvedený problém patří tedy do třídy PSPACE.

Zobecněná varianta hry Hex

PSPACE-obtížnost tohoto problému je možné ukázat redukcí z problému QBF.

Tuto konstrukci zde nebudeme podrobně popisovat.
(Je náplní jednoho z referátů.)

Věta

„Zobecněná varianta hry Hex“ je PSPACE-úplný problém.

Kombinatorické hry — obecný případ

Na libovolnou kombinatorickou hru $G = (Pos, Moves, lab, \alpha_0)$ se můžeme dívat jako na **orientovaný graf**, kde:

- vrcholy jsou prvky množiny *Pos* (tj. **pozice**)
- hrany jsou prvky množiny *Moves* (tj. **tahy**)

Zatím jsme uvažovali pouze hry, kde přímo z pravidel vyplývalo, že každá partie musí být konečná a musí tím pádem končit vítězstvím jednoho z hráčů.

Mimo jiné tak v takových hrách není možné, aby v grafu hry existoval cyklus, tj. aby bylo možné, aby se nějaká pozice během partie zopakovala.

Zaměřme se nyní na obecný případ, kde graf hry může být zcela libovolný. Nebudeme však řešit případy, kdy je množina *Pos* nekonečná.
Tj. budeme předpokládat, že graf hry je konečný, může ale obsahovat cykly, a partie tak mohou být libovolně dlouhé a případně i nekonečné.

Kombinatorické hry — obecný případ

Předpokládejme tedy, že máme dánu kombinatorickou hru $G = (Pos, Moves, lab, \alpha_0)$, kde množina Pos je konečná.

Pro tuto hru G můžeme spočítat posloupnost

$$\mathcal{W}_0, \mathcal{W}_1, \mathcal{W}_2, \ldots$$

tvořenou podmnožinami množiny Pos, kde pro $i \in \mathbb{N}$ podmnožina $\mathcal{W}_i$ obsahuje právě ty pozice α z množiny Pos, ze kterých je schopen *Hráč* I vynutit, že během nejvýše i tahů bude dosažena nějaká pozice z množiny Win_{I}, tj. ty pozice, ve kterých dokáže *Hráč* I vyhrát do i tahů.

Kombinatorické hry — obecný případ

Množiny $\mathcal{W}_i$ můžeme počítat následovně:

- $\mathcal{W}_0 = \mathit{Win}_{\mathrm{I}}$
- Množinu $\mathcal{W}_{i+1}$ je možné z množiny $\mathcal{W}_i$ spočítat takto:
  - $\mathcal{W}_{i+1}$ bude obsahovat všechny pozice z $\mathcal{W}_i$.
  - Do $\mathcal{W}_{i+1}$ přidáme také všechny pozice $\alpha \in \mathit{Pos}_{\mathrm{I}}$, pro které **existuje** nějaké $\beta \in \mathcal{W}_i$ takové, že $\alpha \longrightarrow \beta$.
  - Dále přidáme do $\mathcal{W}_{i+1}$ všechny pozice $\alpha \in \mathit{Pos}_{\mathrm{II}}$, kde pro **každé** $\beta \in \mathit{succ}(\alpha)$ platí $\beta \in \mathcal{W}_i$.

Kombinatorické hry — obecný případ

Je zjevné, že množiny $\mathcal{W}_0, \mathcal{W}_1, \mathcal{W}_2, \ldots$ tvoří rostoucí posloupnost

$$\mathcal{W}_0 \subseteq \mathcal{W}_1 \subseteq \mathcal{W}_2 \subseteq \cdots$$

Jestliže je množina *Pos* konečná, musí nutně existovat nějaké $i \in \mathbb{N}$, pro které platí $\mathcal{W}_i = \mathcal{W}_{i+1}$.

Nutně pak platí $\mathcal{W}_j = \mathcal{W}_{j+1}$ i pro každé j, kde $j \geq i$, tj.

$$\mathcal{W}_i = \mathcal{W}_{i+1} = \mathcal{W}_{i+2} = \mathcal{W}_{i+3} = \cdots$$

Označme $\mathcal{W}$ tuto výslednou množinu $\mathcal{W}_i$, kde $\mathcal{W}_i = \mathcal{W}_{i+1}$.

Zápisem $\overline{\mathcal{W}}$ označme ty pozice, které nepatří do $\mathcal{W}$, tj. $\overline{\mathcal{W}} = \mathit{Pos} - \mathcal{W}$.

Kombinatorické hry — obecný případ

Je zjevné, že:

- Z každé pozice $\alpha \in \mathcal{W}$ je *Hráč I* schopen vynutit dosažení množiny Win_{I}, bez ohledu na to, jak hraje *Hráč II*.
- Na druhou stranu, očividně pro každou pozici $\alpha \in \overline{\mathcal{W}}$ platí:
  - Pokud $\alpha \in Pos_{\mathrm{I}}$, tak pro každé $\beta \in succ(\alpha)$ platí $\beta \in \overline{\mathcal{W}}$.
  - Pokud $\alpha \in Pos_{\mathrm{II}}$, tak existuje $\beta \in succ(\alpha)$ takové, že $\beta \in \overline{\mathcal{W}}$.

  Hráč II má tedy strategii, kterou může libovolně dlouho udržovat hru v pozicích z množiny $\overline{\mathcal{W}}$.

