Konfigurace a výpočty jako data
Univerzální Turingův stroj

Konfigurace jako data

Připomeňme si, jak vypadají **konfigurace** různých typů strojů:

- **jednopáskový Turingův stroj**:
  — stav řídící jednotky, obsah pásky, pozice hlavy
- **vícepáskový Turingův stroj**:
  — stav řídící jednotky, obsahy všech pásek, pozice všech hlav
- **stroj RAM**:
  — adresa prováděné instrukce, obsah pracovní paměti, obsah vstupní a výstupní pásky
- **graf řídícího toku**:
  — řídící stav (vrchol v grafu řídícího toku), obsah paměti (hodnoty jednotlivých proměnných)
- **Minského stroj**:
  — stav řídící jednotky, hodnoty všech čítačů

Konfigurace jako data

Ve všech těchto případech (a obecně i u jiných dalších výpočetních modelů) jsou konfigurace daného stroje **konečné** objekty, se kterými je možno pracovat jako s **daty**:

- ve vyšších programovacích jazycích mohou být například reprezentovány nějakou vhodně zvolenou **datovou strukturou**.
- mohou být také reprezentovány jako **slova** v nějaké abecedě — je třeba zvolit nějaký vhodný formát zápisu konfigurací

Tyto datové objekty (datové struktury, slova, apod.) reprezentující konfigurace můžeme ztotožnit s těmito konfiguracemi:

- Když například řekneme, že něco uděláme s konfigurací α, máme tím na mysli, že pracujeme s reprezentací konfigurace α ve formě dat.

Konfigurace jako data

Vezměme si nějaký konkrétní stroj $\mathcal{M}$:

- Označme *Conf* množinu všech konfigurací daného stroje $\mathcal{M}$.
- Definujme relaci

$$\longrightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$$

jako množinu těch dvojic konfigurací α a α', pro které platí, že stroj $\mathcal{M}$ může **jedním krokem** přejít z konfigurace α do konfigurace α', což budeme zapisovat

$$\alpha \longrightarrow \alpha'$$

- Některé konfigurace z množiny *Conf* jsou označeny jako **koncové** — výpočet v nich (úspěšně) končí.

Pro každou koncovou konfiguraci α platí, že neexistuje žádná konfigurace α' taková, že $\alpha \longrightarrow \alpha'$

Konfigurace jako data

Výpočet daného stroje $\mathcal{M}$ pro vstup $w \in In$ (kde In je množina možných vstupů) je konečná nebo nekonečná posloupnost konfigurací z množiny $Conf$, tj.

- **konečný výpočet**: $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_t$
- **nekonečný výpočet**: $\alpha_0, \alpha_1, \alpha_2, \ldots$

kde:

- α_0 je **počáteční konfigurace** pro vstup w
- Pro každé $i \in \mathbb{N}$ (v případě konečného výpočtu pro každé $i \in \mathbb{N}$ takové, že $i < t$) platí
$$\alpha_i \longrightarrow \alpha_{i+1}$$
- V případě konečného výpočtu je α_t koncová konfigurace.

Poznámka: V případě, že stroj $\mathcal{M}$ je **deterministický**, existuje ke každé konfiguraci α nejvýše jedna konfigurace α' taková, že $\alpha \longrightarrow \alpha'$.

Konfigurace jako data

S datovými objekty reprezentujími konfigurace stroje $\mathcal{M}$ můžeme provádět různé operace, např.:

- pro daný vstup w vyrobit odpovídající **počáteční konfiguraci** α_0 stroje $\mathcal{M}$
- otestovat, zda daná konfigurace α je **koncovou konfigurací** stroje $\mathcal{M}$ — pokud ano, tak z dané konfigurace α určit, jaký je výstup stroje $\mathcal{M}$, když se zastavil v této konfiguraci
- pro danou konfiguraci α, která není koncová, spočítat konfiguraci α' takovou, že stroj $\mathcal{M}$ přejde **jedním krokem** z konfigurace α do konfigurace α'
- zjistit pro danou dvojici konfigurací α a α', zda stroj $\mathcal{M}$ může jedním krokem přejít z konfigurace α do konfigurace α'

Takovéto operace mohou být snadno implementovány jako algoritmy.

Konfigurace jako data

S použitím těchto operací je možné snadno implementovat například **simulaci** výpočtu stroje $\mathcal{M}$ nad vstupem w:

Algoritmus: Simulace výpočtu stroje $\mathcal{M}$ nad vstupem w

```
RUN(w):
    α := INIT-CONF(w)
    while not IS-FINAL(α) do
        α := NEXT-CONF(α)
    return EXTRACT-OUTPUT(α)
```

kde:

- INIT-CONF(w) — ke vstupu w spočítá počáteční konfiguraci stroje $\mathcal{M}$
- IS-FINAL(α) — testuje, zda je α koncová konfigurace
- NEXT-CONF(α) — spočítá konfiguraci α' takovou, že $\alpha \longrightarrow \alpha'$
- EXTRACT-OUTPUT(α) — z koncové konfigurace α zjistí výstup stroje $\mathcal{M}$

Výpočty jako data

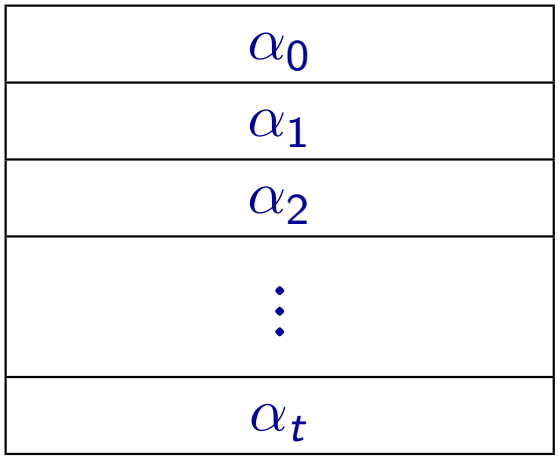
V případě **konečného** výpočtu

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \quad \cdots \quad \longrightarrow \alpha_t$$

je možné pracovat s tímto výpočtem jako s daty.

Může být reprezentován například jako:

- nějaká vhodná **datová struktura** reprezentující posloupnost konfigurací



Výpočty jako data

V případě **konečného** výpočtu

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \quad \cdots \quad \longrightarrow \alpha_t$$

je možné pracovat s tímto výpočtem jako s daty.

Může být reprezentován například jako:

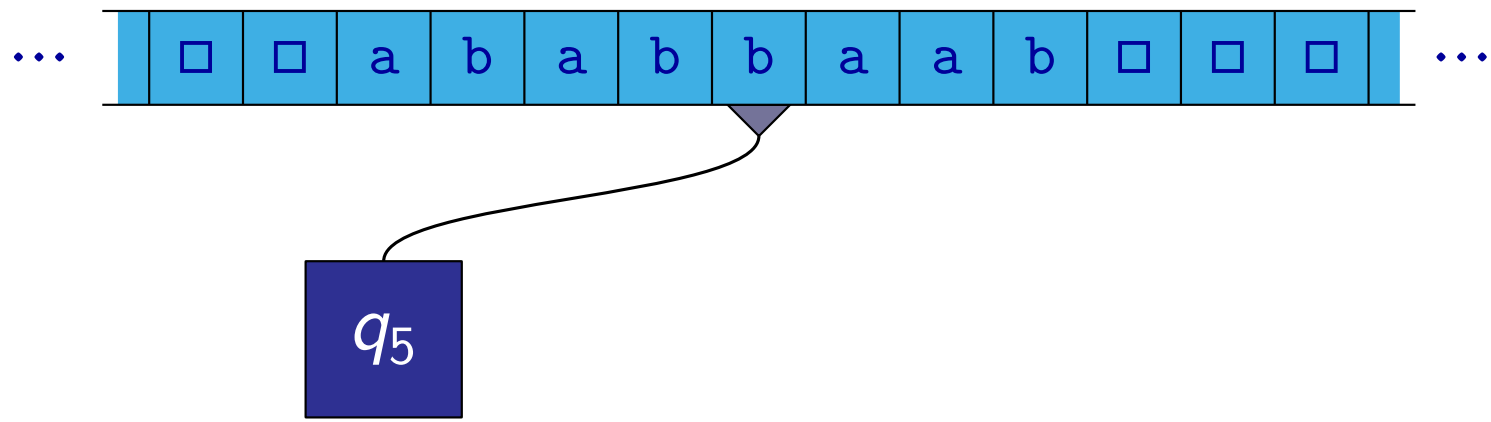
- **slovo**

  $$\#\alpha_0\#\alpha_1\#\alpha_2\# \quad \cdots \quad \#\alpha_t\#$$

  kde:

  - $\alpha_0, \alpha_1, \ldots, \alpha_t$ — slova reprezentující jednotlivé konfigurace výpočtu
  - $\#$ – symbol zvolený jako oddělovač konfigurací (nevyskytuje se v zápisu jednotlivých konfigurací)

Konfigurace Turingova stroje jako data

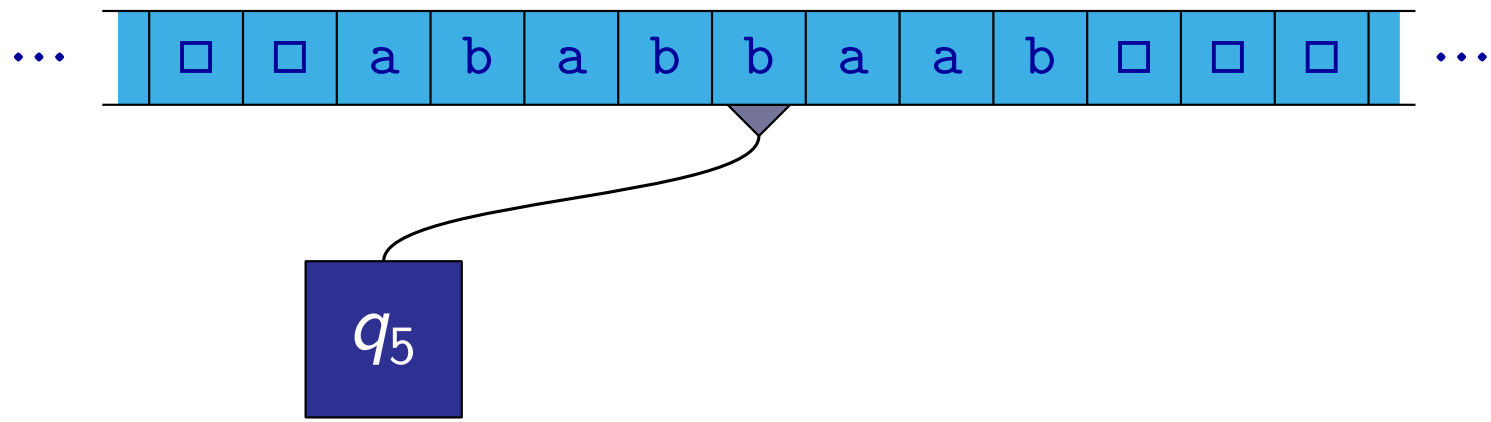


Uvažujme například jednopáskový Turingův stroj $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

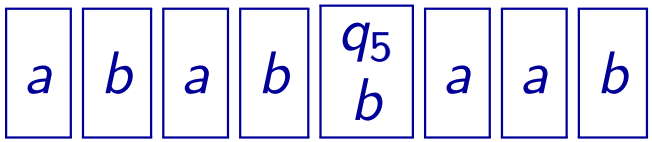
Konfigurace takového stroje musí obsahovat informaci o:

- stavu řídící jednotky
- obsahu pásky
- pozici hlavy

Konfigurace Turingova stroje jako data

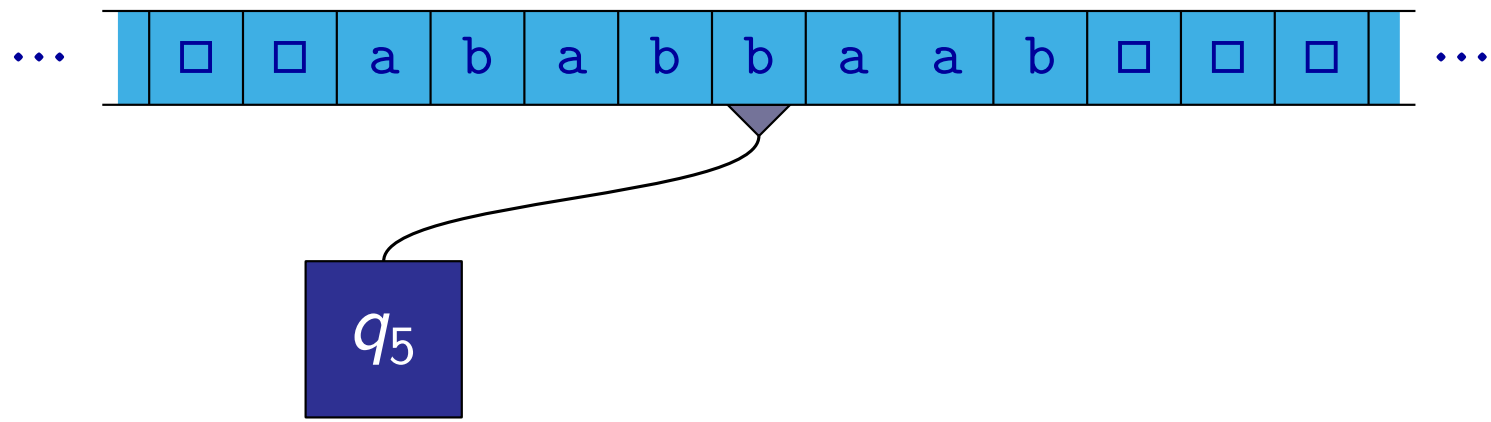


Konfigurace jednopáskového Turingova stroje $\mathcal{M}$ můžeme reprezentovat například jako slova v abecedě $\Delta = \Gamma \cup (Q \times \Gamma)$:

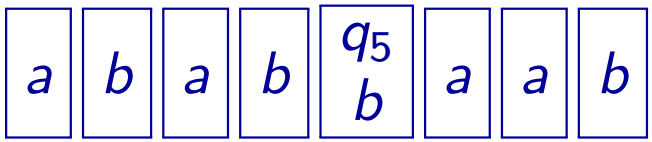


Toto slovo vždy obsahuje právě jeden znak z $(Q \times \Gamma)$, který vyznačuje stav řídící jednotky i pozici hlavy.

