Vzájemná simulace mezi různými druhy strojů

Simulace výpočtu

Vysvětlení toho, co to znamená, že stroj $\mathcal{M}$ je **simulován** strojem $\mathcal{M}'$:

- Výpočet stroje $\mathcal{M}$ pro vstup w je (konečná nebo nekonečná) posloupnost konfigurací stroje $\mathcal{M}$

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

- Tomuto výpočtu odpovídá výpočet stroje $\mathcal{M}'$ tvořený konfiguracemi

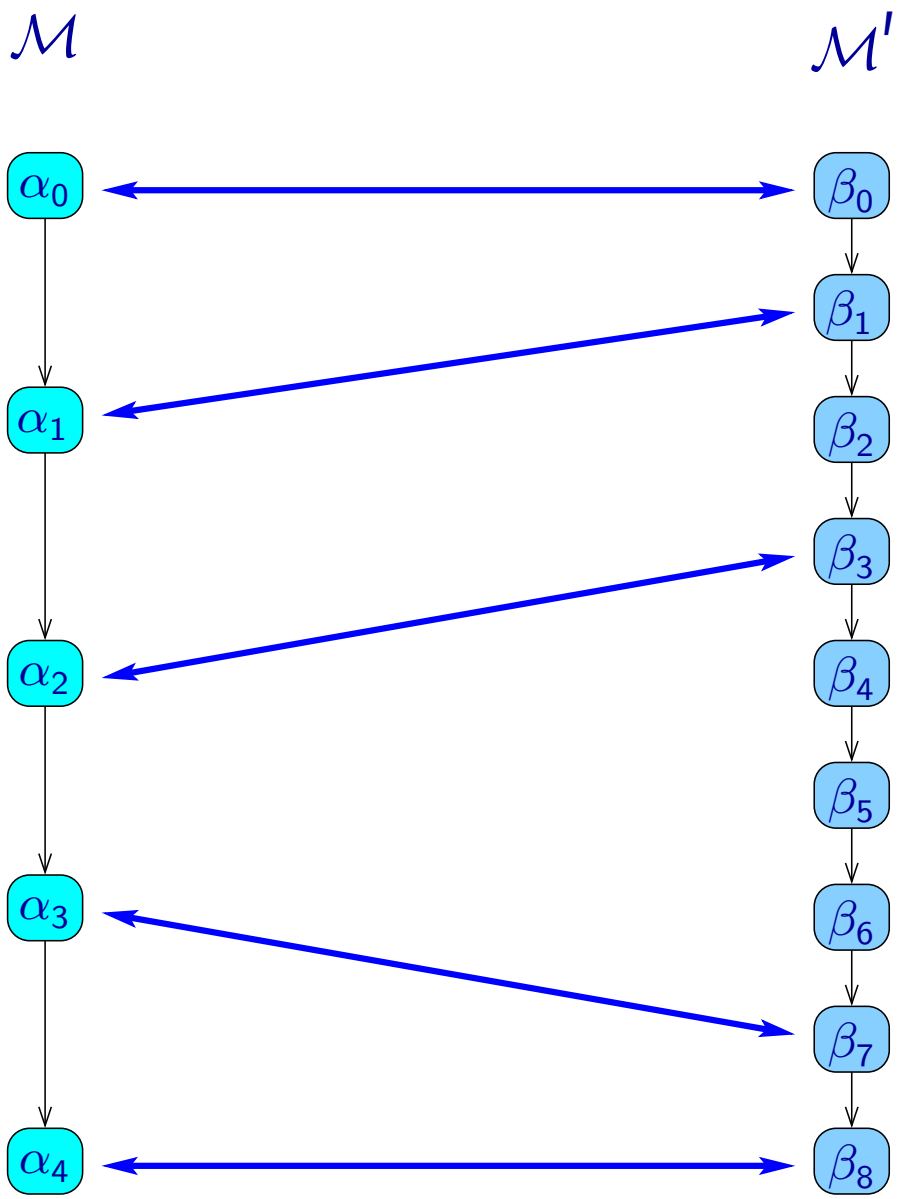
$$\beta_0 \longrightarrow \beta_1 \longrightarrow \beta_2 \longrightarrow \cdots$$

  kde každé konfiguraci α_i odpovídá nějaká konfigurace $\beta_{f(i)}$, kde $f : \mathbb{N} \to \mathbb{N}$ je funkce, pro kterou platí $f(i) \leq f(j)$ pro každé i a j, kde $i < j$.

- Existuje relace mezi vzájemně si odpovídajícími konfiguracemi stroje $\mathcal{M}$ a jim odpovídajícími konfiguracemi stroje $\mathcal{M}'$.

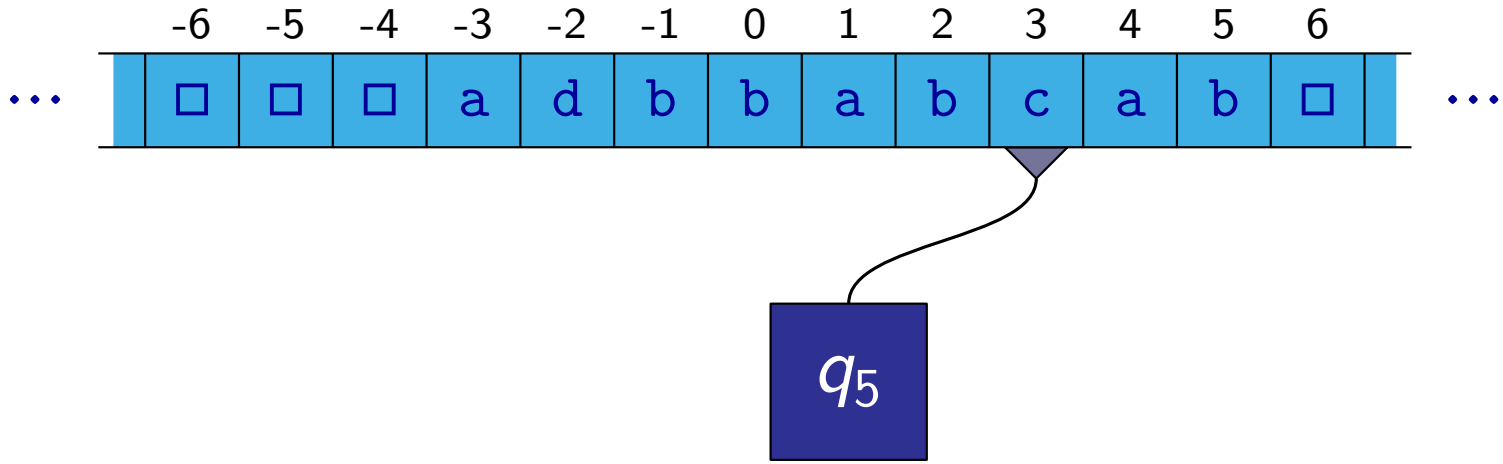
- Existují funkce mapující vstup w na odpovídající počáteční konfigurace α_0 a β_0 a analogicky funkce mapující koncové konfigurace na výsledek výpočtu.

Simulace výpočtu

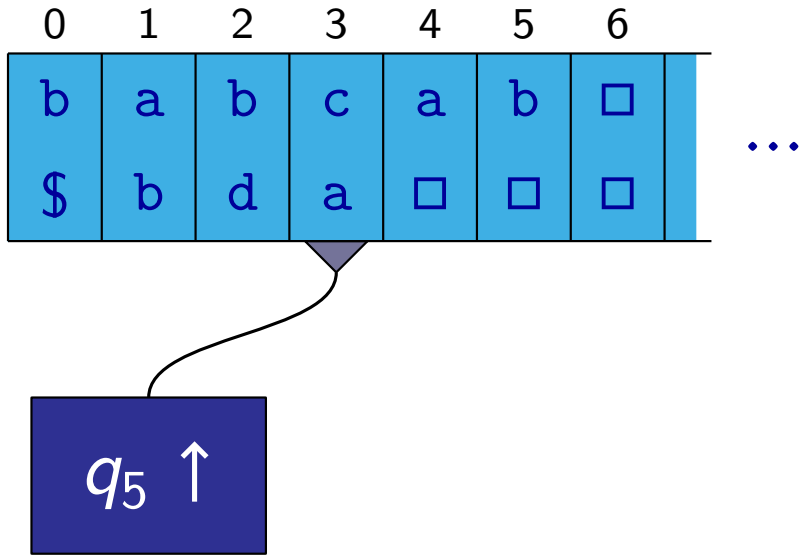


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

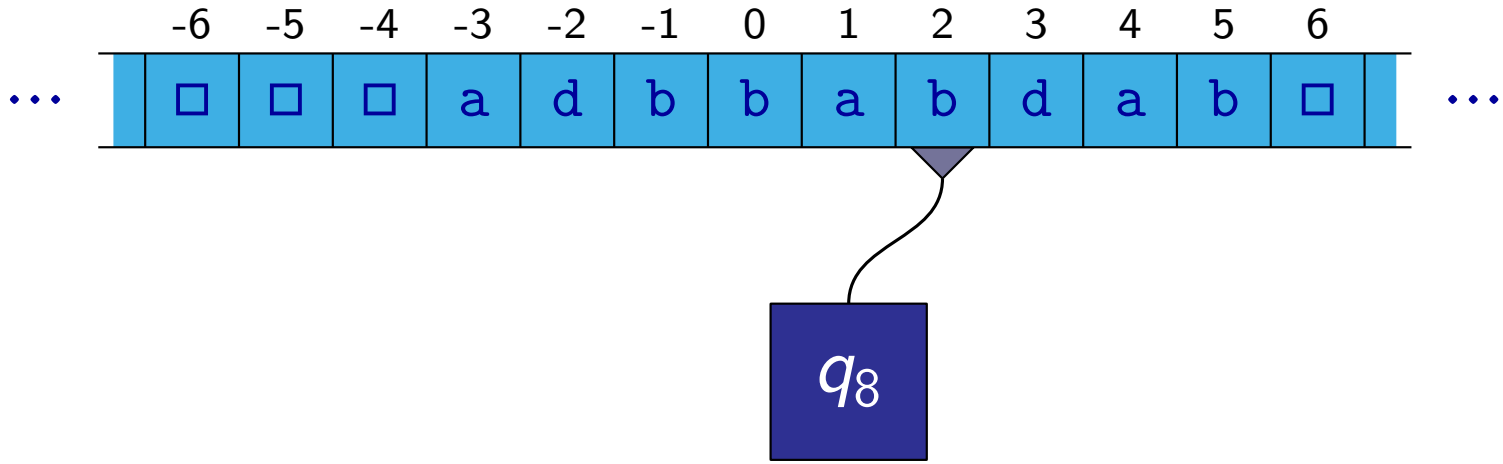


Jednostranně nekonečná páska:

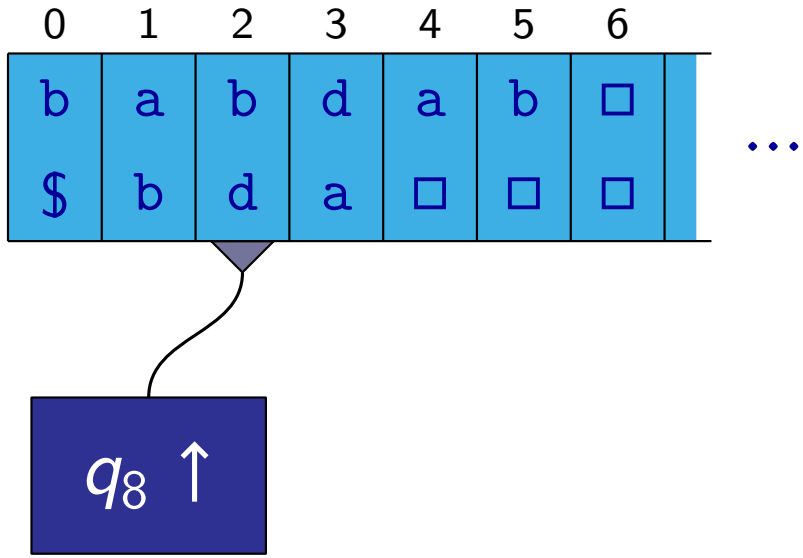


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

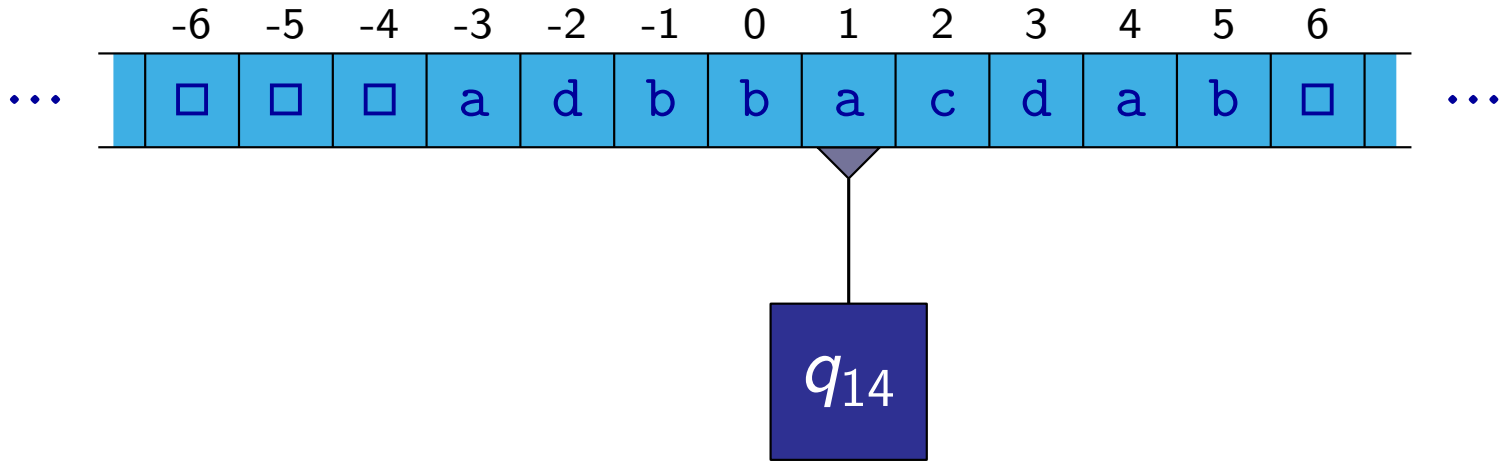


Jednostranně nekonečná páska:

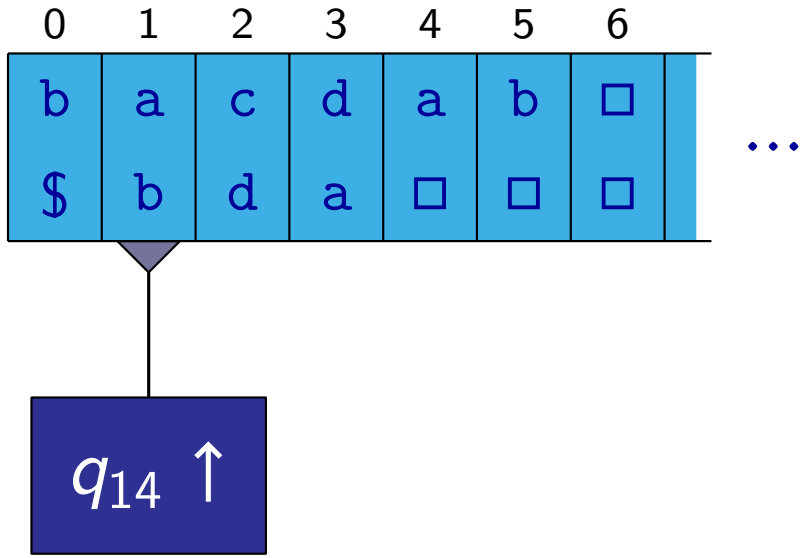


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

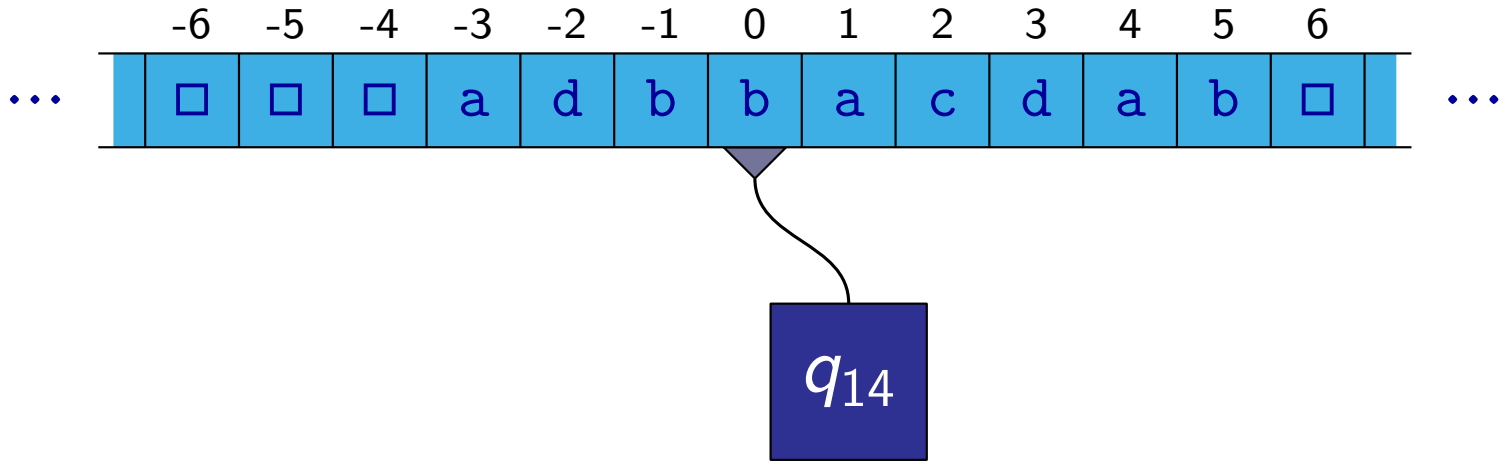


Jednostranně nekonečná páska:

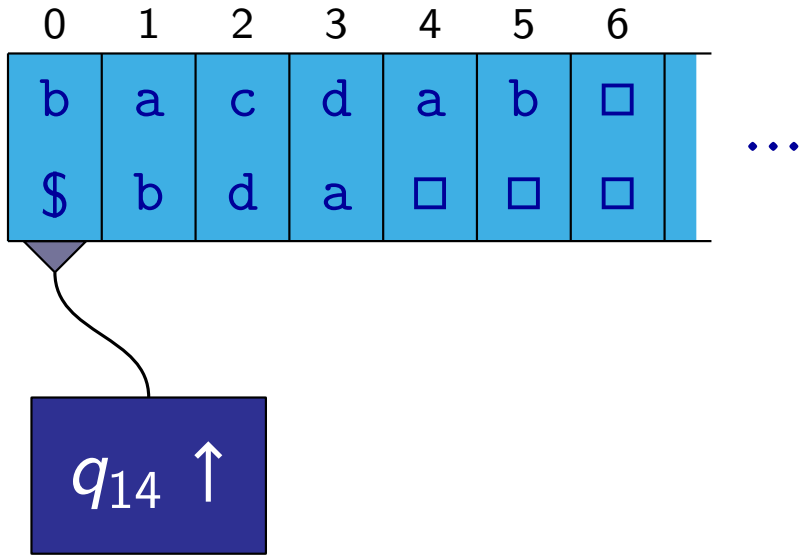


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

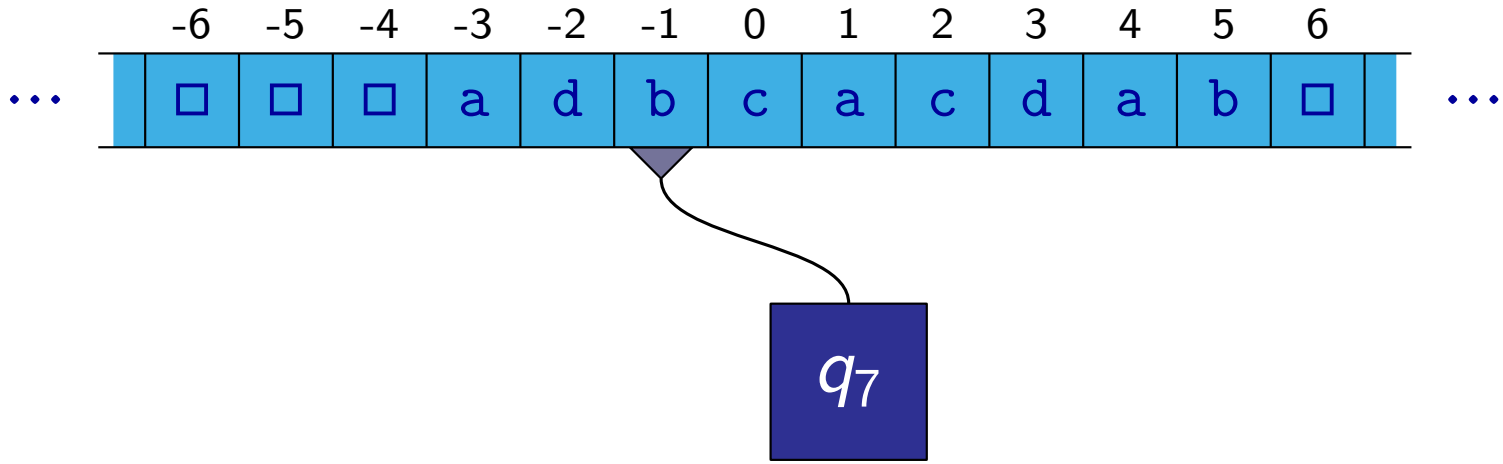


Jednostranně nekonečná páska:

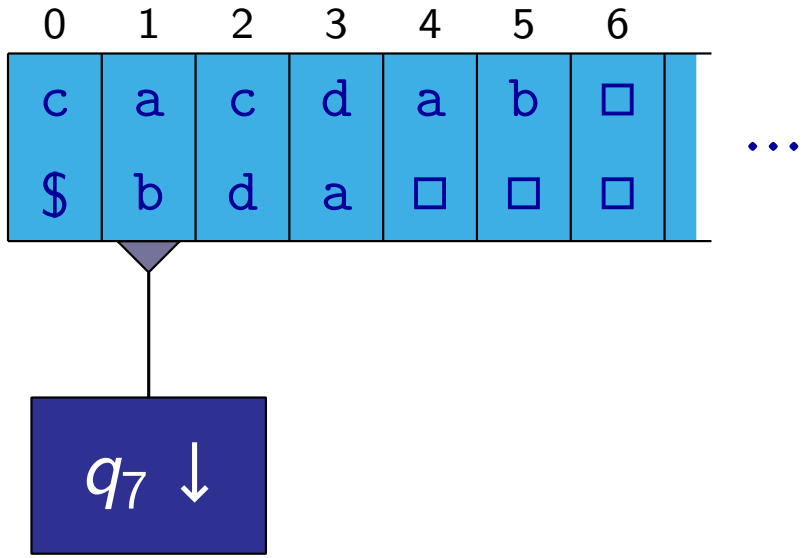


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

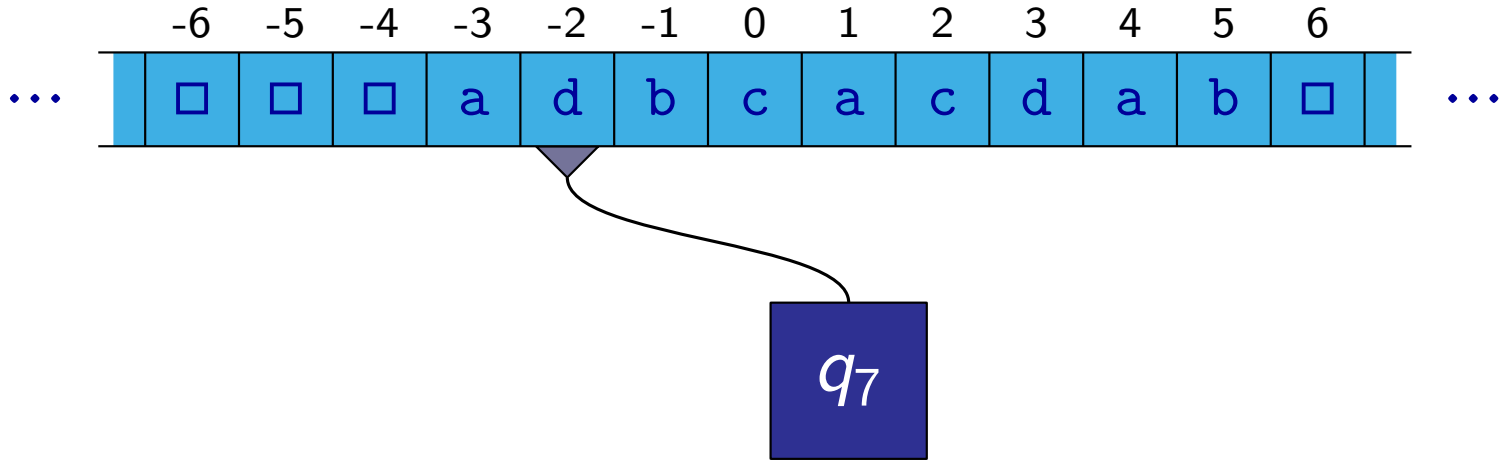


Jednostranně nekonečná páska:

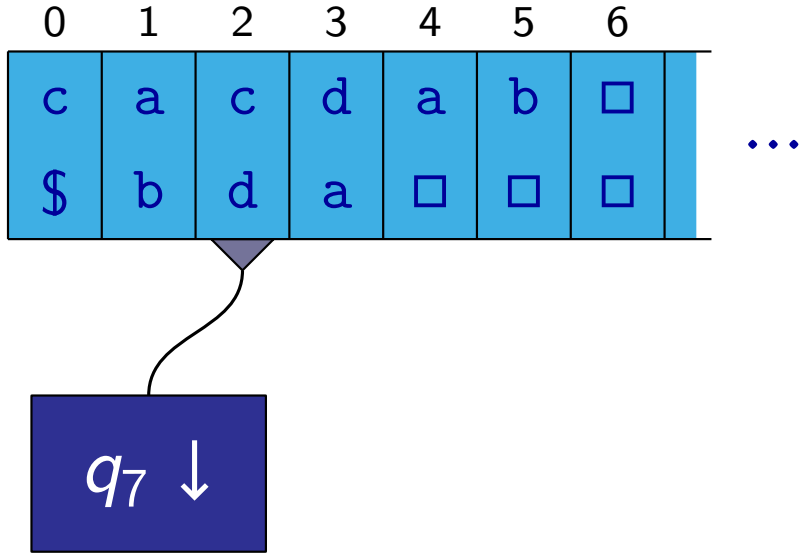


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

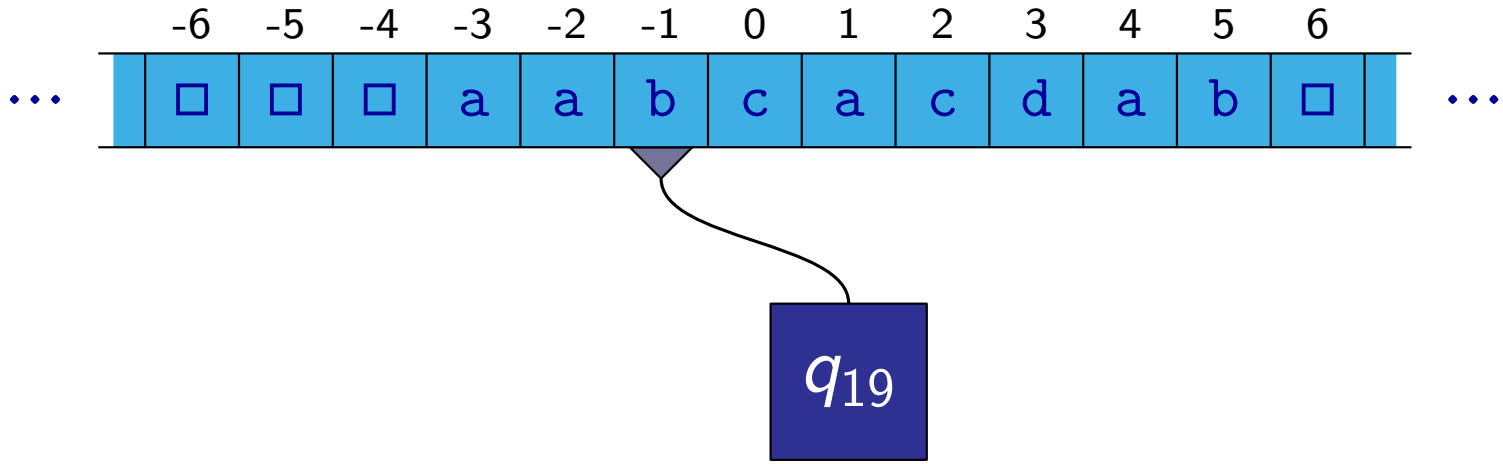


Jednostranně nekonečná páska:

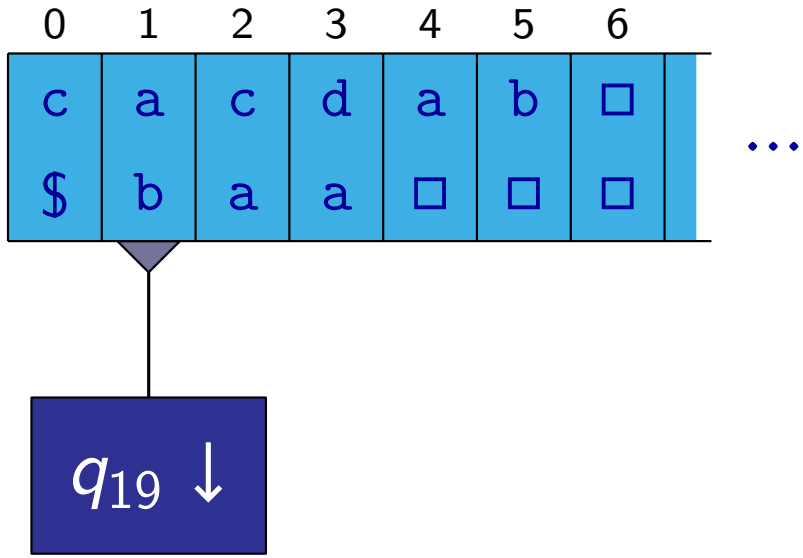


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

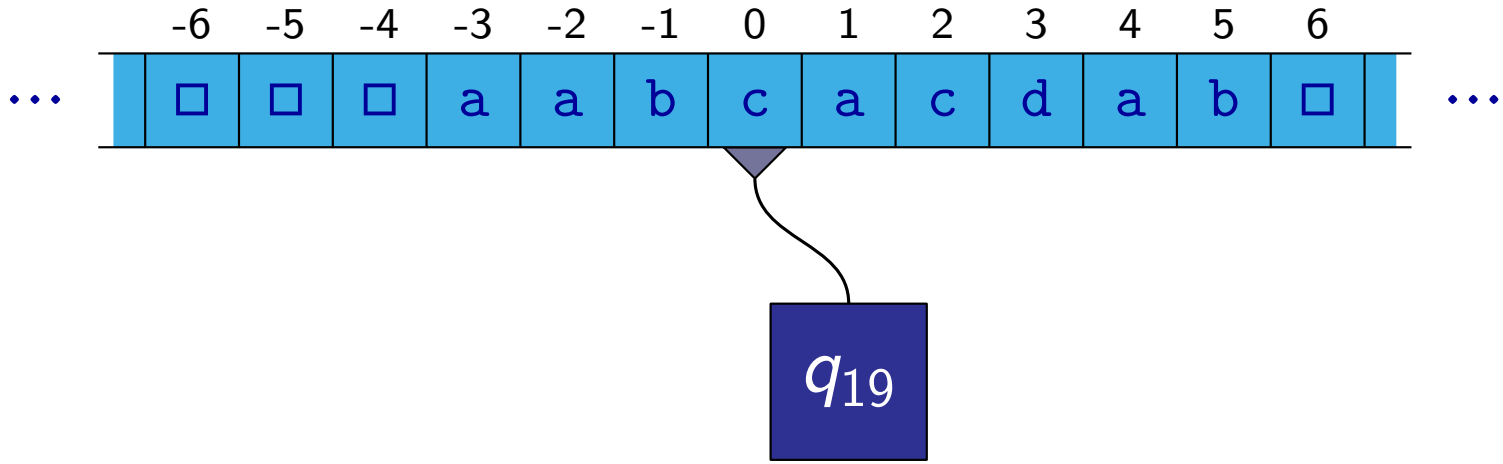


Jednostranně nekonečná páska:

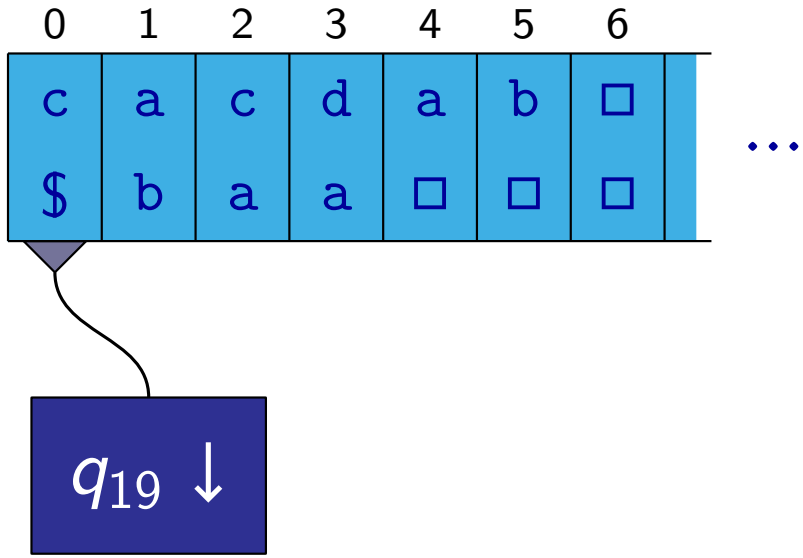


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

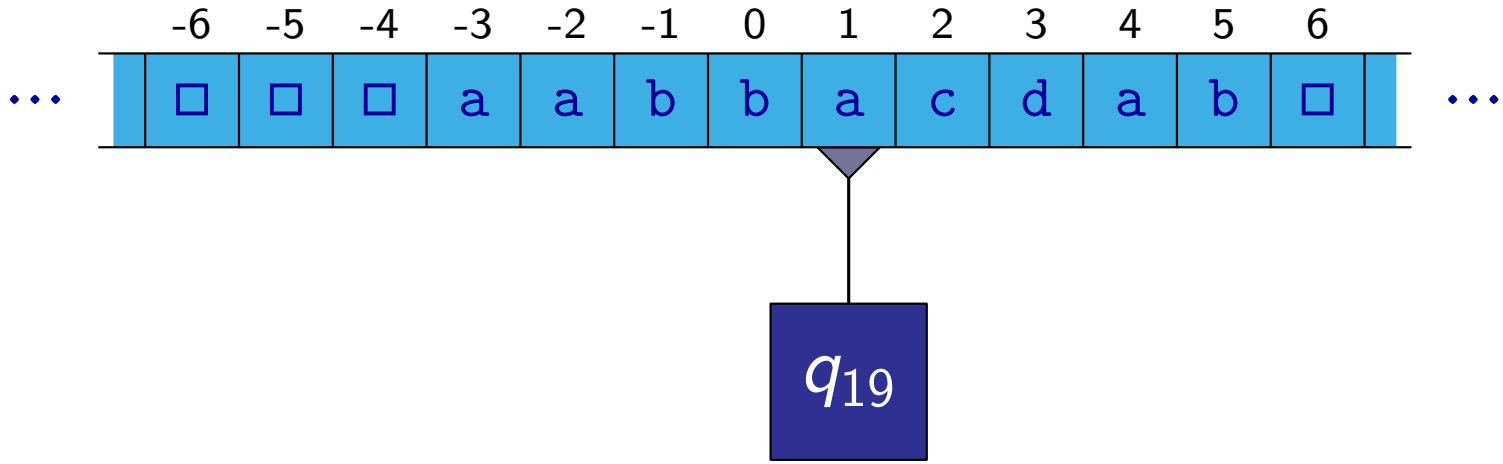


Jednostranně nekonečná páska:

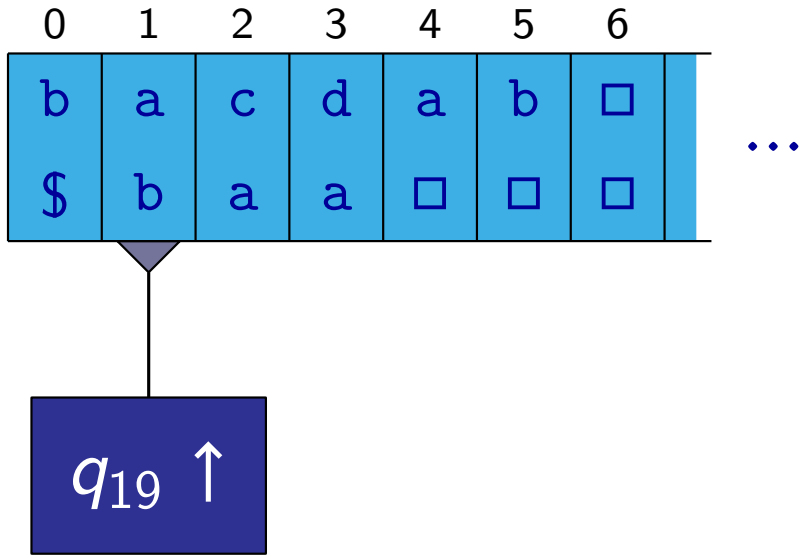


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

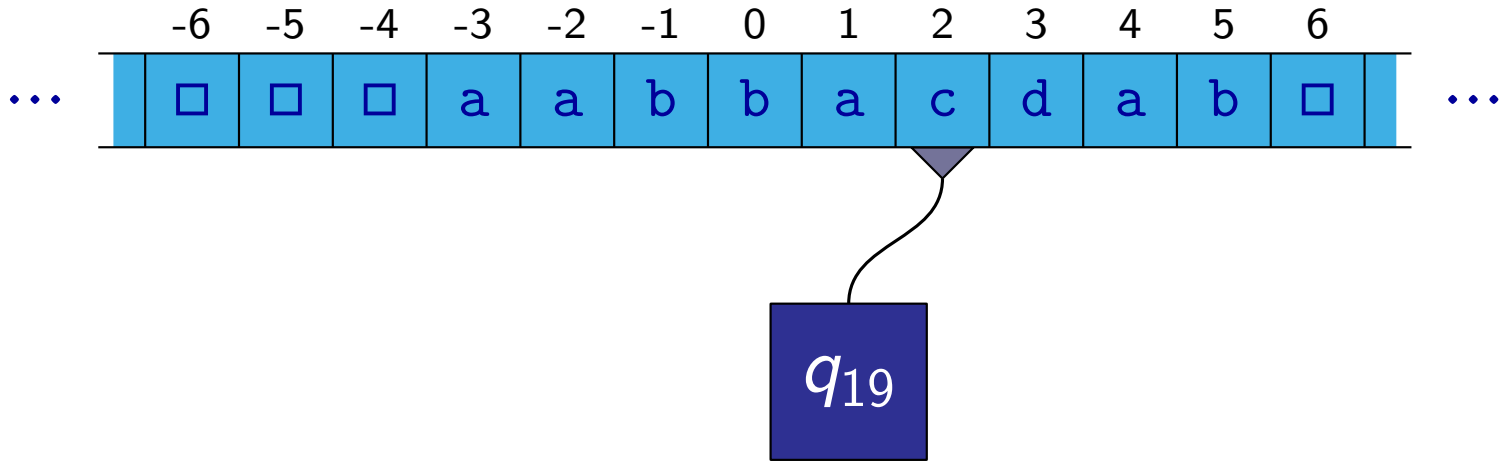


Jednostranně nekonečná páska:

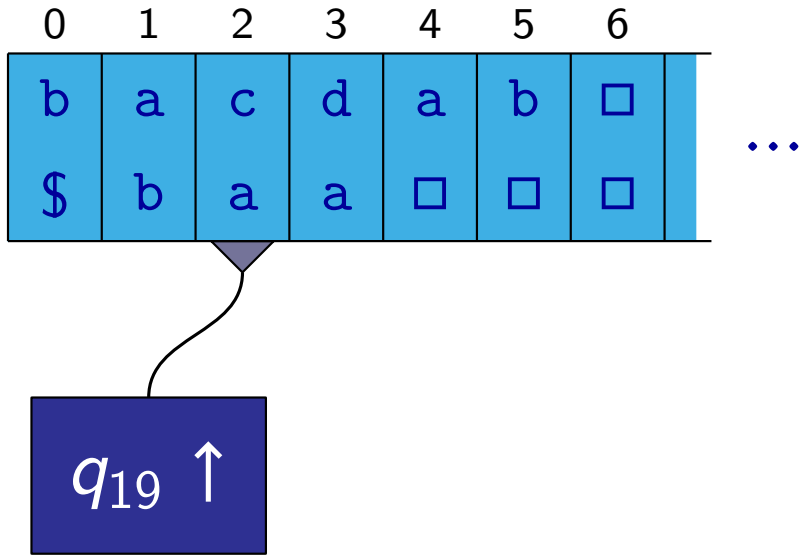


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:



Jednostranně nekonečná páska:



Abeceda $\{0, 1\}$

Činnost stroje s libovolnou páskovou abecedou Γ může být simulována strojem s páskovou abecedou $\{0, 1\}$.

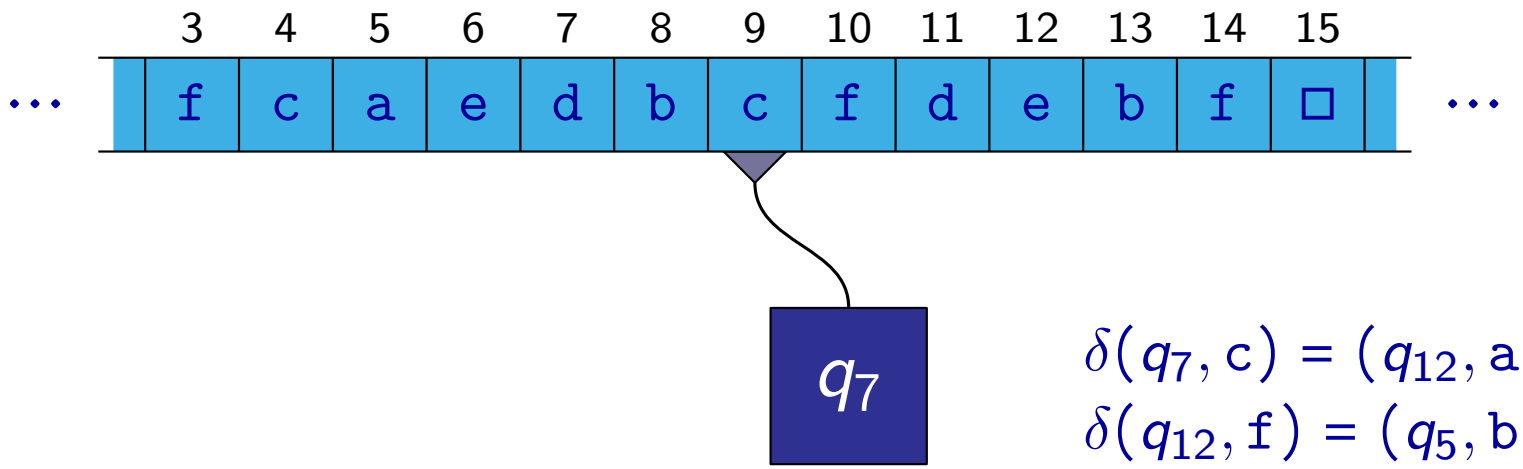
Stačí zvolit nějaké vhodné kódování symbolů abecedy Γ pomocí k-bitových sekvencí.

Příklad: Pásková abeceda $\Gamma = \{\square, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{array}{ccc}
\square & \leftrightarrow & 000 \\
\mathtt{a} & \leftrightarrow & 001 \\
\mathtt{b} & \leftrightarrow & 010 \\
\mathtt{c} & \leftrightarrow & 011 \\
\mathtt{d} & \leftrightarrow & 100 \\
\mathtt{e} & \leftrightarrow & 101 \\
\mathtt{f} & \leftrightarrow & 110 \\
\mathtt{g} & \leftrightarrow & 111
\end{array}
$$

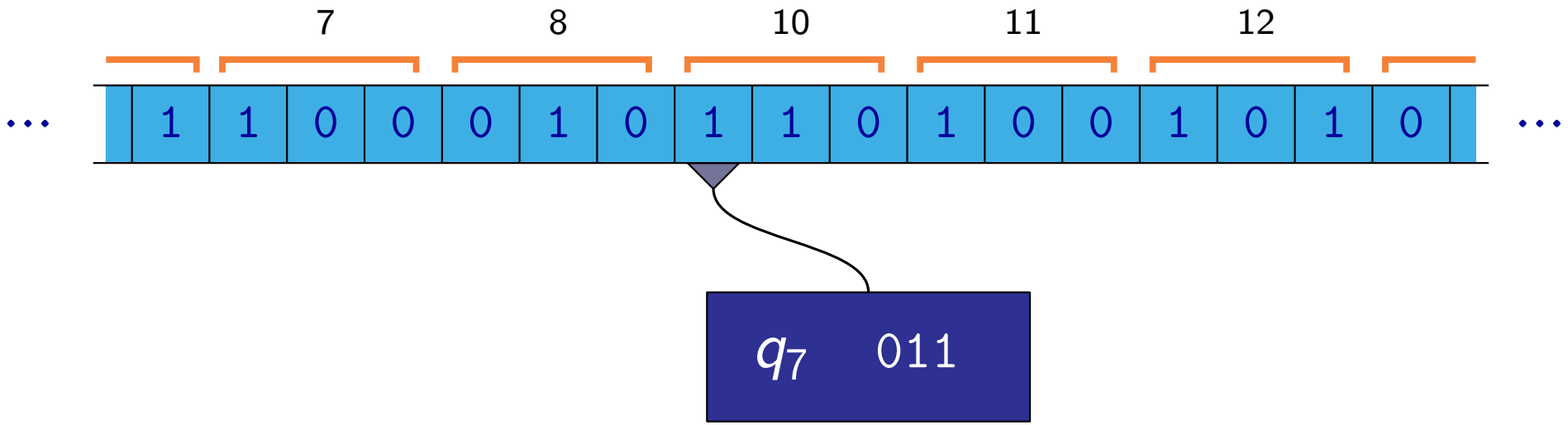
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



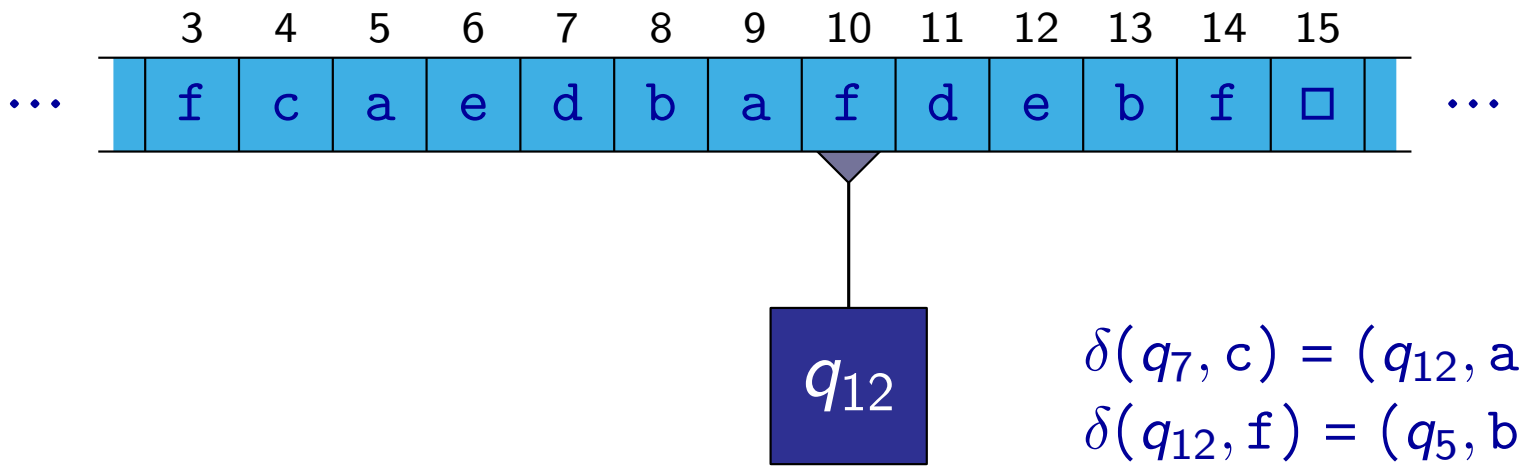
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