  Pokud bude během partie dosažena jakákoli pozice z $\overline{\mathcal{W}}$, tak *Hráč II* je schopen vynutit, že v následujících tazích už nikdy nebude dosažena žádná pozice z $\mathcal{W}$, a speciálně tedy, že nebude dosažena žádná pozice z Win_{I}, neboť $Win_{\mathrm{I}} \subseteq \mathcal{W}$.

Kombinatorické hry — obecný případ

- Vidíme tedy, že výsledná množina $\mathcal{W}$ obsahuje právě ty pozice, ve kterých má *Hráč I* **vyhrávající strategii**.
- Naopak množina $\overline{\mathcal{W}} = Pos - \mathcal{W}$ obsahuje právě ty pozice, ve kterých má *Hráč II* **neprohrávající strategii** — tj. strategii, která mu sice nemusí nutně zajistit vítězství, ale která mu zaručí, že určitě neprohraje (a tedy, že *Hráč I* nevyhraje).

Výše popsaný postup nám dává i algoritmus, který množinu $\mathcal{W}$ spočítá.

Není těžké si promyslet, že časová složitost přímočaré implementace takového algoritmu je polynomiální vzhledem k velikosti grafu hry G.

Kombinatorické hry — obecný případ

O něco **efektivnější implementace** je možné dosáhnout použitím algoritmu, který pracuje dosti podobně jako běžné prohledávání do šířky (breadth-first search), s tím rozdílem, že postupuje **proti** směru hran a u jednotlivých pozic rozlišuje různé případy podle toho, kdo je v dané pozici na tahu.

Kombinatorické hry — obecný případ

Inicializace algoritmu:

- pro každou pozici β se předpočítá množina všech jejích **předchůdců**, tj. množina

$$pred(\beta) = \{\, \alpha \in Pos \mid \alpha \longrightarrow \beta \,\}$$

- všem prvkům $\alpha \in Pos_{\mathrm{II}}$ je nastaven čítač $count(\alpha)$ na hodnotu $|succ(\alpha)|$
- všechny pozice s výjimkou pozic z množiny Win_{I} jsou označeny jako nezpracované
- pozice z množiny Win_{I} jsou označeny jako zpracované
- $\mathcal{W} := Win_{\mathrm{I}}$
- prvky z množiny Win_{I} jsou vloženy do **fronty**

Kombinatorické hry — obecný případ

Dokud je fronta neprázdná, je v cyklu prováděno následující:

- Z fronty je vybrána pozice β.
- Pro každou dosud nezpracovanou pozici $\alpha \in \mathit{pred}(\beta)$ se provede:
  - Pokud $\alpha \in \mathit{Pos}_{\mathrm{I}}$:

    $\mathcal{W} := \mathcal{W} \cup \{\alpha\}$

    Pozice α se označí jako zpracovaná a vloží se do fronty.
  - Pokud $\alpha \in \mathit{Pos}_{\mathrm{II}}$:

    $\mathit{count}(\alpha) := \mathit{count}(\alpha) - 1$

    Pokud $\mathit{count}(\alpha) = 0$:

    $\mathcal{W} := \mathcal{W} \cup \{\alpha\}$

    Pozice α se označí jako zpracovaná a vloží se do fronty.

Kombinatorické hry — obecný případ

Není těžké si promyslet, že složitost tohoto algoritmu bude zhruba lineární vzhledem k velikosti grafu hry G, tj. $\mathcal{O}(n+m)$, kde n je velikost množiny *Pos* a m velikost množiny *Moves*.

Předpokládejme dále, že tento algoritmus pracovat s hrou danou **implicitně**, kde množství paměti potřebné pro uložení jedné pozice bude $\mathcal{O}(f(n))$ bitů, pro nějakou funkci $f(n)$:

- Počet pozic (tj. hodnota n) bude $2^{\mathcal{O}(f(n))}$.
- Počet tahů (tj. hodnota m) bude $2^{\mathcal{O}(f(n))} \cdot 2^{\mathcal{O}(f(n))}$, tj. $2^{\mathcal{O}(f(n))+\mathcal{O}(f(n))}$, což je stále $2^{\mathcal{O}(f(n))}$.

Celková časová složitost algoritmu bude v tom případě $2^{\mathcal{O}(f(n))}$.

Prostorová složitost algoritmu bude rovněž $2^{\mathcal{O}(f(n))}$.

Kombinatorické hry — obecný případ

Uvažujeme verze různých známých deskových her jako jsou například:

- šachy
- dáma
- GO

zobecněné tak, že se nehrají na hrací ploše nějaké pevně dané velikosti (8×8 u šachu či dámy, 19×19 u GO, apod.), ale na ploše velikosti $n \times n$ pro libovolné $n \in \mathbb{N}$.

Uvažujme problémy následujícího typu:

Vstup: Pozice α v nějaké takové zobecněné hře na ploše $n \times n$, kde jsou některé figurky, kameny, apod. již rozmístěny, a informace, který z hráčů je momentálně na tahu.

Otázka: Má *Hráč* I v pozici α vyhrávající strategii?

Kombinatorické hry — obecný případ

Z předchozího plyne, že ve všech případech, kdy je množství paměti potřebné pro uložení jedné pozice polynomiální, bude výše uvedený problém patřit do třídy EXPTIME.

- Pro šachy i dámu je známa EXPTIME-obtížnost dané úlohy.
  Pro obě tyto hry je tedy daný problém EXPTIME-úplný.
- Pro GO je známa PSPACE-obtížnost.

  Dá se to dokázat pomocí převodu z Generalized Geography (v planární verzi).

  (Důkaz této PSPACE-obtížnosti je obsahem jednoho z referátů.)