Konfigurace Turingova stroje jako data

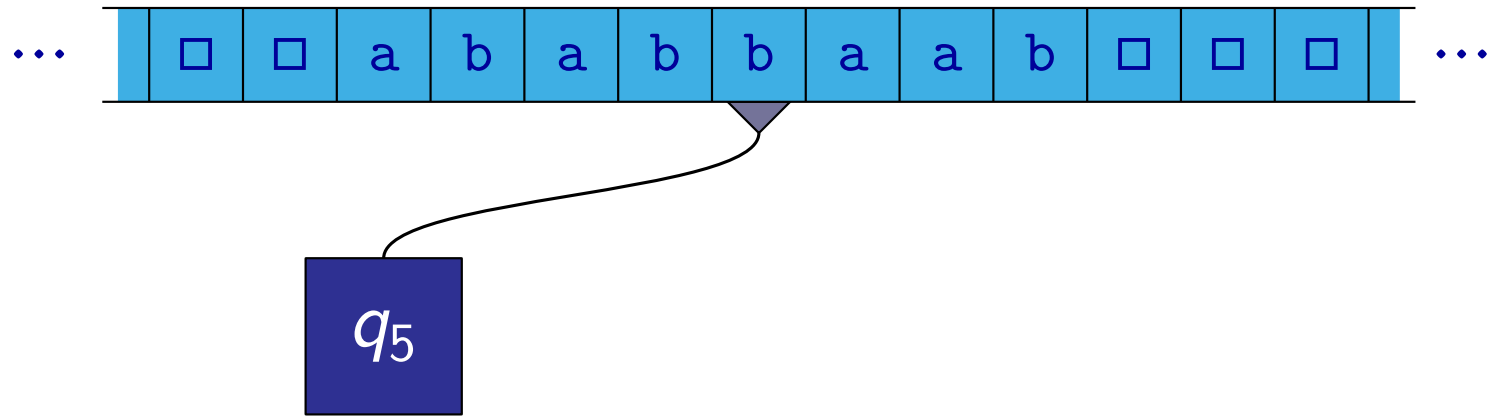


Konfigurace jednopáskového Turingova stroje $\mathcal{M}$ můžeme reprezentovat například jako slova v abecedě $\Delta = \Gamma \cup (Q \times \Gamma)$:

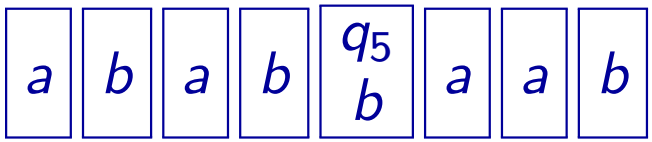


Poznámka: Znaky z $(Q \times \Gamma)$ můžeme též psát jako $\boxed{\begin{matrix} q \\ a \end{matrix}}$ místo $\boxed{(q, a)}$.

Konfigurace Turingova stroje jako data

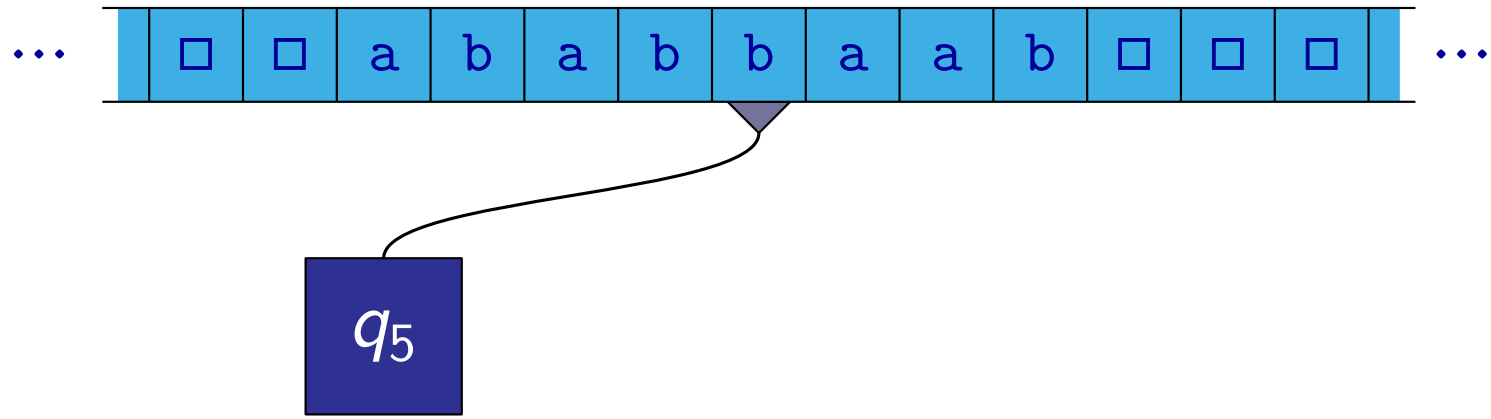


Konfigurace jednopáskového Turingova stroje $\mathcal{M}$ můžeme reprezentovat například jako slova v abecedě $\Delta = \Gamma \cup (Q \times \Gamma)$:

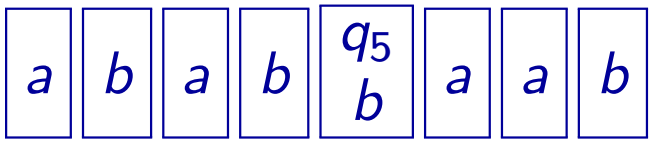


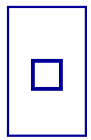
Ostatní symboly (z Γ) reprezentují obsah pásky.

Konfigurace Turingova stroje jako data



Konfigurace jednopáskového Turingova stroje $\mathcal{M}$ můžeme reprezentovat například jako slova v abecedě $\Delta = \Gamma \cup (Q \times \Gamma)$:



Políčka pásky, která nejsou ve slově vyznačena, obsahují symbol  .

Konfigurace Turingova stroje jako data

Označme *Conf* množinu všech konfigurací daného stroje $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

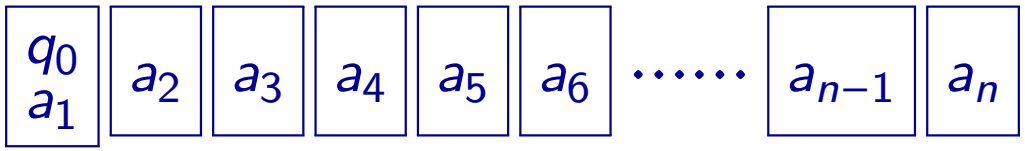
- Pokud budeme reprezentovat konfigurace Turingova stroje $\mathcal{M}$ výše popsaným způsobem jako slova nad abecedou $\Delta = \Gamma \cup (Q \times \Gamma)$, můžeme množinu *Conf* ztotožnit s množinu těch slov nad abecedou Δ, která obsahují právě jeden výskyt symbolu z množiny $(Q \times \Gamma)$, tj. s množinou slov tvaru

$$u(q, a)v$$

kde $q \in Q$, $a \in \Gamma$, $u, v \in \Gamma^*$.

Konfigurace Turingova stroje jako data

Počáteční konfigurace pro vstup $w \in \Sigma^*$, kde $w = a_1 a_2 \cdots a_n$, je tak reprezentována slovem



či při použití běžného matematického způsobu zápisu:

$$(q_0, a_1)\, a_2\, a_3 \cdots a_{n-1}\, a_n$$

Poznámka: Pokud je $w = \varepsilon$, je počáteční konfigurace reprezentována slovem $(q_0, \square)$.

Konfigurace Turingova stroje jako data

Ke každé konfiguraci α tvaru

$$u(q,a)v$$

kde $q \in (Q - F)$, $a \in \Gamma$, $u, v \in \Gamma^*$, existuje právě jedna konfigurace α' taková, že $\alpha \longrightarrow \alpha'$.

Tato konfigurace α' je určena přechodovou funkcí δ.

Předpokládejme, že $\delta(q,a) = (q', a', d)$:

- Pokud $d = 0$, pak $\alpha' = u(q', a')v$.
- Pokud $d = -1$:
  - Pokud $u = \varepsilon$, pak $\alpha' = (q', \square)a'v$.
  - Pokud $u = u'b$ (kde $u' \in \Gamma^*$ a $b \in \Gamma$), pak $\alpha' = u'(q', b)a'v$.
- Pokud $d = +1$:
  - Pokud $v = \varepsilon$, pak $\alpha' = ua'(q', \square)$.
  - Pokud $v = bv'$ (kde $b \in \Gamma$ a $v' \in \Gamma^*$), pak $\alpha' = ua'(q', b)v'$.

Konfigurace Turingova stroje jako data

Koncové konfigurace jsou konfigurace tvaru

$$u(q, a)v$$

kde $q \in F$, $a \in \Gamma$, $u, v \in \Gamma^*$.

Výpočet Turingova stroje jako data

Konečný výpočet Turingova stroje

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \alpha_3 \longrightarrow \quad \cdots \quad \longrightarrow \alpha_{t-1} \longrightarrow \alpha_t$$

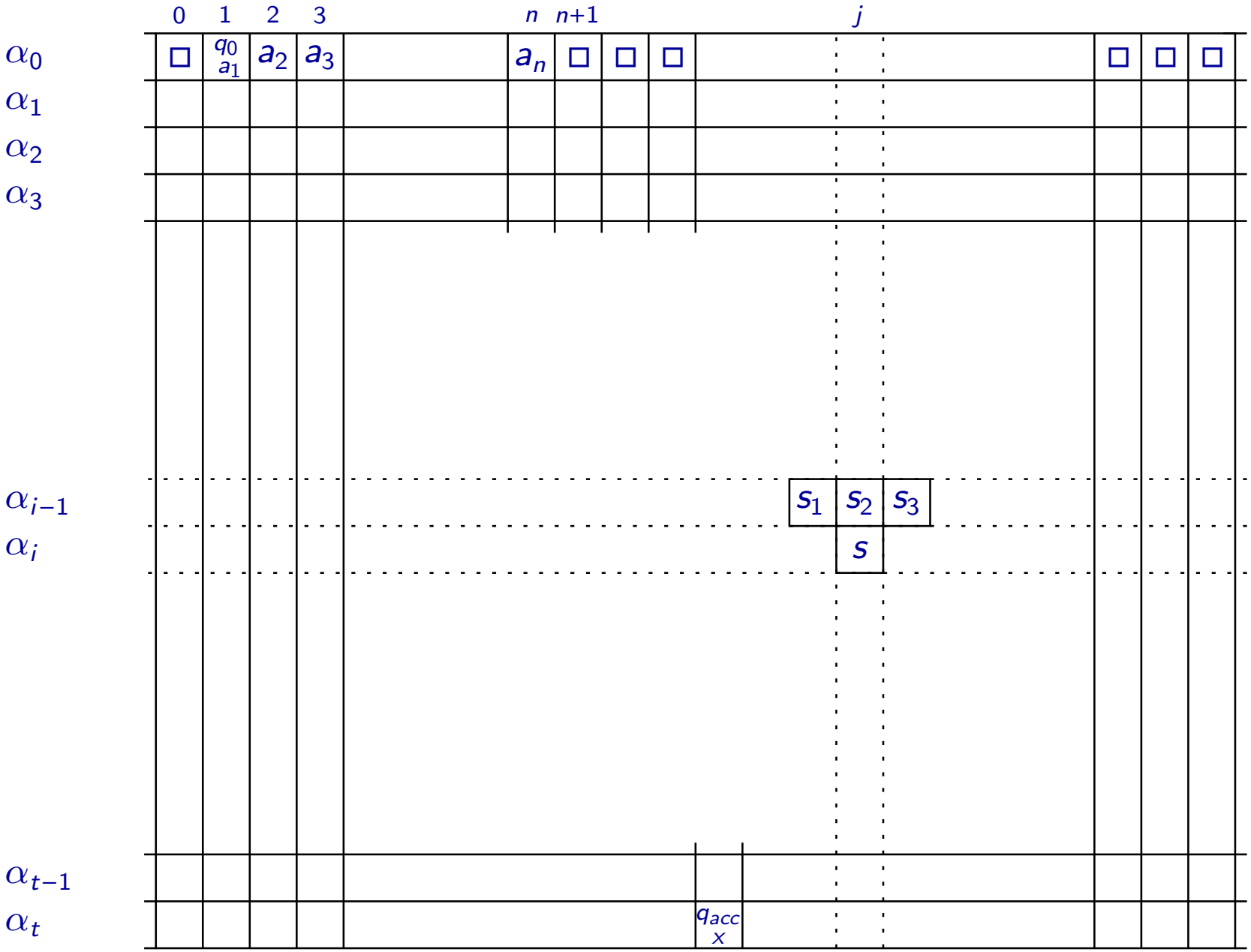
pak může být reprezentován například jako:

- posloupnost konfigurací oddělených nějakým speciálním symbolem $\# \notin \Delta$.

  — tj. jako jedno dlouhé **slovo** nad abecedou $\Delta \cup \{\#\}$

- **tabulka**, jejíž políčka obsahují symboly z abecedy Δ, kde řádky odpovídají jednotlivým konfiguracím a sloupce pozicím na pásce.