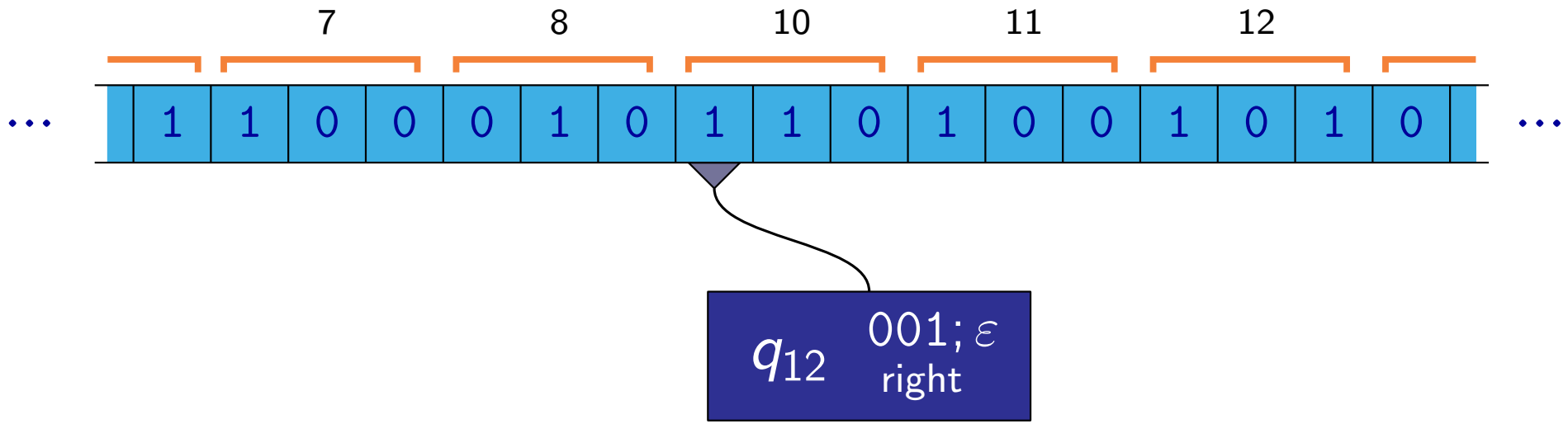
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



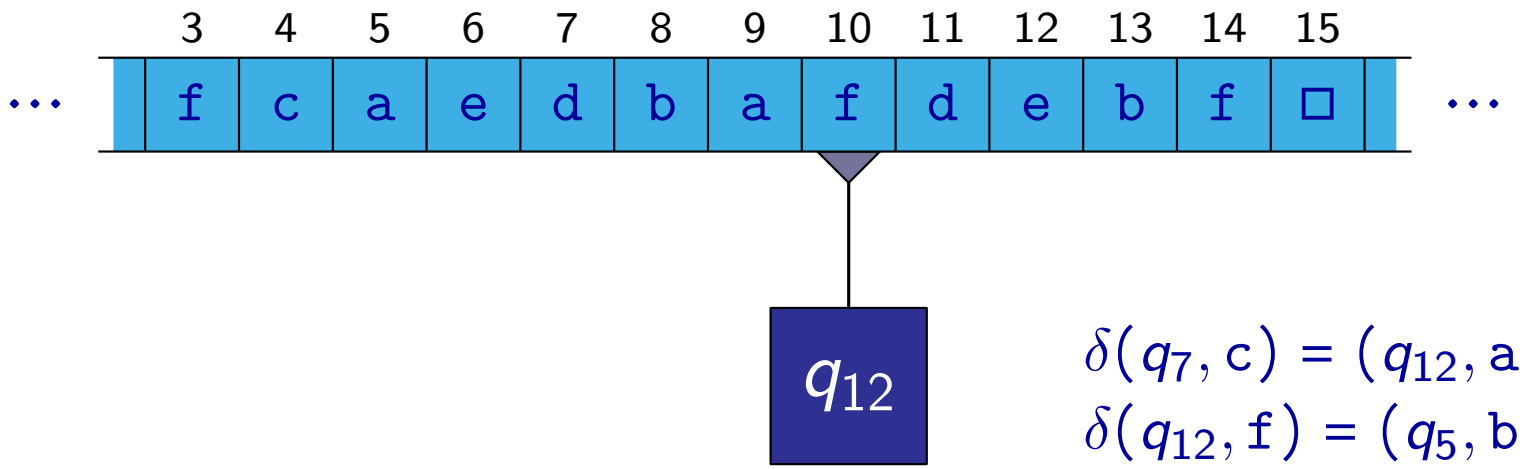
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



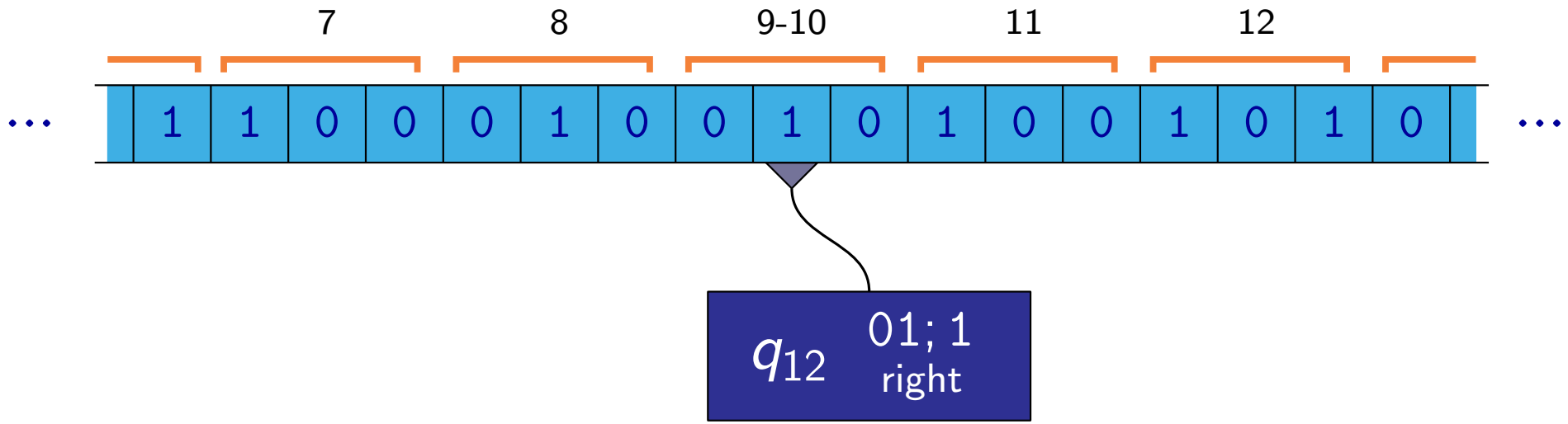
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



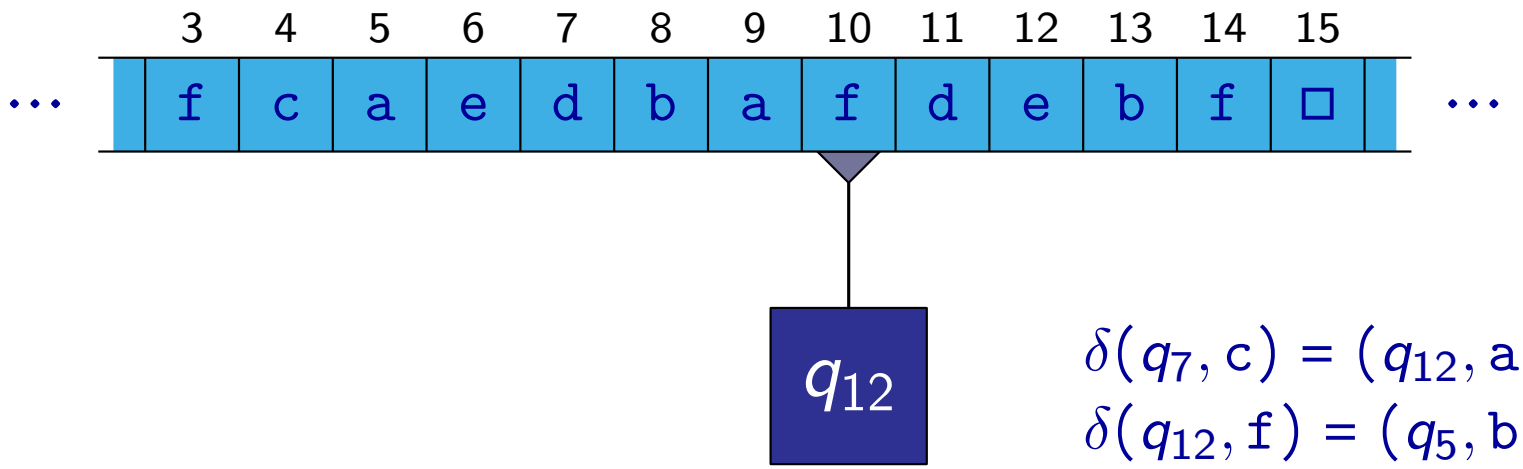
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



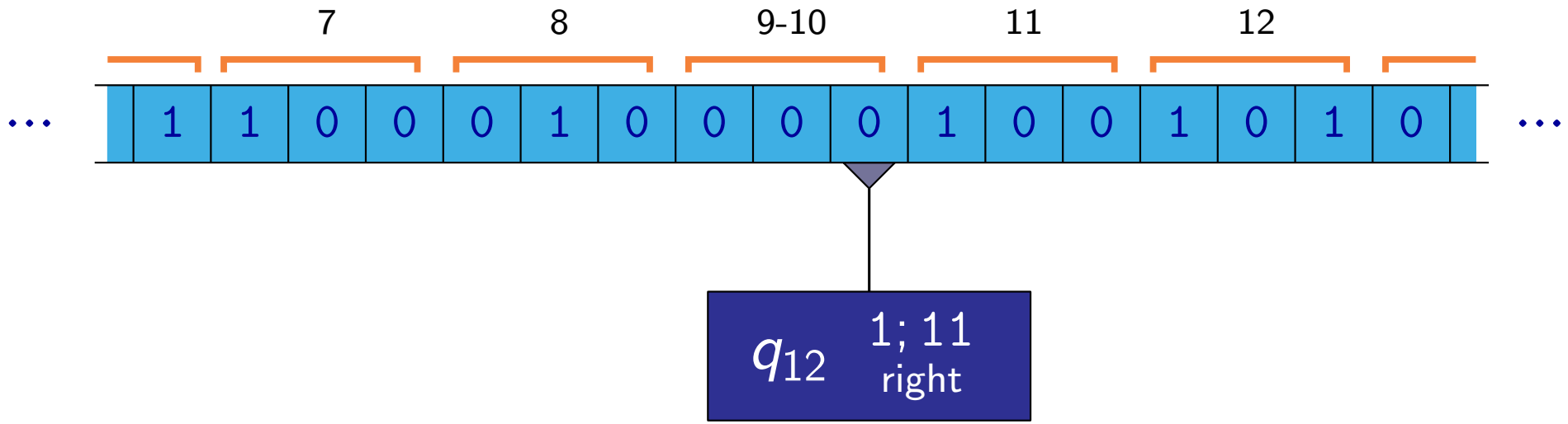
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