  Všimněte si, že v této reprezentaci je obsah políčka na řádku i (kde $i > 0$) a ve sloupci j plně určen popisem stroje $\mathcal{M}$ a obsahem políček ve sloupcích $j-1$, j a $j+1$ na řádku $i-1$.


0
1
2
3
n
n+1
j
α_0
α_1
α_2
α_3
α_{i-1}
α_i
α_{t-1}
α_t
q_0
a_1
a_2
a_3
a_n
s_1
s_2
s_3
s
q_{acc}
x

Výpočet Turingova stroje jako data

Pokud máme dánu nějakou takovou reprezentaci konečného výpočtu stroje $\mathcal{M}$ nad vstupem w, například ve formě slova tvořeného posloupností konfigurací, tabulkou, apod., můžeme na této reprezentaci testovat například, jestli:

- α_0 je opravdu počáteční konfigurace stroje $\mathcal{M}$ pro vstup w.
- pro každé $i < t$ platí $\alpha_i \longrightarrow \alpha_{i+1}$.
- α_t je koncová konfigurace a jaký je případně výstup stroje $\mathcal{M}$ v této konfiguraci.

To, že například stroj $\mathcal{M}$ dává pro daný vstup w odpověď ANO, pak platí právě tehdy, když existuje reprezentace výpočtu stroje $\mathcal{M}$ nad slovem w splňující výše uvedené podmínky a navíc ještě to, že v koncové konfiguraci α_t vrací stroj $\mathcal{M}$ jako výstup hodnotu ANO.

Kód Turingova stroje jako data

Popis libovolného Turingova stroje $\mathcal{M}$ (či jiného výpočetního modelu) můžeme rovněž reprezentovat ve formě dat — např. ve formě slova nad nějakou abecedou.

Zápisem $Code(\mathcal{M})$ označme takovouto reprezentaci stroje $\mathcal{M}$ (v nějakém konkrétním formátu).

- Zápis $Code(\mathcal{M})$ bude obsahovat informace o stavech řídící jednotky, o přechodové funkci, atd.
- Na $Code(\mathcal{M})$ můžeme nahlížet jako na kód programu.

Univerzální Turingův stroj

Univerzální Turingův stroj $\mathcal{U}$ je stroj, který když dostane jako vstup $Code(\mathcal{M})$ a slovo $w \in \Sigma^*$ (kde Σ je vstupní abeceda stroje $\mathcal{M}$), začne simulovat činnost stroje $\mathcal{M}$ nad vstupem w.

(Vstup $Code(\mathcal{M})$ a w může dostat stroj $\mathcal{U}$ například ve formě slova $Code(\mathcal{M})\#w$.)

Univerzální Turingův stroj je tedy schopen vykonávat činnost libovolného jiného Turingova stroje (jehož popis dostane jako součást vstupu).

Poznámka: Odpovídá to situaci, kdy máme:

- hardware počítače (stroj $\mathcal{U}$), který je schopen vykonávat libovolný algoritmus
- kód programu ($Code(\mathcal{M})$) spouštěného na tomto počítači
- vstupní data pro tento program (slovo w)

Univerzální Turingovy stroje

Poznámka: Dají se sestrojit překvapivě malé univerzální Turingovy stroje.

Například jsou známy univerzální Turingovy stroje, které simulují činnost zadaného jednopáskového Turingova stroje, které mají:

- 3 stavy a 11 symbolů
- 5 stavů a 7 symbolů
- 6 stavů a 6 symbolů
- 7 stavů a 5 symbolů
- 8 stavů a 4 symboly

(Počtem stavů se zde myslí počet stavů řídící jednotky Q, přičemž se nepočítají koncové stavy. Počtem symbolů se myslí velikost páskové abecedy Γ daného univerzálního stroje.)

Dají se sestrojit dokonce ještě menší univerzální Turingovovy stroje, které ovšem nesimulují Turingůvy stroje, nýbrž některé jiné druhy (extrémně omezených) Turingovsky úplných modelů.

Nerozhodnutelné problémy

Algoritmicky řešitelné problémy

Předpokládejme, že máme dán nějaký problém P.

Jestliže existuje nějaký algoritmus, který řeší problém P, pak říkáme, že problém P je **algoritmicky řešitelný**.

Jestliže P je rozhodovací problém a jestliže existuje nějaký algoritmus, který problém P řeší, pak říkáme, že problém P je **(algoritmicky) rozhodnutelný**.

Když chceme ukázat, že problém P je algoritmicky řešitelný, stačí ukázat nějaký algoritmus, který ho řeší (a případně ukázat, že daný algoritmus problém P skutečně řeší).

Algoritmicky neřešitelné problémy

Problém, který není algoritmicky řešitelný, je **algoritmicky neřešitelný**.

Rozhodovací problém, který není rozhodnutelný, je **nerozhodnutelný**.

Kupodivu existuje řada algoritmických problémů (přesně definovaných), o kterých je dokázáno, že nejsou algoritmicky řešitelné.

Halting Problem

Vezměme si nějaký libovolný obecný programovací jazyk $\mathcal{L}$.

Navíc předpokládejme, že programy v jazyce $\mathcal{L}$ běží na nějakém idealizovaném stroji, kde mají k dispozici (potenciálně) neomezené množství paměti — tj. kde alokace paměti nikdy neselže kvůli nedostatku paměti.

Příklad: Následující problém zvaný **Problém zastavení (Halting problem)** je nerozhodnutelný:

Halting problem

Vstup: Zdrojový kód programu P v jazyce $\mathcal{L}$, vstupní data x.

Otázka: Zastaví se program P po nějakém konečném počtu kroků, pokud dostane jako vstup data x?

Halting Problem

Předpokládejme, že by existoval nějaký program, který by rozhodoval Halting problem.

Mohli bychom tedy vytvořit podprogram H, deklarovaný jako

Bool H(String kod, String vstup)

kde $H(P, x)$ vrátí:

- true pokud se program P zastaví pro vstup x,
- false pokud se program P nezastaví pro vstup x.

Poznámka: Řekněme, že podprogram $H(P, x)$ by vracel false v případě, že P není syntakticky správný kód programu, nebo pokud by došlo k nějaké chybě za běhu programu P na vstupu x.

Halting Problem

S použitím podprogramu H bychom vytvořili program D, který bude provádět následující kroky:

- Načte svůj vstup do proměnné x typu String.
- Zavolá podprogram $H(x, x)$.
- Pokud podprogram H vrátil true, skočí do nekonečné smyčky

```
loop: goto loop
```

V případě, že H vrátil false, program D se ukončí.

Co udělá program D, pokud mu předložíme jako vstup jeho vlastní kód?

Halting Problem

Pokud D dostane jako vstup svůj vlastní kód, tak se buď zastaví nebo nezastaví.

- Pokud se D zastaví, tak $H(D, D)$ vrátí true a D skočí do nekonečné smyčky. Spor!
- Pokud se D nezastaví, tak $H(D, D)$ vrátí false a D se zastaví. Spor!

V obou případech dospějeme ke sporu a další možnost není. Nemůže tedy platit předpoklad, že H řeší Halting problem.

Částečně rozhodnutelné problémy

Rozhodovací problém P je **částečně rozhodnutelný**, jestliže existuje algoritmus $\mathcal{A}$, který:

- Pokud dostane jako vstup instanci problému P, pro kterou je správná odpověď ANO, tak se na tomto vstupu po konečném počtu kroků zastaví a dá odpověď ANO.
- Pokud dostane jako vstup instanci problému P, pro kterou je správná odpověď NE, tak se na tomto vstupu buď zastaví a dá odpověď NE nebo se na tomto vstupu nikdy nezastaví.

Částečně rozhodnutelné problémy

Poznámky:

- Algoritmus $\mathcal{A}$, který částečně rozhoduje problém P, se tedy nemusí zastavit pro všechny vstupy, ale jen pro vstupy, pro které je odpověď ANO.
- Algoritmus $\mathcal{A}$ také samozřejmě nesmí dávat chybné odpovědi pro vstupy, pro které zastaví.
- Je očividné, že každý rozhodnutelný problém je také částečně rozhodnutelný:
  — Algoritmus $\mathcal{A}$, který daný rozhodovací problém P řeší, jej také částečně rozhoduje.
- Existují částečně rozhodnutelné problémy, které nejsou rozhodnutelné.

Částečně rozhodnutelné problémy

Příklad: **Halting problem (HP)** je typickým příkladem problému, který je částečně rozhodnutelný, ale není rozhodnutelný.

Algoritmus $\mathcal{A}$, který jej částečně rozhoduje:

- Načíst program P a vstupní data x.
- Simulovat činnost programu P na vstupních datech x krok po kroku.
- Pokud tato simulace skončí (tj. pokud se výpočet programu P nad daty x po nějakém konečném počtu kroků zastaví), vypsat odpověď ANO.
- Pokud se výpočet programu P nad vstupními daty x nikdy nezastaví, poběží i simulace donekonečna.

  — To je v pořádku, protože správná odpověď je v tomto případě NE a algoritmus $\mathcal{A}$ tedy může pro vstup tvořený dvojicí P a x běžet donekonečna.

Doplňkové problémy

Doplňkový problém k danému rozhodovacímu problému P je problém, kde vstupy jsou stejné jako u problému P a otázka je negací otázky z problému P.

Příklady:

- Doplňkový problém k Halting problému:

Vstup: Zdrojový kód programu P v jazyce $\mathcal{L}$, vstupní data x.

Otázka: Pokud program P dostane jako vstup data x, poběží do nekonečna (tj. nezastaví se na nich)?

- Doplňkový problém k problému SAT:

Vstup: Booleovská formule φ.

Otázka: Je formule φ nesplnitelná (tj. je kontradikcí)?

Doplňkové problémy

- Pokud je P nějaký rozhodovací problém, jeho doplňkový problém budeme označovat is a decision problem, its complement problem will be denoted $\overline{P}$.

- Je zjevné, že problém P je rozhodnutelný právě tehdy, když jeho doplňkový problém $\overline{P}$ je rozhodnutelný:

  — Z algoritmu $\mathcal{A}$, který řeší problém P, snadno dostaneme algoritmus $\mathcal{A}'$ řešící problém $\overline{P}$ tak, že $\mathcal{A}'$ pro daný vstup zavolá $\mathcal{A}$ jako podprogram, jím vrácenou odpověď zneguje a tuto znegovanou odpověď vrátí jako výstup.

  — Zcela analogicky můžeme algoritmus řešící problém $\overline{P}$ upravit na algoritmus řešící problém P.

- Pokud však problém P rozhodnutelný není, není možné, aby problémy P i $\overline{P}$ byly oba částečně rozhodnutelné.

  Plyne to z následující věty.

Postova věta

Postova věta

Pro každý rozhodovací problém P platí, že jestliže problém P i jeho doplňkový problém $\overline{P}$ jsou částečně rozhodnutelné, pak je problém P rozhodnutelný.

Důkaz:

- Předpokládejme, že problém P je částečně rozhodnutelný, a že tedy existuje nějaký algoritmus $\mathcal{A}_1$, který jej částečně rozhoduje.

  Můžeme navíc předpokládat, že algoritmus $\mathcal{A}_1$ se zastaví právě pro ty instance problému P, pro které je odpověď ANO.

- Předpokládejme dále, že problém $\overline{P}$ je částečně rozhodnutelný, a že tedy existuje nějaký algoritmus $\mathcal{A}_2$, který jej částečně rozhoduje.

  Můžeme navíc předpokládat, že algoritmus $\mathcal{A}_2$ se zastaví právě pro ty instance problému $\overline{P}$, pro které je odpověď ANO, tedy právě pro ty instance, pro které je v problému P odpověď NE.

Postova věta

- Algoritmus, který by měl řešit problém P, nemůže pracovat tak, že by přímo volal algoritmy $\mathcal{A}_1$ nebo $\mathcal{A}_2$ jako podprogramy, protože pro některé vstupy mohou tyto algoritmy běžet donekonečna.
- Pro každý vstup x problému P se ale vždy jeden z těchto algoritmů po konečném počtu kroků zastaví.
- Algoritmus $\mathcal{A}$ řešící problém P tedy může pracovat tak, že začne simulovat činnost obou algoritmů $\mathcal{A}_1$ a $\mathcal{A}_2$ na daném vstupu x **paralelně**:
  - Pamatuje si konfigurace obou paralelně běžících algoritmů.
  - Na těchto konfiguracích provádí střídavě vždy jeden krok (nebo i více kroků) algoritmu $\mathcal{A}_1$ a jeden krok (nebo více kroků) algoritmu $\mathcal{A}_2$.
  - Jakmile jeden z těchto algoritmů dosáhne koncové konfigurace, činnost algoritmu $\mathcal{A}$ se ukončí a algoritmus $\mathcal{A}$ vrátí příslušnou odpověď:
    ANO — pokud se zastavil $\mathcal{A}_1$, NE — pokud se zastavil $\mathcal{A}_2$

Postova věta

Z Postovy věty plyne následující:

Doplňkový problém k Halting problému není částečně rozhodnutelný.

Důkaz:

- Halting problem je částečně rozhodnutelný.
- Pokud by i jeho doplňkový problém byl částečně rozhodnutelný, tak podle Postovy věty by byl Halting problem rozhodnutelný.
- To je ovšem ve sporu s již dříve dokázaným faktem, že Halting problem je nerozhodnutelný. Doplňkový problém k Halting problému tedy nemůže být částečně rozhodnutelný.

Stejnou úvahou je možné pro každý částečně rozhodnutelný problém P, který není rozhodnutelný, zdůvodnit, že jeho doplňkový problém $\overline{P}$ není částečně rozhodnutelný.

Částečná rozhodnutelnost — alternativní charakterizace

Částečnou rozhodnutelnost problému P je alternativně možné charakterizovat následujícím způsobem:

- Předpokládejme, že In je množina vstupů problému P.
- Předpokládajme dále, že $\mathcal{W}$ je množina **potenciálních svědků** toho, že pro danou instanci $x \in In$ je správná odpověď ANO:

Problém P je **částečně rozhodnutelný** právě tehdy, když existuje algoritmus $\mathcal{A}$, který pro každou dvojici (x, w), kde $x \in In$ a $w \in \mathcal{W}$, po konečném počtu kroků určí, zda w je **skutečným svědkem** dosvědčujícím, že odpoveď pro x je skutečně ANO, tj.

odpověď pro $x \in In$ je ANO

$$\Longleftrightarrow$$

existuje nějaký **skutečný svědek** $w \in \mathcal{W}$

Částečná rozhodnutelnost — alternativní charakterizace

Pro jednoduchost předpokládejme, že In i $\mathcal{W}$ jsou množiny slov nad nějakou abecedou Σ (např. $\{0, 1\}$).

- Předpokládejme, že pro problém P existuje algoritmus $\mathcal{A}$, který se zastaví (a vydá výstup ANO) právě pro ty vstupy $x \in In$, pro které je odpoveď ANO:
  - Jako množinu potenciálních svědků $\mathcal{W}$ můžeme uvažovat zápisy výpočtů algoritmu $\mathcal{A}$ nad různými vstupy (ve formě posloupnosti konfigurací).
  - Posloupnost konfigurací w bude (skutečným) svědkem pro x právě tehdy, bude-li w korektním zápisem konečného výpočtu algoritmu $\mathcal{A}$ nad vstupem x, který dá výstup ANO.

Částečná rozhodnutelnost — alternativní charakterizace

- Předpokládejme, že pro problém P existuje množina potenciálních svědků $\mathcal{W}$ a algoritmus $\mathcal{A}$, který pro každou dvojici (x, w), kde $x \in In$ a $w \in \mathcal{W}$ umí rozhodnout, zda je w skutečným svědkem toho, že pro x je odpověď ANO.

  Můžeme pak sestrojit algoritmus $\mathcal{A}'$, který bude částečně rozhodovat problém P:

  - Algoritmus $\mathcal{A}'$ dostane vstup $x \in In$.
  - Algoritmus $\mathcal{A}'$ bude postupně generovat všechny potenciální svědky, tj. všechny prvky z množiny $\mathcal{W}$, jako posloupnost $w_0, w_1, w_2, \ldots$ (např. všechna slova nad danou abecedou v pořadí podle délky a v rámci stejné délky v lexikografickém pořadí)
  - Pro každého vygenerovaného potenciálního svědka w_i zavolá pro dvojici (x, w_i) jako podprogram algoritmus $\mathcal{A}$.
    Pokud ten vrátí odpověď ANO, algoritmus $\mathcal{A}'$ skončí s odpovědí ANO.
    Jinak bude pokračovat generováním dalšího potenciálního svědka.

Částečná rozhodnutelnost — alternativní charakterizace

Ještě jiná charakterizace částečně rozhodnutelných problémů vypadá takto:

Problém P je **částečně rozhodnutelný** právě tehdy, pokud existuje algoritmus $\mathcal{A}$, který:

- Neočekává nic na vstupu.
- Běží do nekonečna.
- Jako svůj výstup postupně vypisuje posloupnost instancí problému P:

  $$x_0, x_1, x_2, \ldots$$

  Jednotlivé instance jsou od sebe odděleny nějakým vhodným způsobem, např. nějakým speciálním znakem, aby bylo poznat, kdy bylo dokončeno vypsání každé jednotlivé instance.
- Tato posloupnost je tvořena právě těmi instancemi problému P, pro které je odpověď ANO, tj. každá taková instance se po nějakém konečném počtu kroků algoritmu $\mathcal{A}$ v této posloupnosti objeví.

Částečná rozhodnutelnost — alternativní charakterizace

Pokud máme takový algoritmus $\mathcal{A}$ generující (potenciálně nekonečnou) posloupnost všech instancí problému P, pro které je odpověď ANO, můžeme pomocí něj snadno sestrojit algoritmus $\mathcal{A}'$, který částečně rozhoduje problém P:

- Algoritmus $\mathcal{A}'$ načte vstup x a uloží si jej do paměti.
- Algoritmus $\mathcal{A}'$ začne simulovat jednotlivé kroky algoritmu $\mathcal{A}$.
- Vždy, když $\mathcal{A}$ vygeneruje další instanci x_i problému P, je simulace přerušena a $\mathcal{A}'$ otestuje, zda $x_i = x$.

  Pokud platí $x_i = x$, činnost programu $\mathcal{A}'$ skončí a $\mathcal{A}'$ vydá výstup ANO.

  Pokud platí $x_i \neq x$, bude $\mathcal{A}'$ pokračovat v simulaci algoritmu $\mathcal{A}$, dokud nebude vygenerována další instance, pro kterou je odpověď ANO, a celý cyklus se znova opakuje.

Částečná rozhodnutelnost — alternativní charakterizace

Naopak, pokud máme algoritmus $\mathcal{A}$, který částečně rozhoduje problém P (tj. zastaví se a vrátí odpověď ANO právě pro ty vstupy, pro které je odpověď ANO), je možné vytvořit algoritmus $\mathcal{A}'$, který poběží do nekonečna a bude postupně generovat posloupnost všech instancí problému P, pro které je odpověď ANO:

- Algoritmus $\mathcal{A}'$ bude postupně generovat všechny instance problému P z množiny In.
- Algoritmus $\mathcal{A}'$ bude simulovat činnost algoritmu $\mathcal{A}$ na těchto instancích.
- Algoritmus $\mathcal{A}'$ to ale nemůže dělat tak, že by začal simulovat činnost algoritmu $\mathcal{A}$ na dané instanci x_i a simuloval ho tak dlouho, až tento výpočet skončí, protože daný výpočet nemusí skončit nikdy, a $\mathcal{A}'$ by se k dalším instancím nikdy nedostal.

Částečná rozhodnutelnost — alternativní charakterizace

- Algoritmus $\mathcal{A}'$ to bude dělat tak, že činnost algoritmu $\mathcal{A}$ bude simulovat na mnoha vstupech paralelně.

  Bude donekonečna opakovat cyklus, kdy v jedné iteraci tohoto cyklu provede vždy následující:

  - Odsimuluje jeden krok výpočtu pro každý dosud simulovaný výpočet.
  - Množinu dosud běžících simulovaných výpočtů rozšíří o další simulovaný výpočet algoritmu $\mathcal{A}$, kdy jeho vstupem bude následující instance z množiny In.
  - Jakmile některý ze simulovaných výpočtů skončí s odpovědí ANO, algoritmus $\mathcal{A}$ vypíše vstup tohoto výpočtu na výstup.

    (Aby mohl tento výstup vypsat, musí si algoritmus $\mathcal{A}'$ u každého z momentálně simulovaných výpočtů pamatovat, jak vypadal vstup tohoto výpočtu.)

Částečná rozhodnutelnost — alternativní charakterizace

Následující čtvrtá možnost charakterizace částečně rozhodnutelných problémů je dost podobná té předchozí:

Problém P je **částečně rozhodnutelný** právě tehdy, pokud existuje algoritmus $\mathcal{A}$, který má následující vlastnosti:

- Jako svůj vstup očekává přirozené číslo i (tj. $i \in \mathbb{N}$).
- Pro libovolný vstup $i \in \mathbb{N}$ se algoritmus $\mathcal{A}$ po konečném počtu kroků zastaví a vydá jako svůj výstup nějakou instanci problému P, pro kterou je odpověď Ano.
- Pokud zápisem $f(i)$ označíme výstup algoritmu $\mathcal{A}$ pro vstup i, bude platit, že nekonečná posloupnost

$$f(0), f(1), f(2), \ldots$$

bude obsahovat všechny instance problému P, pro které je odpověď Ano.

Částečná rozhodnutelnost — alternativní charakterizace

Poznámka:

- Výše uvedená charakterizace předpokládá, že existuje alespoň jedna instance problému P, pro kterou je odpoveď ANO.
- Je zjevné, že s použitím s algoritmu $\mathcal{A}$ s výše uvedenými vlastnostmi je možné snadno sestrojit algoritmus, který poběží donekonečna a bude postupně generovat všechny instance problému P, pro které je odpověď ANO.
- Pokud naopak máme algoritmus $\mathcal{A}'$ běžící donekonečna a generující všechny instance, pro které je odpověď ANO, je snadné sestrojit algoritmus $\mathcal{A}$ s výše uvedenými vlastnostmi:
  - Načte číslo i a uloží jej do čítače k.
  - Bude simulovat činnost algoritmu $\mathcal{A}'$ po jednotlivých krocích. Vždy, když $\mathcal{A}'$ vygeneruje další instanci problému P, sníží čítač k o 1.
  - Jakmile čítač k klesne na nulu, bude pokračovat v simulaci algoritmu $\mathcal{A}'$ tak dlouho, dokud nevygeneruje následující instanci.
  - Tuto instanci vydá jako výstup a simulaci ukončí.

Rekurzivní a rekurzivně spočetné množiny

V literatuře se běžně používá také následující terminologie:
Řekněme, že A je množina těch instancí problému P, pro které je odpověď ANO.

- Množina A se nazývá **rekurzivní** (**recursive**), jestliže existuje algoritmus, který pro každou instanci x určí, zda x patří do A, tj. jestliže je problém P algoritmicky rozhodnutelný.
- Množina A se nazývá **rekurzivně spočetná** (**recursively enumerable**), jestliže existuje algoritmus, který bude postupně vypisovat všechny tyto instance, tj. pokud je částečně rozhodnutelné pro každou instanci x, zda daná instance patří do množiny A.

Převody mezi problémy

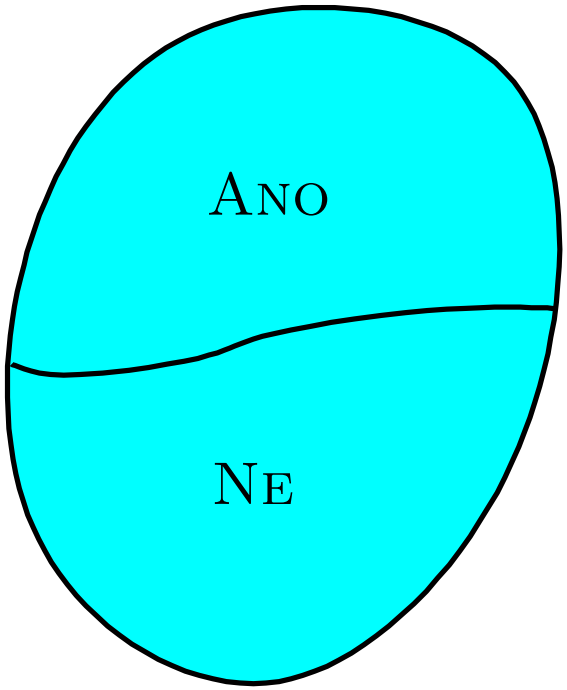
Pokud máme o nějakém (rozhodovacím) problému dokázáno, že je nerozhodnutelný, můžeme ukázat nerozhodnutelnost dalších problémů pomocí **redukcí** (**převodů**) mezi problémy.

Problém P_1 je **převeditelný** na problém P_2, jestliže existuje algoritmus Alg takový, že:

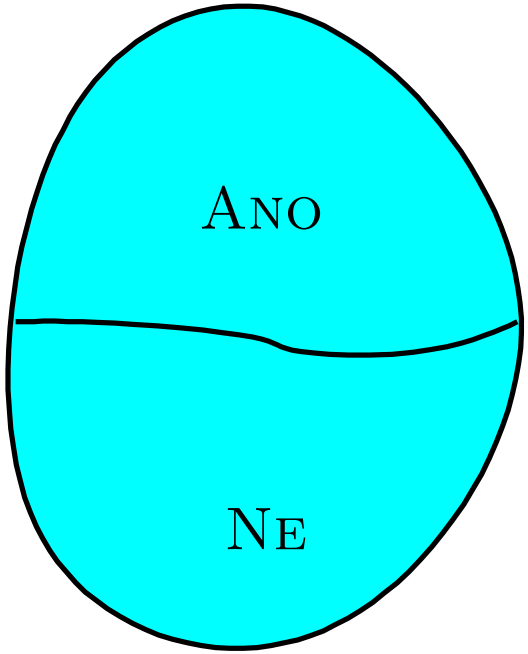
- Jako vstup může dostat libovolnou instanci problému P_1.
- K instanci problému P_1, kterou dostane jako vstup (označme ji w), vyprodukuje jako svůj výstup instanci problému P_2 (označme ji $Alg(w)$).
- Platí, že pro vstup w je v problému P_1 odpověď ANO právě tehdy, když pro vstup $Alg(w)$ je v problému P_2 odpověď ANO.