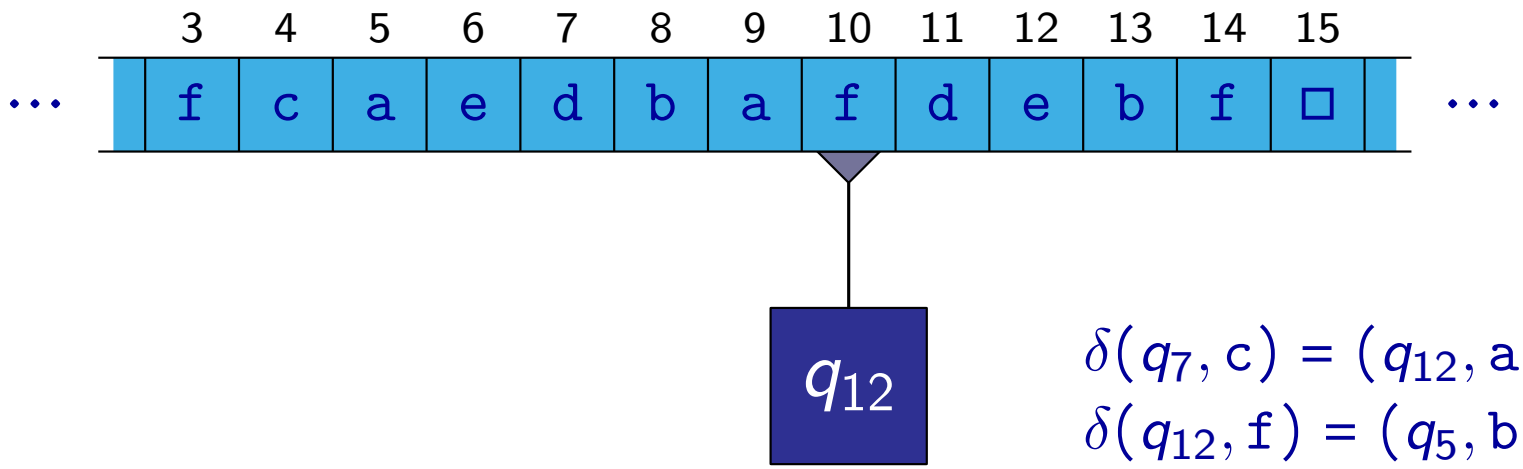
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



Abeceda $\{0, 1\}$

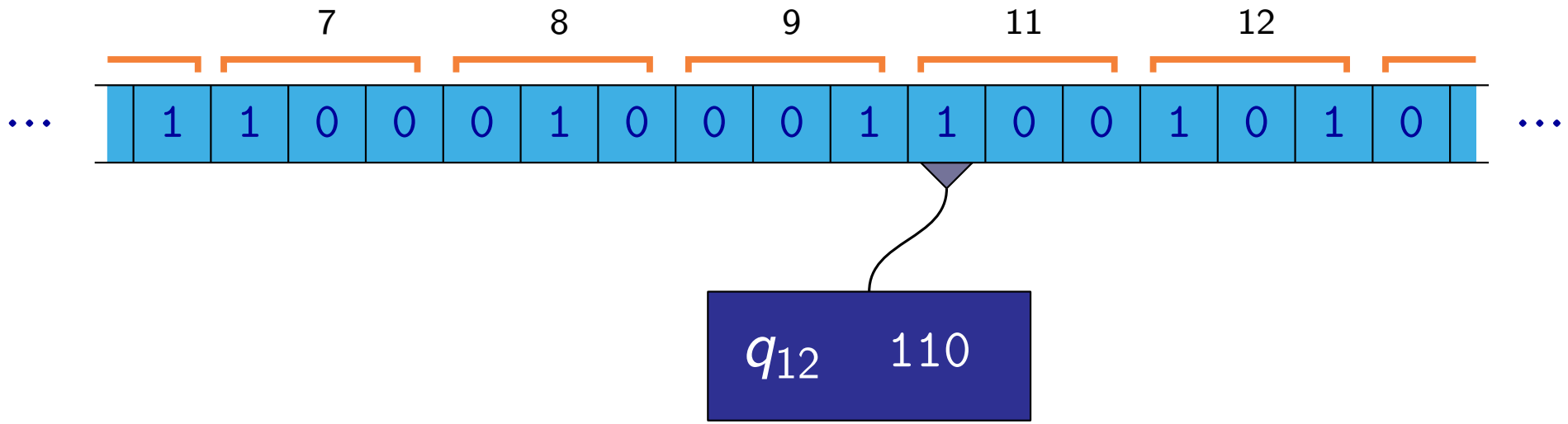
Stroj s páskovou abecedou Γ:



$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$

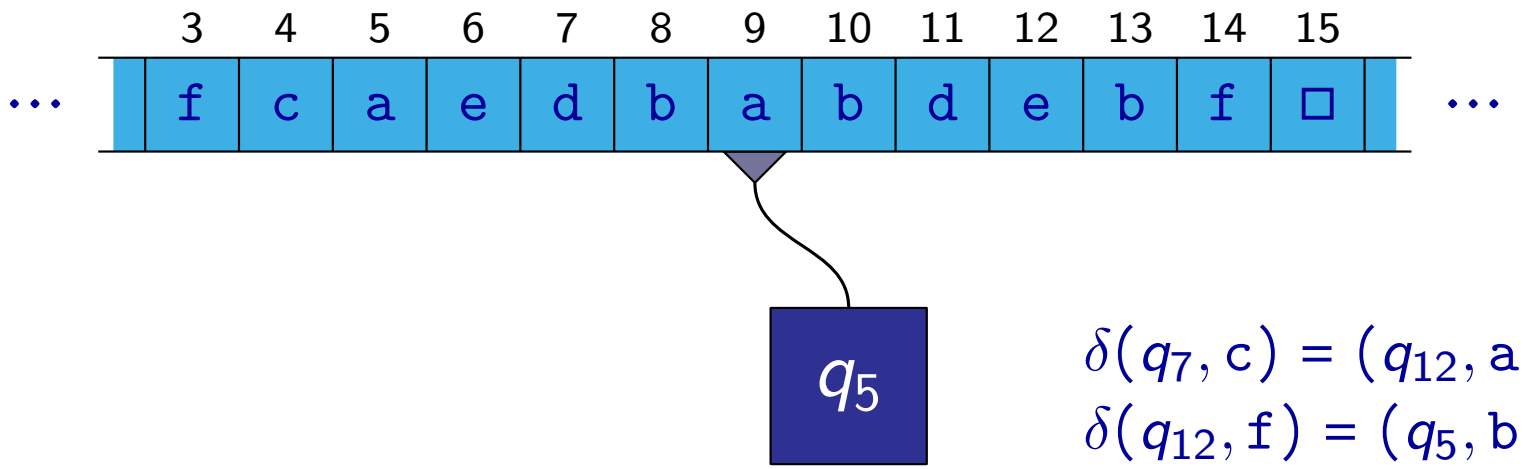
$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$

Stroj s abecedou $\{0, 1\}$:



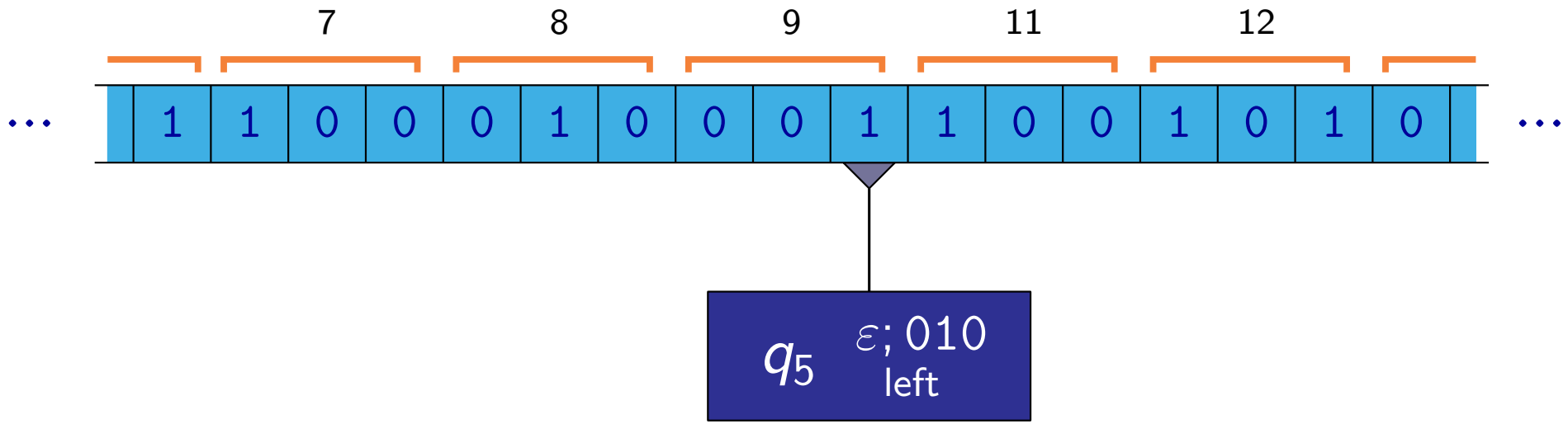
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



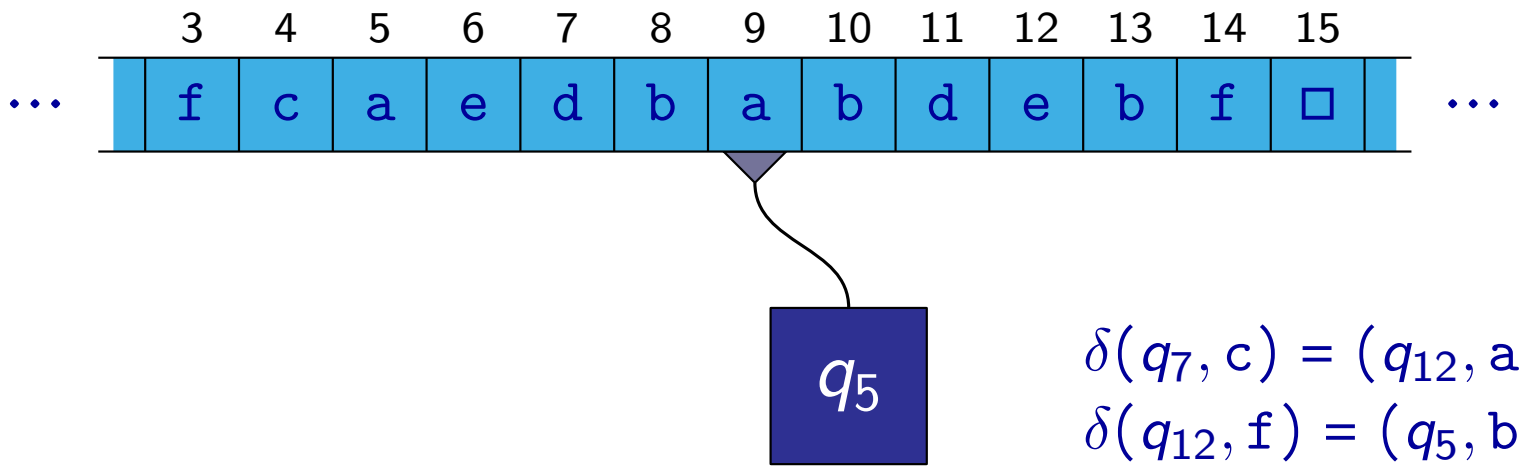
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



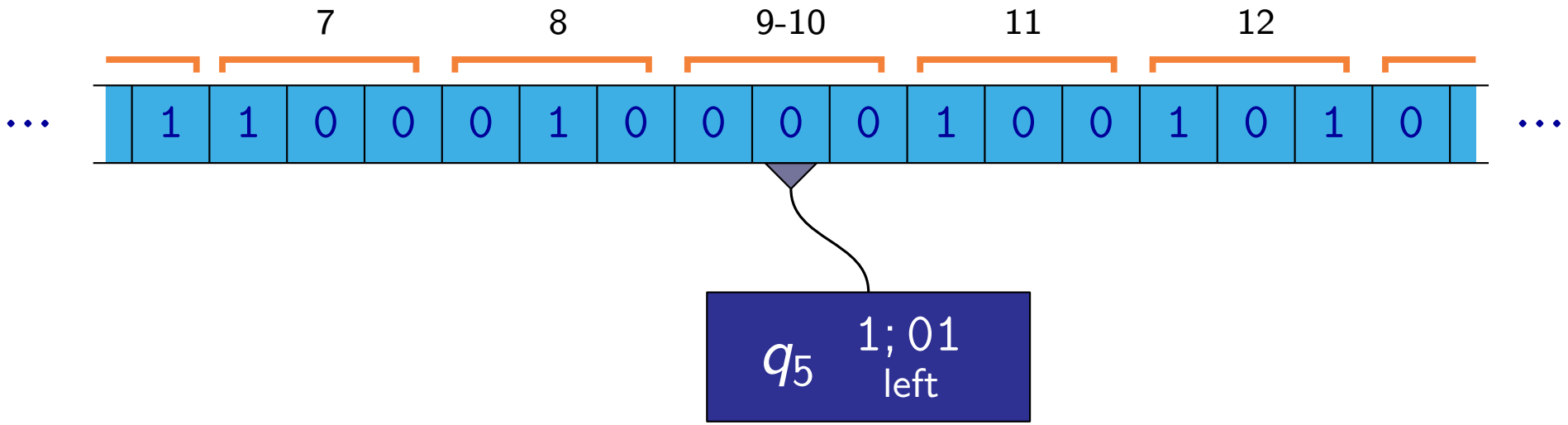
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



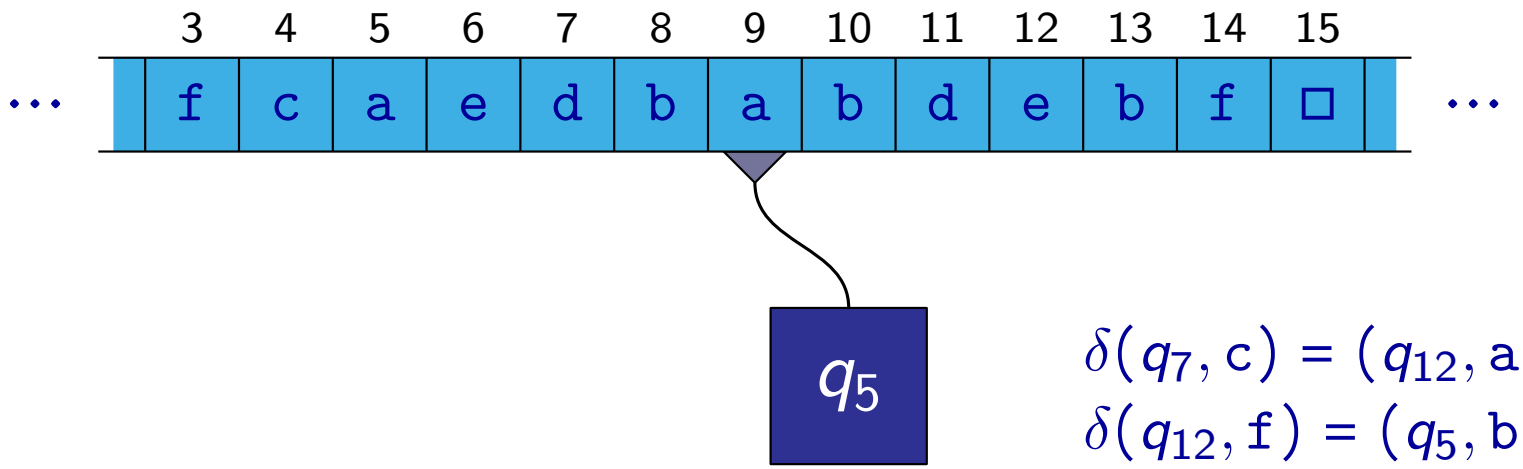
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



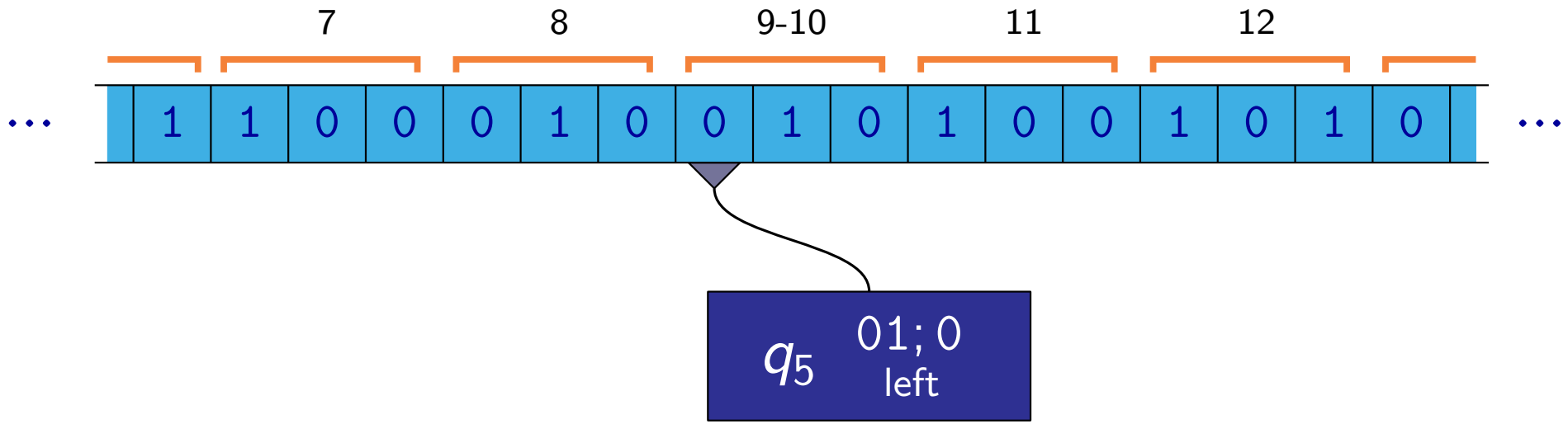
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



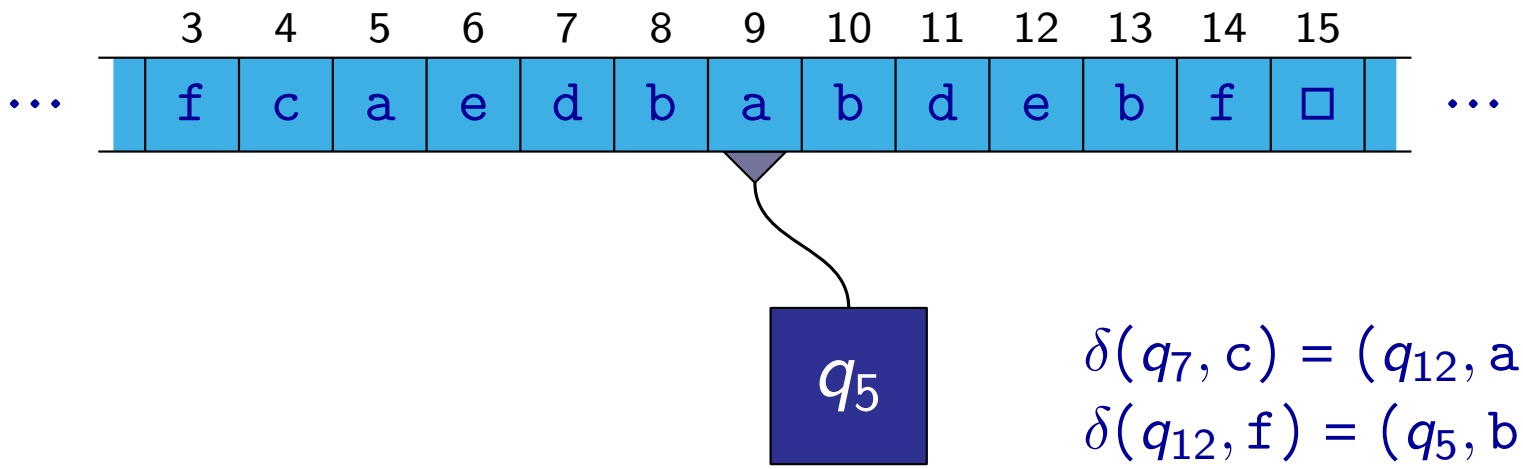
$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



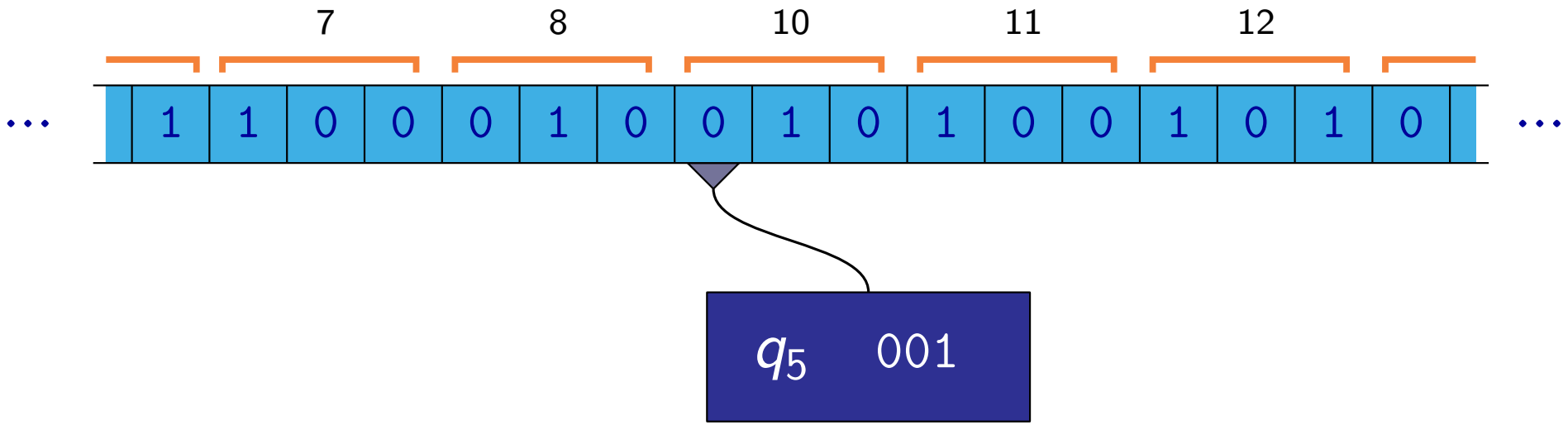
Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ:



$$\delta(q_7, \mathtt{c}) = (q_{12}, \mathtt{a}, +1)$$
$$\delta(q_{12}, \mathtt{f}) = (q_5, \mathtt{b}, -1)$$

Stroj s abecedou $\{0, 1\}$:



Abeceda $\{0, 1\}$

Při výše uvedené simulaci je jeden krok původního stroje simulován $k + 1$ kroky, kde k je počet bitů kódující jeden symbol abecedy Γ.

Pokud tedy původní stroj provede během výpočtu t kroků, simulující stroj provede $\mathcal{O}(t)$ kroků.

Zmenšení počtu stavů řídící jednotky

Poznámka: Tak, jako je možné zmenšit páskovou abecedu na pouhé dva symboly za cenu nárůstu velikosti počtu stavů řídící jednotky, je rovněž možné snížit počet stavů řídící jednotky:

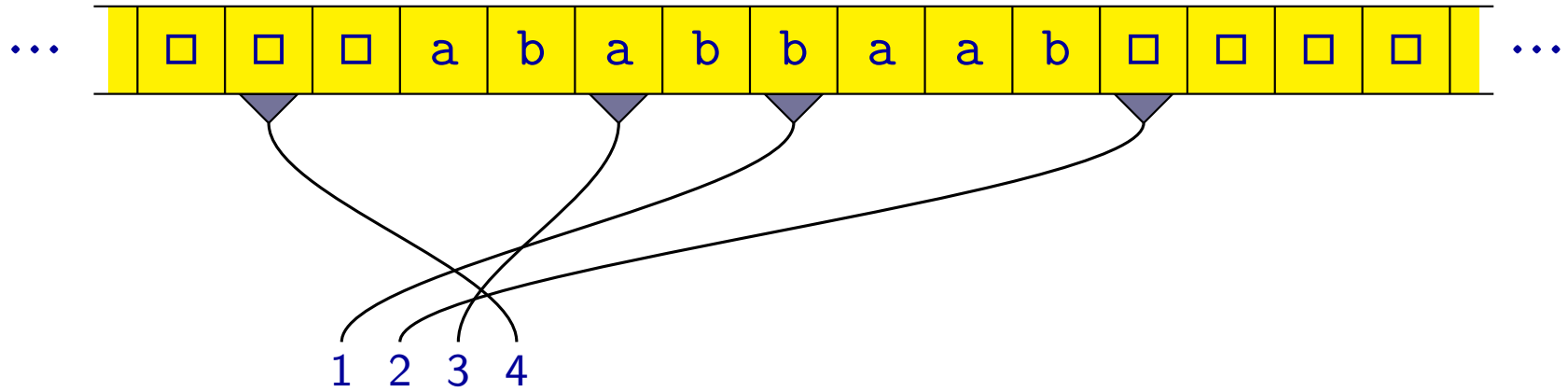
- Činnost libovolného Turingova stroje je možné simulovat Turingovým strojem, který má pouze dva nekoncové stavy řídící jednotky (a případně nějaké koncové stavy), ovšem za cenu nárůstu velikosti páskové abecedy.