Převody mezi problémy

vstupy problému P_1



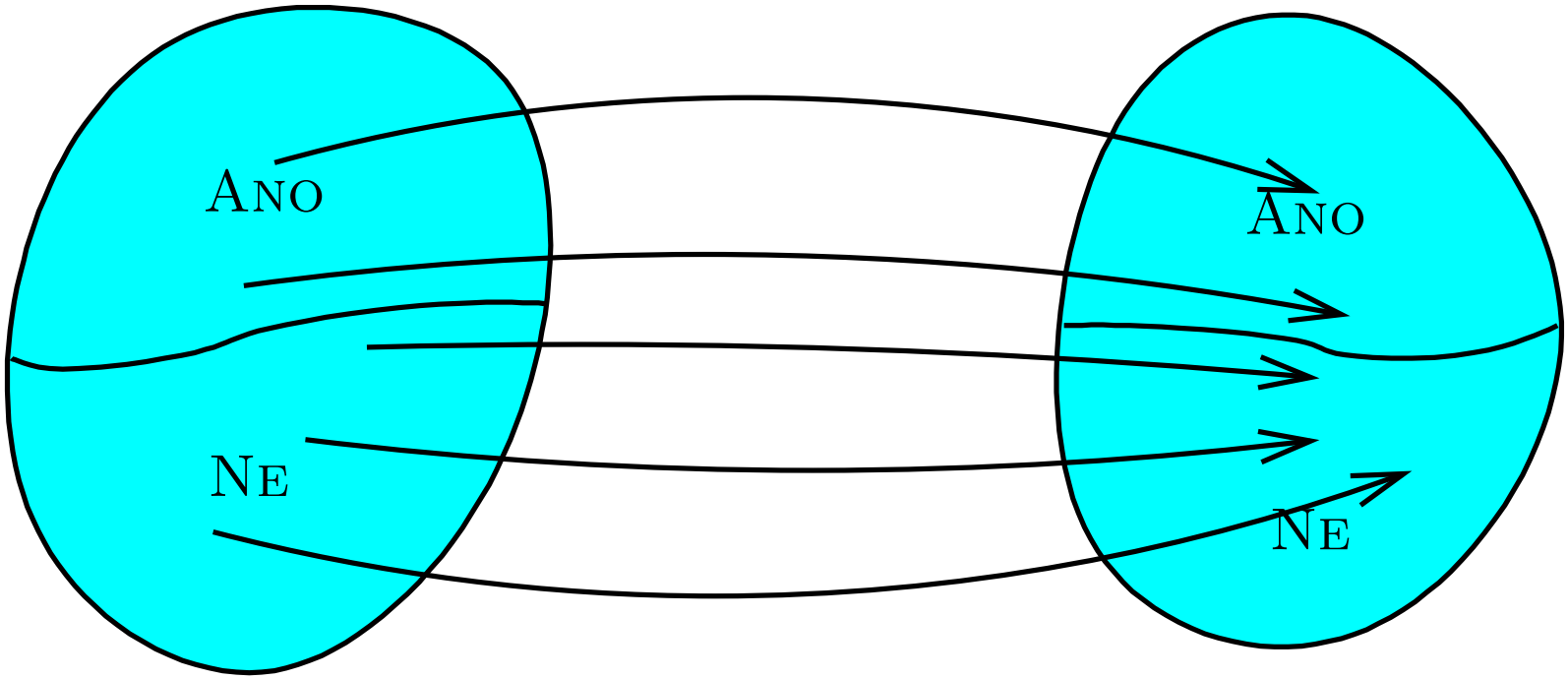
vstupy problému P_2



Převody mezi problémy

vstupy problému P_1

vstupy problému P_2



Alg

Převody mezi problémy

Řekněme, že existuje redukce Alg problému P_1 na problém P_2.

Pokud by problém P_2 byl rozhodnutelný, pak i problém P_1 je rozhodnutelný.

Řešení problému P_1 pro vstup x:

- Zavoláme Alg se vstupem x, vrátí nám hodnotu $Alg(x)$.
- Zavoláme algoritmus řešící problém P_2 se vstupem $Alg(x)$.
- Hodnotu, kterou nám vrátí, vypíšeme jako výsledek.

Je zřejmé, že pokud P_1 je nerozhodnutelný, tak P_2 nemůže být rozhodnutelný.

Halting problem

Pro účely důkazů nerozhodnutelnosti dalších problému pomocí redukcí se Halting problem často používá v následující podobě:

Halting problem

Vstup: Popis Turingova stroje $\mathcal{M}$ a slovo w.

Otázka: Zastaví se stroj $\mathcal{M}$ po nějakém konečném počtu kroků, pokud dostane jako svůj vstup slovo w?

Halting problem

Tento problém je nerozhodnutelný i v případě, kdy předpokládáme, že vstupem pro stroj $\mathcal{M}$ je prázdné slovo ε:

Halting problem (kde vstup je ε)

Vstup: Popis Turingova stroje $\mathcal{M}$.

Otázka: Zastaví se stroj $\mathcal{M}$ po nějakém konečném počtu kroků, pokud dostane jako svůj vstup slovo ε?

Redukce ze standardního Halting problému na tuto variantu je jednoduchá.

K danému stroji $\mathcal{M}$ se vstupem w sestrojíme stroj $\mathcal{M}'$, který:

- Zapíše na pásku slovo w a přesune hlavu zpět na začátek.
- Začne se chovat jako $\mathcal{M}$.

Halting problem

U Halting problému používaného při redukcích, které slouží k důkazům nerozhodnutelnosti dalších problémů, může být někdy výhodné předpokládat různá další omezení na daný Turingův stroj $\mathcal{M}$, např.:

- že používá jen jednu pásku, která je jednostranně nekonečná
- že používá páskovou abecedu $\{0, 1\}$
- že po skončení výpočtu se hlava nachází na stejné pozici, na jaké se nacházela na začátku
- že po skončení je obsah pásky prázdný
- ...

Halting problem

Halting problém se také při redukcích často používá ve variantě, kdy je místo Turingova stroje použit Minského stroj:

Halting problem (pro Minského stroj)

Vstup: Popis Minského stroje $\mathcal{M}$.

Otázka: Zastaví se daný stroj $\mathcal{M}$ po konečném počtu kroků pokud začne v konfiguraci, kde všechny čítače budou na začátku obsahovat hodnotu 0?

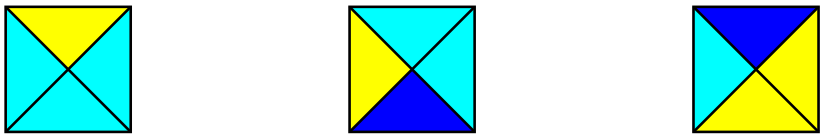
Poznámka: Také zde může být výhodné používat různé zjednodušující předpoklady, např.:

- že se stroj nikdy nepokusí snížit o 1 čítač, jehož hodnota je 0
- že čítače jsou jen dva
- že po skončení výpočtu všechny čítače obsahují hodnotu 0

Kachličkování roviny

Ukážeme si jiný příklad **nerozhodnutelného** problému.

Vstupem je množina typů kachliček, jako třeba:

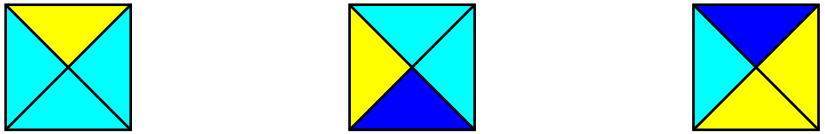


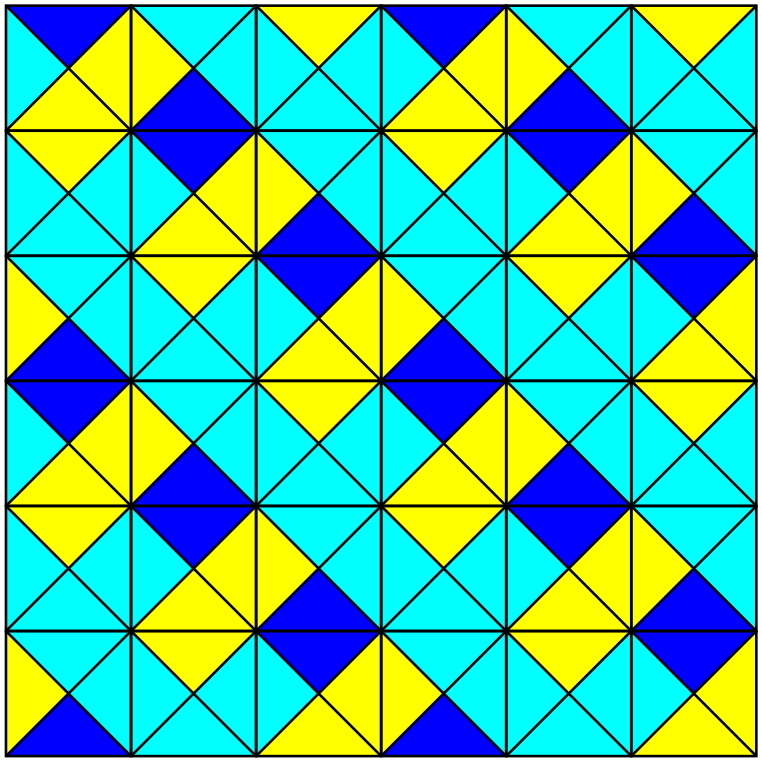
Otázka je, zda je možné použitím daných typů kachliček pokrýt celou nekonečnou rovinu tak, aby všechny kachličky spolu sousedily stejnými barvami.

Poznámka: Můžeme předpokládat, že máme v zásobě neomezené množství kachliček všech typů.

Kachličky není dovoleno otáčet.

Kachličkování roviny





Kachličkování roviny

Více formálně můžeme tento problém popsat takto:

- Předpokládejme, že C je nějaká konečná množina **barev**.
- Množina $\{\mathtt{N}, \mathtt{S}, \mathtt{E}, \mathtt{W}\}$ představuje čtyři **směry** — sever, jih, východ, západ.
- **Typ kachličky** je dán jako přiřazení barev jednotlivým směrům, tj. jako funkce $\tau : \{\mathtt{N}, \mathtt{S}, \mathtt{E}, \mathtt{W}\} \to C$.
- Předpokládejme, že máme dánu množinu typů kachliček $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$.
- **Pokrytí roviny** kachličkami je funkce $p : \mathbb{Z} \times \mathbb{Z} \to \mathcal{T}$ splňující následující dvě podmínky pro každé $i, j \in \mathbb{Z}$:
  - Pokud $p(i,j) = \tau$ a $p(i+1,j) = \tau'$, tak $\tau(\mathtt{E}) = \tau'(\mathtt{W})$.
  - Pokud $p(i,j) = \tau$ a $p(i,j+1) = \tau'$, tak $\tau(\mathtt{N}) = \tau'(\mathtt{S})$.

Kachličkování roviny

Uvažujme následující variantu problému:

Vstup: Množina typů kachliček $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ a počáteční kachlička $\tau_0 \in \mathcal{T}$.

Otázka: Existuje nějaké pokrytí roviny p kachličkami z množiny $\mathcal{T}$ takové, že $p(0,0) = \tau_0$?

V této variantě je tedy jeden z typů kachliček vyčleněn jako speciální a ptáme se, zda je možné pokrýt celou rovinu tak, aby byl tento typ použit. (Ostatní typy kachliček mohou a nemusí být použity.)

Kachličkování roviny

Nerozhodnutelnost tohoto problému (resp. doplňkového problému k tomuto problému) je možné dokázat například pomocí redukce z Halting problému v následující variantě :

Vstup: Turingův stroj $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$.

Otázka: Zastaví se Turingův stroj $\mathcal{M}$ po konečném počtu kroků, pokud jako vstup dostane prázdné slovo ε ?

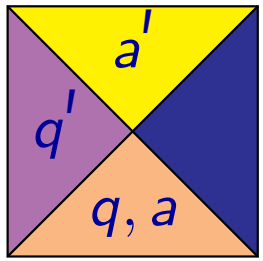
Popíšeme algoritmus, který:

- Dostane jako vstup popis Turingova stroje $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, \{q_f\})$.
- K danému stroji $\mathcal{M}$ vyrobí (a vydá jako výstup) množinu typů kachliček $\mathcal{T}$ se speciální vyčleněnou kachličkou $\tau_0 \in \mathcal{T}$.
- Bude platit: Celou rovinu bude možné pokrýt použitím kachliček z $\mathcal{T}$ tak, aby na pozici $(0, 0)$ byla kachlička τ_0, právě tehdy, když se stroj $\mathcal{M}$ na vstupu ε nikdy nezastaví (tj. jeho výpočet bude nekonečný).

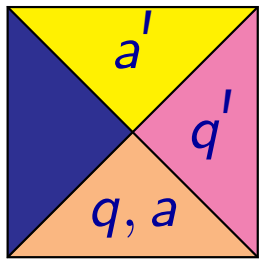
Kachličkování roviny

Algoritmus vyrobí kachličky pro daný stroj $\mathcal{M}$ následujícím způsobem (v následujícím popisu kachliček jsou kromě barev použity také názvy prvků z množin Q, Γ a $(Q \times \Gamma)$ — nahrazení těchto názvů prvků dalšími barvami je přímočaré):

- Pro každé $q, q' \in Q - F$ a $a, a' \in \Gamma$, kde $\delta(q, a) = (q', a', -1)$, přidáme následující typ kachličky:

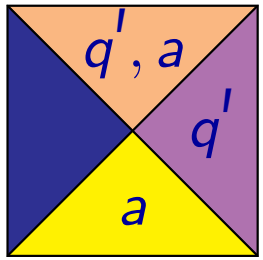


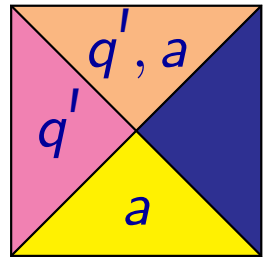
- Pro každé $q, q' \in Q - F$ a $a, a' \in \Gamma$, kde $\delta(q, a) = (q', a', +1)$, přidáme následující typ kachličky:



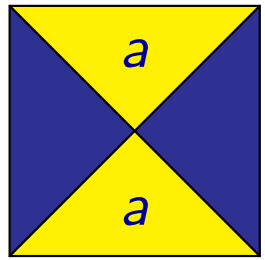
Kachličkování roviny

- Pro každé $q' \in Q$ a $a \in \Gamma$ přidáme následující dva typy kachliček:



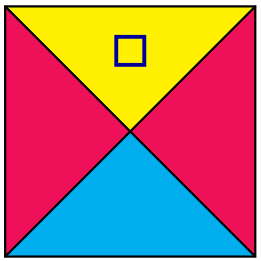


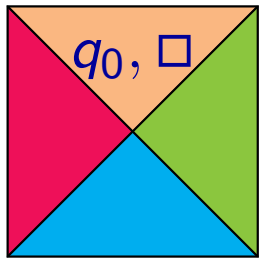
- Pro každé $a \in \Gamma$ přidáme následující typ kachliček:

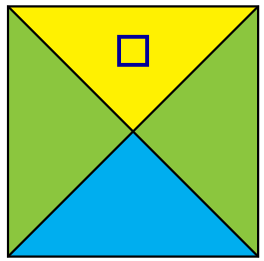


Kachličkování roviny

- Přidáme následující tři typy kachliček
  (prostřední z nich bude vyčleněna jako **počáteční** kachlička τ_0):

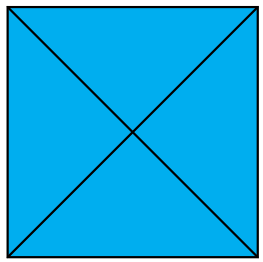






  (Symbol $\square \in \Gamma$ zde reprezentuje symbol **blank** Turingova stroje of the Turing machine $\mathcal{M}$.)

- Nakonec přidáme také „prázdný“ typ kachličky:



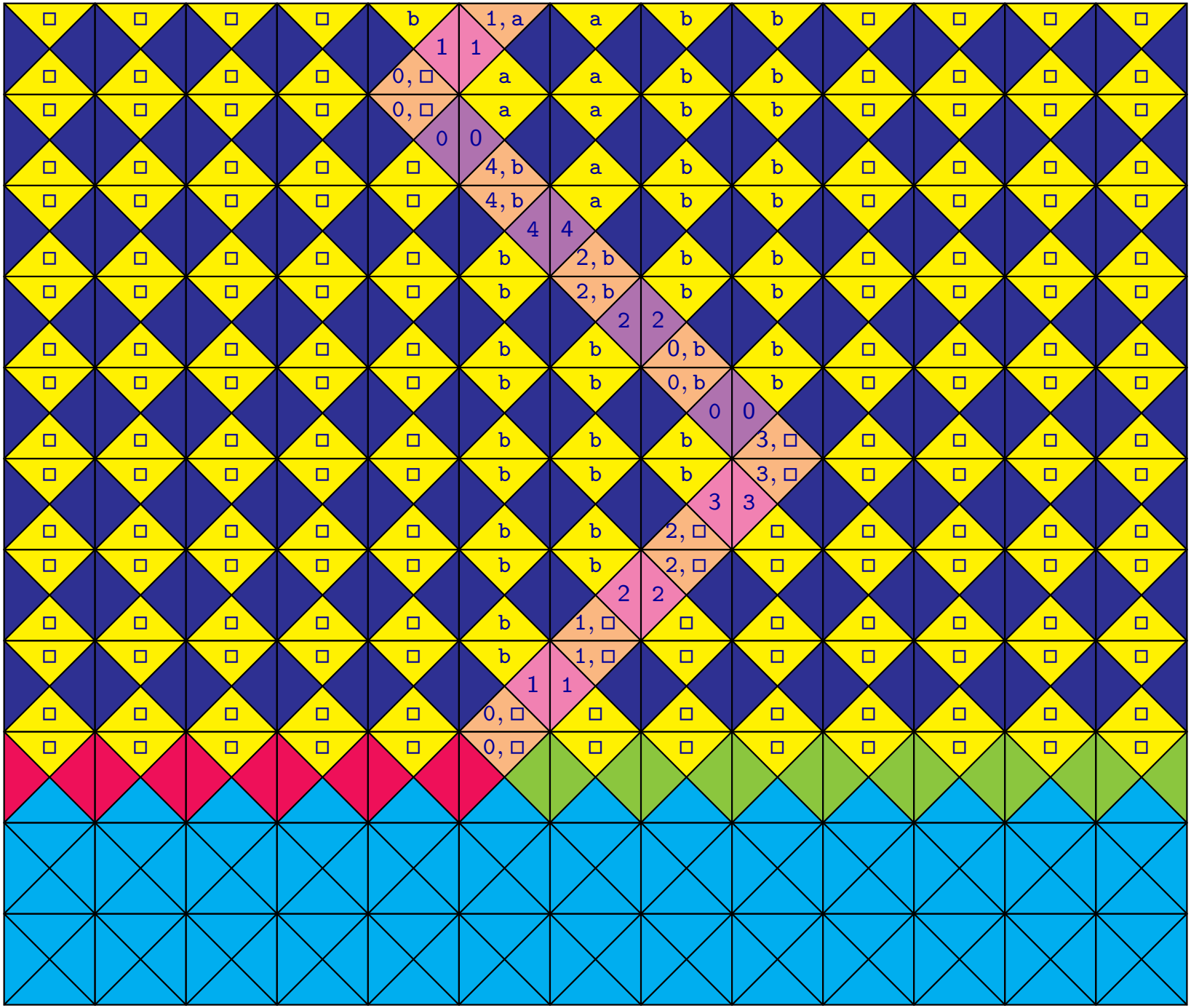
Kachličkování roviny

Řekněme, že výpočet Turingova stroje nad prázdným slovem je tvořen posloupností konfigurací

$$\alpha_0, \alpha_1, \alpha_2, \ldots$$

Není těžké si promyslet následující ohledně vyplňování roviny kachličkami výše popsaných typů (když na pozici $(0, 0)$ bude uvedená kachlička τ_0):

- Barvy na horních okrajích kachliček v řádku 0 budou muset odpovídat konfiguraci α_0.
- To vynutí, že barvy na horních okrajích kachliček v řádku 1 budou muset odpovídat konfiguraci α_1.
- To vynutí, že barvy na horních okrajích kachliček v řádku 2 budou muset odpovídat konfiguraci α_2.
- ...


□ □ □ □ b 1, a a b b □ □ □ □
1 1
□ □ □ □ 0, □ a a b b □ □ □ □
□ □ □ □ 0, □ a a b b □ □ □ □
0 0
□ □ □ □ □ 4, b a b b □ □ □ □
□ □ □ □ □ 4, b a b b □ □ □ □
4 4
□ □ □ □ □ b 2, b b b □ □ □ □
□ □ □ □ □ b 2, b b b □ □ □ □
2 2
□ □ □ □ □ b b 0, b b □ □ □ □
□ □ □ □ □ b b 0, b b □ □ □ □
0 0
□ □ □ □ □ b b b 3, □ □ □ □ □
□ □ □ □ □ b b b 3, □ □ □ □ □
3 3
□ □ □ □ □ b b 2, □ □ □ □ □ □
□ □ □ □ □ b b 2, □ □ □ □ □ □
2 2
□ □ □ □ □ b 1, □ □ □ □ □ □ □
□ □ □ □ □ b 1, □ □ □ □ □ □ □
1 1
□ □ □ □ □ 0, □ □ □ □ □ □ □ □
□ □ □ □ □ 0, □ □ □ □ □ □ □ □

Kachličkování roviny

Obecně tedy bude muset platit pro každé i, kde $i \geq 0$:

- Barvy na horních okrajích kachliček v řádku i odpovídají konfiguraci α_i.

To je ale možné, jen pokud je výpočet stroj $\mathcal{M}$ nad vstupem ε nekonečný.

V případě, že bude dosažena koncová konfigurace (např. na řádku t), následující řádek $(t + 1)$ nebude možné doplnit.

Pokud výpočet bude nekonečný:

- máme možnost vyplnit všechny řádky i, kde $i \geq 0$.

Bez ohledu na to, jestli je výpočet konečný nebo nekonečný a jak přesně bude vypadat vyplnění řádků 0, 1, 2, . . . , řádky -1, -2, -3, . . . je možné vyplnit „prázdnými“ kachličkami.

Kachličkování roviny

Vidíme tedy, že platí následující:

- Jestliže je výpočet $\mathcal{M}$ nad ε nekonečný, je možné sestrojenou sadou kachliček vyplnit celou rovinu.
- Jestliže se výpočet $\mathcal{M}$ nad ε po konečném počtu kroků zastaví, celou rovinu danými kachličkami vyplnit nelze.

Pokud by tedy existoval algoritmus, který by uměl pro libovolnou sadu kachliček (a danou počáteční kachličku) určit, zda je možné pomocí ní vyplnit celou rovinu, bylo by možné tento algoritmus použít i pro řešení Halting problému.

To ale nelze (už víme, že neexistuje algorimus, který by řešil Halting problém), takže žádný takový algoritmus existovat nemůže.

Kachličkování roviny

Poznámka: Dá se dokázat, že problém kachličkování roviny je nerozhodnutelný i ve variantě, kdy není specifikována žádná „počáteční“ kachlička:

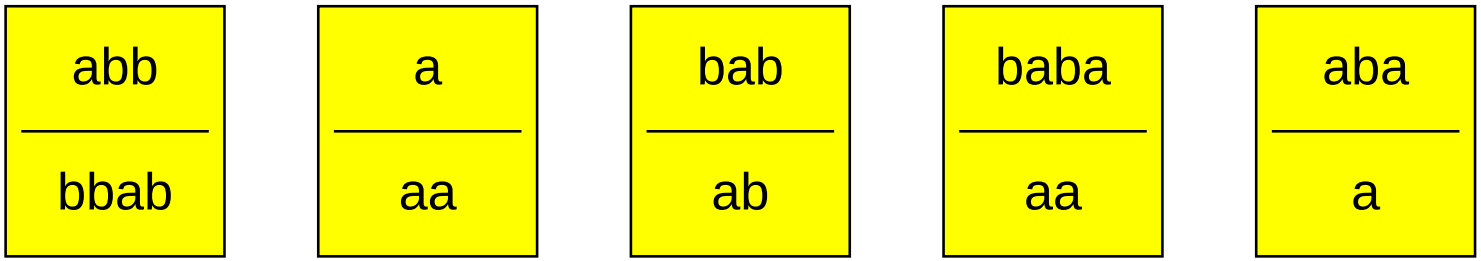
Vstup: Množina typů kachliček $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$.

Otázka: Existuje nějaké pokrytí roviny kachličkami z množiny $\mathcal{T}$?

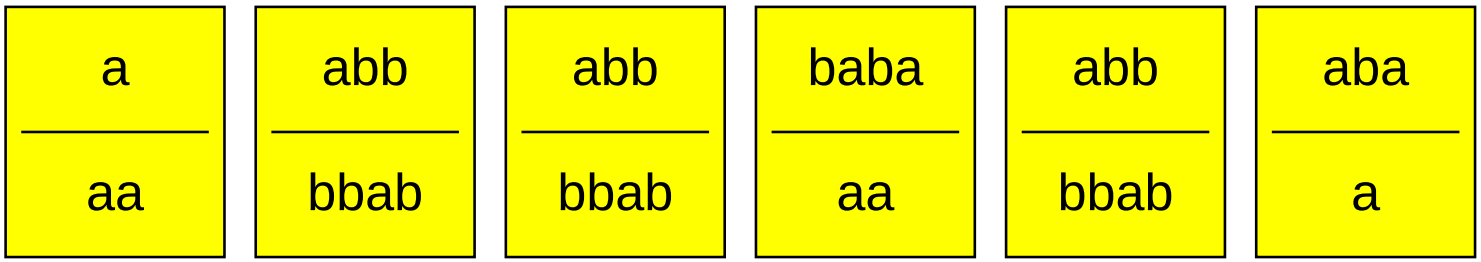
Důkaz je technicky komplikovanější, ale je také postaven na podobných myšlenkách, jaké byly uvedeny výše — tj. kódování výpočtu Turingova stroje, kdy je rovinu možné pokrýt jen v případě, že je výpočet daného stroje nekonečný.

Postův korespondenční problém

Vstupem je množina typů kartiček, jako třeba:



Otázka je, zda je možné z těchto typů kartiček vytvořit neprázdnou konečnou posloupnost, kde zřetězením slov nahoře i dole vznikne totéž slovo. Každý typ kartičky je možné používat opakovaně.



Nahoře i dole vznikne slovo `aabbabbbabaabbaba`.

Postův korespondenční problém

Tento problém se označuje jako **Postův korespondenční problém** (**Post Correspondence Problem** — **PCP**):

Postův korespondenční problém (PCP)

Vstup: Posloupnosti slov $u_1, u_2, \ldots, u_n$ a $v_1, v_2, \ldots, v_n$ nad nějakou abecedou Σ.

Otázka: Existuje nějaká posloupnost $i_1, i_2, \ldots, i_m$, kde $m \geq 1$, kde pro každé i_j platí $1 \leq i_j \leq n$, a kde

$$u_{i_1} u_{i_2} \cdots u_{i_m} = v_{i_1} v_{i_2} \cdots v_{i_m} \, ?$$

Postův korespondenční problém

Nerozhodnutelnost tohoto problému se dá dokázat redukcí z Halting problému.

Při popisu této redukce je výhodné použít jako mezikrok následující variantu Postova korespondenčního problému, kde je jedna z kartiček předepsána jako počáteční:

Iniciální Postův korespondenční problém (IPCP)

Vstup: Posloupnosti slov $u_1, u_2, \ldots, u_n$ a $v_1, v_2, \ldots, v_n$ nad nějakou abecedou Σ.

Otázka: Existuje nějaká posloupnost $i_1, i_2, \ldots, i_m$, kde $m \geq 1$, kde pro každé i_j platí $1 \leq i_j \leq n$, a kde

$$u_{i_1} u_{i_2} \cdots u_{i_m} = v_{i_1} v_{i_2} \cdots v_{i_m}$$

a kde navíc platí $i_1 = 1$?

Postův korespondenční problém

Redukce HP na IPCP:

- Algoritmus dostane jako vstup popis Turingova stroje $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ a jeho vstupu $w = a_1 a_2 \dots a_n$.
- Algoritmus vytvoří instanci IPCP, tj. sadu kartiček.
- Vytvořená instance IPCP bude mít řešení právě tehdy, když se stroj $\mathcal{M}$ nad vstupem w zastaví.

  Toto řešení bude vypadat tak, že společné slovo vytvořené v horním i dolním řádku v tomto řešení bude v podstatě popis výpočtu stroje $\mathcal{M}$ nad slovem w:

  - bude to posloupnost zápisů jednotlivých konfigurací
  - jednotlivé konfigurace budou odděleny speciálním znakem #

Postův korespondenční problém

- První kartička, kterou se bude muset začít, bude vypadat takto:

$$\left[\frac{\#}{\#q_0a_1a_2\cdots a_n\#}\right]$$

- Pro každé $q, q' \in Q$ a $a, a' \in \Gamma$, kde $\delta(q, a) = (q', a', +1)$, se přidá kartička:

$$\left[\frac{qa}{a'q'}\right]$$

- Pro každé $q, q' \in Q$ a $a, a', b \in \Gamma$, kde $\delta(q, a) = (q', a', -1)$, se přidá kartička:

$$\left[\frac{bqa}{q'ba'}\right]$$

Postův korespondenční problém

- Pro každé $a \in \Gamma$ se přidá kartička:

$$\left[\frac{a}{a}\right]$$

- Přidají se kartičky:

$$\left[\frac{\#}{\#}\right] \qquad \left[\frac{\#}{\square\#}\right]$$

- Pro každé $a \in \Gamma$ a $q_f \in F$ se přidají kartičky:

$$\left[\frac{aq_f}{q_f}\right] \qquad \left[\frac{q_f a}{q_f}\right] \qquad \left[\frac{q_f\#\#}{\#}\right]$$

Postův korespondenční problém

Redukovat IPCP na PCP je pak možné následujícím způsobem:

- Místo každé kartičky tvaru

$$\left[\frac{a_1 a_2 \cdots a_k}{b_1 b_2 \cdots b_\ell}\right]$$

se přidá kartička tvaru

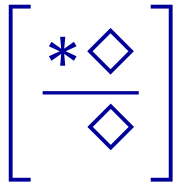
$$\left[\frac{*a_1 * a_2 * \cdots * a_k}{b_1 * b_2 * \cdots * b_\ell *}\right]$$

- Pro první kartičku, kterou je třeba začít, se přidá také kartička tvaru

$$\left[\frac{*a_1 * a_2 * \cdots * a_k}{*b_1 * b_2 * \cdots * b_\ell *}\right]$$

Postův korespondenční problém

- Přidá se kartička



Poznámka: Předpokládá se, že znaky $*$ a $\diamond$ jsou nějaké nové speciální znaky, které nejsou použity v původní instanci IPCP.

Postův korespondenční problém

Redukcí z Postova korespondenčního problému se dá například snadno ukázat nerozhodnutelnost některých problémů z oblasti bezkontextových gramatik:

Problém

Vstup: Bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$.

Otázka: Je $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) = \varnothing$?

Problém

Vstup: Bezkontextová gramatika $\mathcal{G}$.

Otázka: Je $\mathcal{G}$ nejednoznačná?

Ekvivalence bezkontextových gramatik

Také následující dva problémy týkající se bezkontextových gramatik jsou nerozhodnutelné:

Problém

Vstup: Bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$.

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

Problém

Vstup: Bezkontextová gramatika $\mathcal{G}$ generující jazyk nad abecedou Σ.

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

Ekvivalence bezkontextových gramatik

Důkaz nerozhodnutelnosti je možné opět provést pomocí redukce z HP.

K danému Turingově stroji $\mathcal{M}$ a jeho vstupu w se vyrobí bezkontextová gramatika $\mathcal{G}$ taková, že:

- pokud se stroj $\mathcal{M}$ na w zastaví, bude $\mathcal{L}(\mathcal{G})$ obsahovat všechna slova s výjimkou slova, které bude zápisem výpočtu stroje $\mathcal{M}$ nad slovem w ve formě posloupnosti konfigurací
- pokud se stroj $\mathcal{M}$ na w nezastaví, bude $\mathcal{L}(\mathcal{G})$ obsahovat všechna slova

Ekvivalence bezkontextových gramatik

Gramatika $\mathcal{G}$ tedy bude generovat právě ta slova, která **nejsou** zápisem výpočtu stroje $\mathcal{M}$ nad slovem w, tj.:

- vůbec nemají tvar posloupnosti konfigurací, nebo
- nezačínají počáteční konfigurací
- nekončí koncovou konfigurací
- existuje v nich nějaká dvojice po sobě jdoucích konfigurací, která neodpovídá tomu, jaký krok by udělal daný Turingův stroj $\mathcal{M}$

Aby bylo možné pomocí bezkontextové gramatiky popsat čtvrtou z výše uvedených podmínek, je třeba použít následující „trik“:

- sudé konfigurace budou zapisovány běžným způsobem zleva doprava
- liché konfigurace budou zapisovány pozpátku, tj. zprava doleva

Aritmetika na přirozených číslech

Problém

Vstup: Uzavřená formule predikátové logiky (prvního řádu), ve které může být použit jako predikátový symbol pouze $=$, jako funkční symboly pouze $+$ a $\cdot$ a jako konstantní symboly pouze 0 a 1.

Otázka: Je daná formule pravdivá v oboru přirozených čísel (při přirozené interpretaci všech funkčních a predikátových symbolů)?

Příklad vstupu:

$$\forall x \exists y \forall z ((x \cdot y = z) \wedge (y + 1 = x))$$

Poznámka: Úzce souvisí s Gödelovou větou o neúplnosti.

Aritmetika na přirozených číslech

Ukážeme, že tento problém je **nerozhodnutelný**.

Pro tento důkaz použijeme redukci z problému zastavení pro Minského stroj:

Vstup: Popis Minského stroje $\mathcal{M}$.

Otázka: Zastaví se daný stroj $\mathcal{M}$ po konečném počtu kroků pokud začne v konfiguraci, kde všechny čítače budou na začátku obsahovat hodnotu 0?

Aritmetika na přirozených číslech

Popišeme algoritmus, který:

- Dostane jako vstup popis Minského stroje $\mathcal{M}$.
- K danému stroji $\mathcal{M}$ sestrojí formuli φ a tuto formuli vydá jako výstup.
- Pro tuto formuli bude platit následující:

  Formule φ bude pravdivá (ve standardní interpretaci na přirozených číslech) právě tehdy, když se stroj $\mathcal{M}$ se zastaví po konečném počtu kroků.

Poznámka: Formuli φ budeme vytvářet postupně.

Budeme ji skládat z jednodušších formulí.

Aritmetika na přirozených číslech

Označme Var nekonečnou spočetnou množinu všech **proměnných**, které se mohou vyskytovat ve formulích — $Var = \{x, y, z, \ldots\}$

Termy jsou definovány následovně:

- Každá proměnná x z množiny Var je dobře utvořený term.
- Konstanty 0 a 1 jsou dobře utvořené termy.
- Pokud t_1 a t_2 jsou dobře utvořené termy, tak i $t_1 + t_2$ a $t_1 \cdot t_2$ jsou dobře utvořené termy.

Formule jsou definovány následovně:

- Pokud t_1 a t_2 jsou dobře utvořené termy, tak $t_1 = t_2$ je dobře utvořená formule.
- Pokud φ_1 a φ_2 jsou dobře utvořené formule, tak i $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$ a $\varphi_1 \leftrightarrow \varphi_2$ jsou dobře utvořené formule.
- Pokud φ je dobře utvořená formule a x proměnná z množiny Var, tak i $\forall x.\varphi$ a $\exists x.\varphi$ jsou dobře utvořené formule.

Aritmetika na přirozených číslech

Poznámky:

- Formule budou interpretovány nad množinou **přirozených čísel** $\mathbb{N} = \{0, 1, 2, \ldots\}$.
- Konstanty 2, 3, 4, $\ldots$ můžeme chápat jako zkratky pro $1 + 1$, $1 + 1 + 1$, $1 + 1 + 1 + 1$, $\ldots$
- Zápis $t_1 \leq t_2$ budeme brát jako zkratku pro

  $$\exists x.(t_1 + x = t_2)$$

  (Předpokládáme, že x se nevyskytuje v t_1, ani v t_2.)
- Podobně zápis $t_1 < t_2$ budeme brát jako zkratku pro

  $$\exists x.(t_1 + x + 1 = t_2)$$

- Analogicky můžeme definovat také $t_1 \geq t_2$ a $t_1 > t_2$.

Aritmetika na přirozených číslech

- $\textsc{Divides}(x, y)$ — x je dělitelem y:

$$\exists k.(x \cdot k = y)$$

- $\textsc{Prime}(p)$ — p je prvočíslo:

$$p > 1 \ \wedge \ \forall x.(\textsc{Divides}(x, p) \ \rightarrow \ (x = 1) \vee (x = p))$$

- $\textsc{Prime-Power}(p, x)$ — x je mocninou prvočísla p (tj. existuje $i \in \mathbb{N}$ takové, že $x = p^i$):

$$\textsc{Prime}(p) \ \wedge \ (x \geq 1) \ \wedge \ \forall y.(\textsc{Divides}(y, x) \ \wedge \ \textsc{Prime}(y) \ \rightarrow \ (y = p))$$

Aritmetika na přirozených číslech

Řekněme, že máme dán Minského stroj $\mathcal{M}$:

- Množina **stavů** řídící jednotky stroje $\mathcal{M}$ je $S = \{0, 1, \ldots, s\}$.
- Počáteční stav je 0
- Koncový stav je s.
- Stroj má r čítačů označených x_1, x_2, $\ldots$, x_r.

Konfigurace stroje $\mathcal{M}$ může být popsána jako $(r+1)$-tice přirozených čísel

$$(q, v_1, \ldots, v_r)$$

kde q reprezentuje aktuální stav řídící jednotky a $v_1, \ldots, v_r$ jsou hodnoty čítačů $x_1, \ldots, x_r$.

Aritmetika na přirozených číslech

Řekněme pro konkrétnost, že stroj $\mathcal{M}$ bude používat např. 3 čítače, tj. $r = 3$.

Můžeme snadno vytvořit formule charakterizující **počáteční** a **koncové** konfigurace:

- $\textsc{Initial-Conf}(q, v_1, v_2, v_3)$ — jedná se o počáteční konfiguraci:

$$(q = 0) \wedge (v_1 = 0) \wedge (v_2 = 0) \wedge (v_3 = 0)$$

- $\textsc{Final-Conf}(q, v_1, v_2, v_3)$ — jedná se o koncovou konfiguraci:

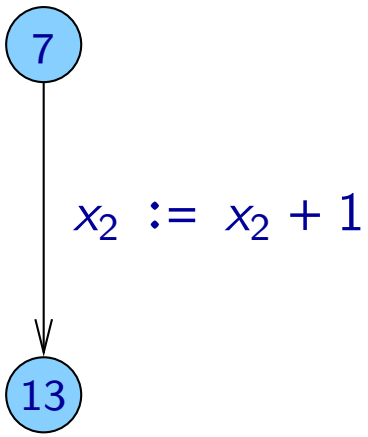
$$q = s$$

Aritmetika na přirozených číslech

Podobně není příliš komplikované vytvořit k danému Minského stroji $\mathcal{M}$ formuli, která bude charakterizovat, kdy je možné přejít z jedné konfigurace do druhé:

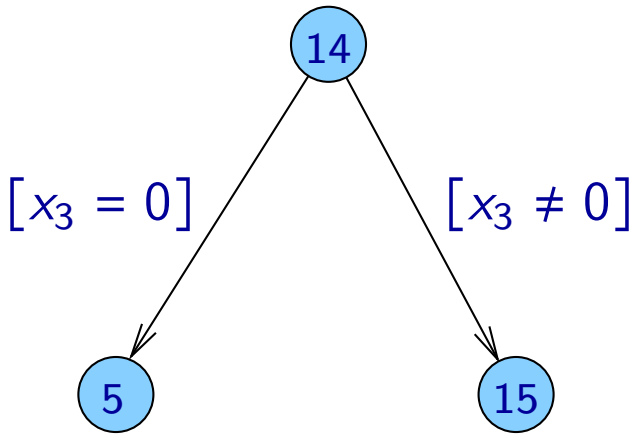
- $\textsc{Step}(q, v_1, v_2, v_3, q', v_1', v_2', v_3')$ — stroj $\mathcal{M}$ může přejít jedním krokem z konfigurace (q, v_1, v_2, v_3) do konfigurace (q', v_1', v_2', v_3')

  Bude se jednat o disjunkci mnoha formulí, kdy každá z těchto formulí bude popisovat činnost jedné instrukce stroje $\mathcal{M}$, např.



$$(q = 7) \wedge (q' = 13) \wedge$$
$$(v_1' = v_1) \wedge (v_2' = v_2 + 1) \wedge (v_3' = v_3)$$

Aritmetika na přirozených číslech



$$
\begin{aligned}
&(q = 14) \wedge \\
&(((v_3 = 0) \wedge (q' = 5)) \vee \\
&\quad ((v_3 > 0) \wedge (q' = 15))) \wedge \\
&(v_1' = v_1) \wedge (v_2' = v_2) \wedge (v_3' = v_3)
\end{aligned}
$$

Aritmetika na přirozených číslech

Výpočet stroje $\mathcal{M}$ můžeme popsat jako posloupnost konfigurací

$$\alpha_0, \alpha_1, \alpha_2, \ldots$$

Tuto posloupnost můžeme popsat jako několik samostatných posloupností:

- posloupnost stavů řídící jednotky
- posloupnost hodnot čítače x_1
- posloupnost hodnot čítače x_2

$$\vdots$$

- posloupnost hodnot čítače x_r

Obecně je možné jakoukoli konečnou posloupnost přirozených čísel kódovat jedním přirozeným číslem.

Pokud se tedy stroj $\mathcal{M}$ zastaví, tak můžeme každou z výše uvedených posloupností reprezentovat jako jedno přirozené číslo.

Aritmetika na přirozených číslech

Pokud máme například posloupnost přirozených čísel

$$a_0, a_1, \ldots, a_t$$

můžeme ji kódovat jako číslo

$$a_t \cdot b^t + a_{t-1} \cdot b^{t-1} + \cdots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0 \cdot b^0$$

kde b je nějaké dostatečně velké číslo, tj. takové číslo,
kde pro všechna a_i (kde $0 \leq i \leq t$) platí

$$0 \leq a_i < b$$

Posloupnost $a_0, a_1, \ldots, a_t$ tedy může být kódována jako jednotlivé číslice v zápisu čísla v číselné soustavě o základu b.

Aritmetika na přirozených číslech

- Pokud A je číslo kódující sekvenci $a_0, a_1, \ldots, a_t$ výše uvedeným způsobem, tak hodnotu a_i můžeme vyjádřit takto:

$$\exists u.\exists v.((A = (u \cdot b + a_i) \cdot b^i + v) \ \wedge \ (v < b^i))$$

  Zde je ale problém v tom, jak vyjádřit b^i.

- Jako základ b můžeme zvolit nějaké dostatečně velké prvočíslo p.

- Ve skutečnosti pro naše účely nepotřebujeme pracovat přímo s indexy i (kde $0 \leq i \leq t$).

  Místo toho postačí, pokud budeme pracovat s mocninami prvočísla p, tj. s hodnotami p^0, p^1, p^2, p^3, ...

  Místo i tedy budeme používat hodnotu p^i.

Aritmetika na přirozených číslech

Řekněme tedy, že p je prvočíslo, a že posloupnost $a_0, a_1, \ldots, a_t$ je kódována číslem

$$A = a_t \cdot p^t + a_{t-1} \cdot p^{t-1} + \cdots + a_2 \cdot p^2 + a_1 \cdot p^1 + a_0 \cdot p^0$$

(Předpokládáme, že pro každé i platí $0 \leq a_i < p$.)

- $\textsc{Digit}(p, d, A, c)$ — pro nějaké $i \in \mathbb{N}$ platí $d = p^i$ a v posloupnosti $a_0, a_1, \ldots, a_t$ kódované číslem A je $a_i = c$:

$$\textsc{Prime-Power}(p, d) \wedge (c < p) \wedge \exists u.\exists v.((A = (u \cdot p + c) \cdot d + v) \wedge (v < d))$$

Aritmetika na přirozených číslech

Výpočet stroje $\mathcal{M}$ tedy můžeme popsat pomocí následujících proměnných (pro konkrétnost předpokládáme, že počet čítačů je $r = 3$):

- p — dostatečně velké prvočíslo (větší než počet stavů s a větší než jakákoli hodnota kteréhokoli čítače během výpočtu)
- T — hodnota p^t, kde t je celkový počet kroků provedených strojem $\mathcal{M}$ během výpočtu
- Q — číslo kódující posloupnost stavů řídící jednotky
- X_1 — číslo kódující posloupnost hodnot čítače x_1
- X_2 — číslo kódující posloupnost hodnot čítače x_2
- X_3 — číslo kódující posloupnost hodnot čítače x_3

Aritmetika na přirozených číslech

- $\textsc{Conf}(p, d, Q, X_1, X_2, X_3, q, v_1, v_2, v_3)$:
  — konfigurace α_i, kde $d = p^i$, je rovna (q, v_1, v_2, v_3)

$$\textsc{Digit}(p, d, Q, q) \wedge \textsc{Digit}(p, d, X_1, v_1) \wedge \textsc{Digit}(p, d, X_2, v_2) \wedge \textsc{Digit}(p, d, X_3, v_3)$$

- $\textsc{Check-Initial}(p, Q, X_1, X_2, X_3)$:
  — kontrola toho, že daný výpočet začíná počáteční konfigurací

$$\exists q.\exists v_1.\exists v_2.\exists v_3.(\textsc{Conf}(p, 1, Q, X_1, X_2, X_3, q, v_1, v_2, v_3) \wedge \textsc{Initial-Conf}(q, v_1, v_2, v_3))$$

- $\textsc{Check-Final}(p, T, Q, X_1, X_2, X_3)$:
  — kontrola toho, že daný výpočet končí koncovou konfigurací

$$\exists q.\exists v_1.\exists v_2.\exists v_3.(\textsc{Conf}(p, T, Q, X_1, X_2, X_3, q, v_1, v_2, v_3) \wedge \textsc{Final-Conf}(q, v_1, v_2, v_3))$$

Aritmetika na přirozených číslech

- $\textsc{Check-One-Step}(p, d, Q, X_1, X_2, X_3)$:
  — kontrola toho, že v daném výpočtu stroj korektně přejde z konfigurace α_i do konfigurace α_{i+1}, kde $d = p^i$

$$\begin{aligned}
&\exists q.\exists v_1.\exists v_2.\exists v_3.\exists q'.\exists v_1'.\exists v_2'.\exists v_3'.(\\
&\qquad \textsc{Conf}(p, d, Q, X_1, X_2, X_3, q, v_1, v_2, v_3) \wedge \\
&\qquad \textsc{Conf}(p, d \cdot p, Q, X_1, X_2, X_3, q', v_1', v_2', v_3') \wedge \\
&\qquad \textsc{Step}(q, v_1, v_2, v_3, q', v_1', v_2', v_3'))
\end{aligned}$$

- $\textsc{Check-All-Steps}(p, T, Q, X_1, X_2, X_3)$:
  — kontrola toho, že všechny kroky jsou v pořádku

$$\begin{aligned}
&\forall d.((d < T) \wedge \textsc{Prime-Power}(p, d) \rightarrow \\
&\qquad \textsc{Check-One-Step}(p, d, Q, X_1, X_2, X_3))
\end{aligned}$$

Aritmetika na přirozených číslech

- MACHINE-HALTS:
  — kontrola toho, že existuje konečný výpočet daného stroje

$$\exists p.\exists T.\exists Q.\exists X_1.\exists X_2.\exists X_3.(\\ \text{PRIME}(p) \wedge \\ \text{PRIME-POWER}(p, T) \wedge \\ \text{CHECK-INITIAL}(p, Q, X_1, X_2, X_3) \wedge \\ \text{CHECK-ALL-STEPS}(p, T, Q, X_1, X_2, X_3) \wedge \\ \text{CHECK-FINAL}(p, T, Q, X_1, X_2, X_3))$$

Není těžké ověřit, že tato formule je pravdivá právě tehdy, pokud se výpočet stroje $\mathcal{M}$ zastaví po nějakém konečném počtu kroků.

Aritmetika na přirozených číslech

Pokud by tedy existoval algoritmus, který by pro každou takovou formuli umožňoval zjistit, zda je pravdivá, dostali bychom algoritmus řešící Halting problem. To ale není možné.

Poznámky:

- Je zajímavé, že analogický problém, kde ale místo přirozených čísel uvažujeme čísla reálná, je algoritmicky rozhodnutelný (i když popis daného algoritmu a důkaz jeho korektnosti jsou značně netriviální).

- Rovněž pokud uvažujeme přirozená nebo celá čísla a stejné formule jako v předchozím případě, ale s tím rozdílem, že v nich nesmí být použit funkční symbol $\cdot$ (násobení), tak je problém algoritmicky rozhodnutelný.

Aritmetika na přirozených číslech

Pokud můžeme používat $\cdot$, je ve skutečnosti nerozhodnutelný už velmi omezený případ:

Desátý Hilbertův problém

Vstup: Polynom $f(x_1, x_2, \ldots, x_n)$ vytvořený z proměnných $x_1, x_2, \ldots, x_n$ a celočíselných konstant.

Otázka: Existují přirozená čísla $x_1, x_2, \ldots, x_n$ taková, že $f(x_1, x_2, \ldots, x_n) = 0$?

Příklad vstupu: $5x^2y - 8yz + 3z^2 - 15$

Tj. ptáme se, zda

$$\exists x \exists y \exists z (5 \cdot x \cdot x \cdot y + (-8) \cdot y \cdot z + 3 \cdot z \cdot z + (-15) = 0)$$

platí v oboru přirozených čísel.

Další nerozhodnutelné problémy

Také následující problém je algoritmicky nerozhodnutelný:

Problém

Vstup: Uzavřená formule φ predikátové logiky prvního řádu.

Otázka: Platí $\models \varphi$?

Poznámka: Zápis $\models \varphi$ znamená, že formule φ je logicky platná, tj. pravdivá v každé interpretaci.

Další nerozhodnutelné problémy

Redukcí z Halting problému se dá ukázat nerozhodnutelnost celé řady problémů, které se týkají ověřování chování programů:

- Vydá daný program pro nějaký vstup odpověď ANO?
- Zastaví se daný program pro libovolný vstup?
- Dávají dva dané programy pro stejné vstupy stejný výstup?
- ...

Riceova věta

Řekněme, že P je nějaká vlastnost Turingových strojů.

Vlastnost P je:

- **netriviální** — pokud existuje alespoň jeden stroj, který vlastnost P má, a alespoň jeden stroj, který vlastnost P nemá
- **vstupně-výstupní** — pokud každé dva stroje, které se zastaví pro stejné vstupy, a pro stejné vstupy dávají stejné výstupy, vždy oba vlastnost P mají nebo oba nemají

Věta

Každý problém tvaru

Vstup: Turingův stroj $\mathcal{M}$.

Otázka: Má stroj $\mathcal{M}$ vlastnost P?

kde P je netriviální vstupně-výstupní vlastnost, je nerozhodnutelný.

Riceova věta

Důkaz:

- Důkaz bude proveden redukcí z Halting problému.
- Nejedná o jednu konkrétní redukci, ale o obecné **schéma** popisující, jak pro **každou** konkrétní netriviální vstupně-výstupní vlastnost P vytvořit příslušnou redukci Halting problému na jeden z následujících dvou problémů:
  - Otázku zda daný Turingův stroj danou vlastnost P má.
  - Otázku zda daný Turingův stroj danou vlastnost P nemá.

Riceova věta

Algoritmus provádějící tuto redukci:

- Dostane jako svůj vstup instanci Halting problému $(\mathcal{M}, w)$ (kde $\mathcal{M}$ je Turingův stroj a w jeho vstup).
- K dané dvojici vyrobí Turingův stroj $\mathcal{M}$'.

Pro danou redukci bude vždy platit jedna z následujících dvou možností:

- Stroj $\mathcal{M}'$ bude mít vlastnost P právě tehdy, když se stroj $\mathcal{M}$ nad vstupem w zastaví.
- Stroj $\mathcal{M}'$ bude mít vlastnost P právě tehdy, když se stroj $\mathcal{M}$ nad vstupem w nezastaví.

Která možnost to bude, závisí na dané vlastnosti P.

Riceova věta

Označme $\mathcal{M}_0$ Turingův stroj, který se pro žádný vstup nikdy nezastaví a nikdy nevygeneruje žádný výstup, tj. pro každý vstup vždy jen běží do nekonečna.

Mohou nastat dvě možnosti:

- Stroj $\mathcal{M}_0$ vlastnost P má.
- Stroj $\mathcal{M}_0$ vlastnost P nemá.

Budeme se soustředit na druhou možnost, tj. když $\mathcal{M}_0$ vlastnost P nemá. (Důkaz pro první možnost, tj. když $\mathcal{M}_0$ vlastnost P má, bude podobný.)

Protože je vlastnost P netriviální, musí existovat nějaký alespoň jeden stroj $\mathcal{M}_1$, který tuto vlastnost má.

Riceova věta

Algoritmus provádějící redukci k dané instanci Halting problému $(\mathcal{M}, w)$, kterou dostane jako vstup, vyrobí Turingův stroj $\mathcal{M}'$, který se bude chovat následovně:

- Nechá si na pásce uložený svůj vstup w' a zbylou „prázdnou“ část pásky použije pro simulaci činnosti stroje $\mathcal{M}$ na vstupu w.
- V okamžiku, kdy tato simulace výpočtu stroje $\mathcal{M}$ na vstupu w skončí, tak:
  - smaže všechna políčka pásky použitá při této simuluci (tj. přepíše je symboly blank),
  - zajede hlavou na začátek slova w',
  - začne se chovat jako stroj $\mathcal{M}_1$.

Riceova věta

Je zjevné, že:

- Pokud se výpočet stroje $\mathcal{M}$ na vstupu w zastaví:

  Stroj $\mathcal{M}'$ se z hlediska vstupu a výstupu bude chovat naprosto stejně jako stroj $\mathcal{M}_1$.
  Stroj $\mathcal{M}'$ tedy bude mít vlastnost P
  (protože je to vstupně-výstupní vlastnost a stroj $\mathcal{M}_1$ tuto vlastnost má).

- Pokud se výpočet stroje $\mathcal{M}$ na vstupu w nezastaví:

  Simulace činnosti stroje $\mathcal{M}$ na vstupu w prováděná strojem $\mathcal{M}'$ se nikdy nezastaví.
  Stroj $\mathcal{M}'$ se tedy z hlediska vstupu a výstupu bude chovat naprosto stejně jako stroj $\mathcal{M}_0$.
  Stroj $\mathcal{M}'$ proto nebude mít vlastnost P
  (protože je to vstupně-výstupní vlastnost a stroj $\mathcal{M}_0$ tuto vlastnost nemá).

Riceova věta

Případ, kdy stroj $\mathcal{M}_0$ má vlastnost P, bude podobný:

- Jako $\mathcal{M}_1$ zvolíme stroj, který vlastnost P nemá.
- Pokud se $\mathcal{M}$ na w zastaví, bude se $\mathcal{M}'$ chovat stejně jako $\mathcal{M}_1$ a nebude tedy mít vlastnost P.
- Pokud se $\mathcal{M}$ na w nezastaví, bude se $\mathcal{M}'$ chovat stejně jako $\mathcal{M}_0$ a bude tedy mít vlastnost P.