Podobně jako v předchozím případě je jeden krok původního stroje simulován s kroky, kde s je konstanta závisející pouze na počtu stavů řídící jednotky původního stroje (tj. na velikosti množiny Q).

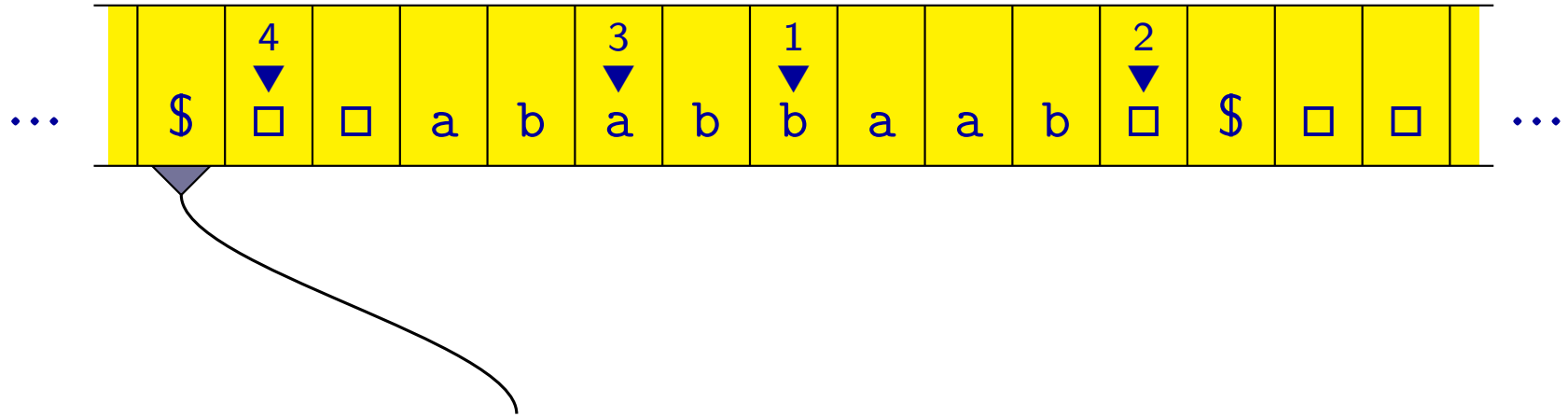
Opět zde tedy platí, že pokud původní stroj provede během výpočtu t kroků, simulující stroj provede $\mathcal{O}(t)$ kroků.

Simulace více hlav na pásce pomocí jedné

Více hlav na pásce:

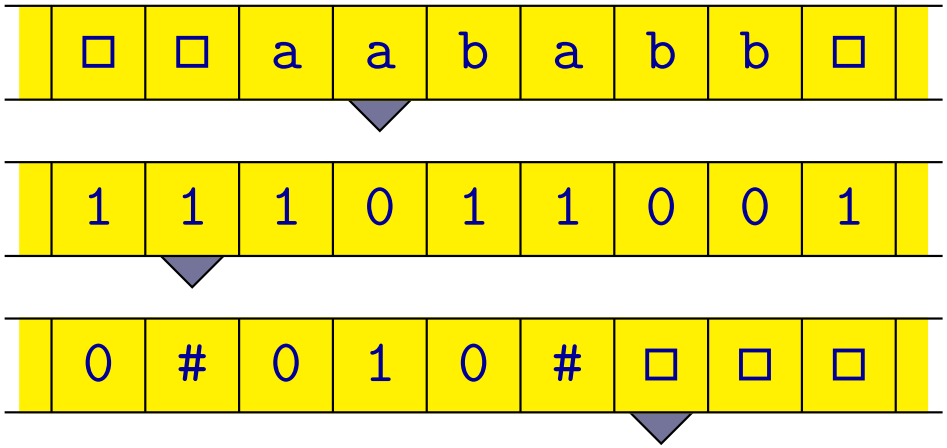


Páska s jednou hlavou:

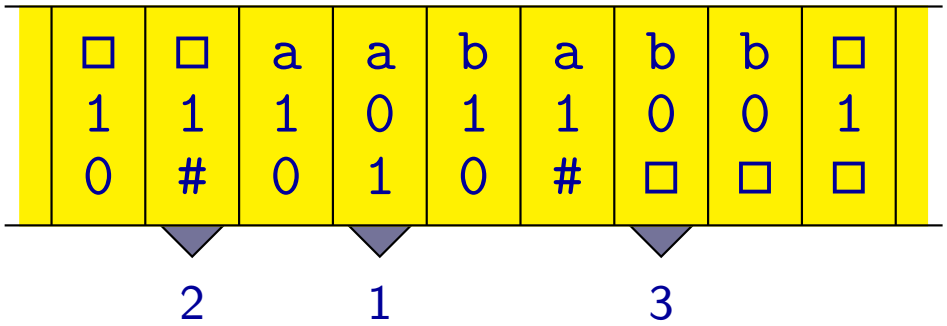


Simulace více pásek pomocí jedné

Více pásek:

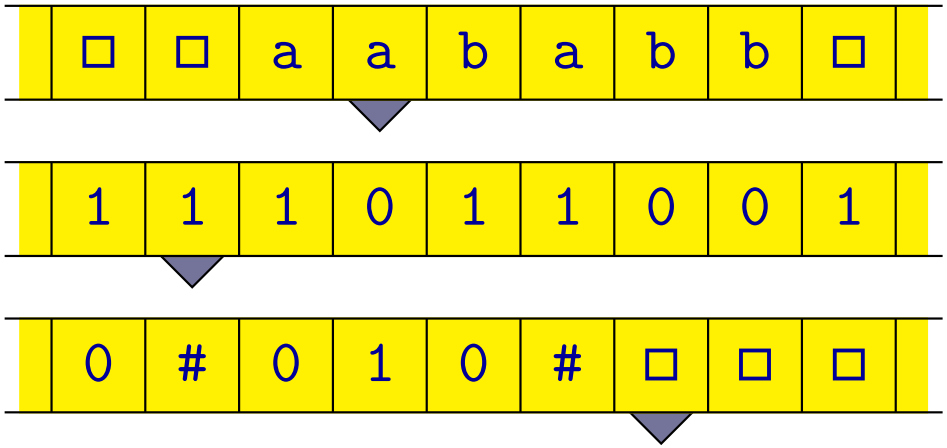


Jedna páska s více hlavami:

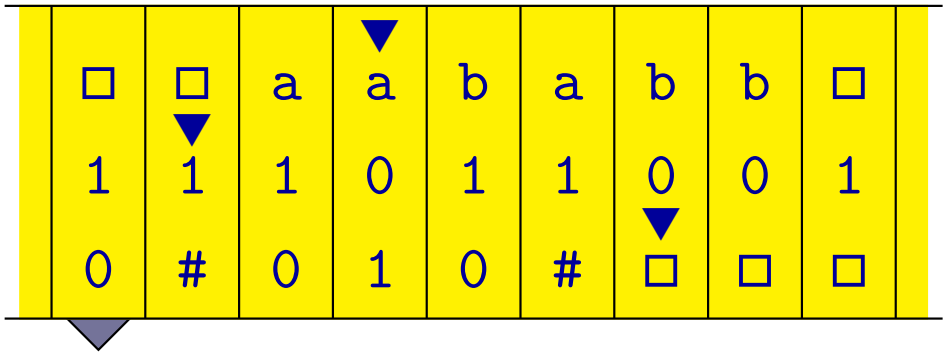


Simulace více pásek pomocí jedné

Více pásek:

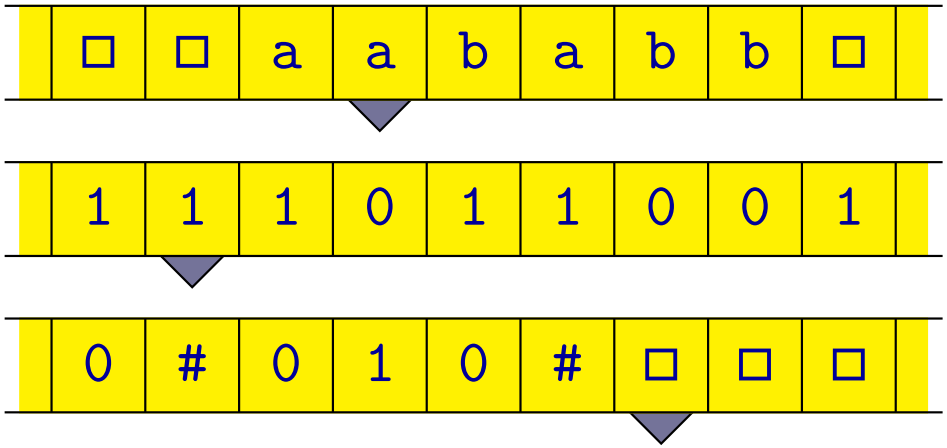


Jedna páska s jednou hlavou: varianta, kde se posunují značky hlav

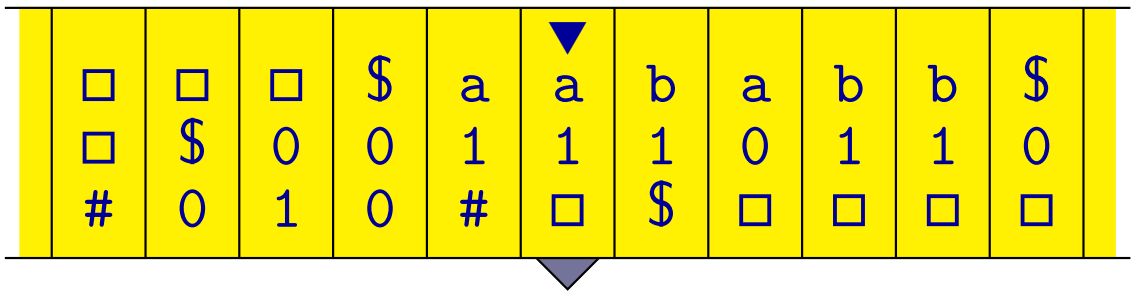


Simulace více pásek pomocí jedné

Více pásek:



Jedna páska s jednou hlavou: varianta, kde se posunují obsahy pásek



Simulace více pásek nebo více hlav pomocí jedné

- Všechny výše uvedené simulace více pásek (nebo jedné pásky s více hlavami) pomocí jedné pásky s jednou hlavou vyžadují pro simulaci jednoho kroku v nejhorším případě až $\mathcal{O}(m)$ kroků, kde m je maximální počet políček, které původní stroj navštívil na libovolné z pásek.

- Pokud původní stroj provedl t kroků, mohl na kterékoli pásce navštívit maximálně t políček.

- Simulace výpočtu, při kterém bylo provedeno t kroků, tedy vyžaduje nejvýše $\mathcal{O}(t^2)$ kroků.

Simulace více pásek nebo více hlav pomocí jedné

Doplňující poznámky:

- Existují algoritmické problémy, které se dají Turingovým strojem se dvěma páskami (nebo se dvěma hlavami na jedné pásce) snadno řešit v lineární čase (tj. počet provedených kroků je v $\mathcal{O}(n)$, kde n je velikost vstupu), ale přitom se pro ně dá dokázat, že jakýkoli Turingův stroj s jednou páskou a jednou hlavou, který by je řešil, musí alespoň pro některé vstupy provést minimálně $\Omega(n^2)$ kroků.

  Příklad:
  Rozpoznávání palindromů, tj. slov jazyka

$$\{ w \in \{a, b\}^* \mid w = w^R \}$$

Simulace více pásek nebo více hlav pomocí jedné

- Existují (komplikované a netriviální) způsoby, jak činnost pásky s více hlavami simulovat pomocí více pásek, kde ale každá páska má pouze jednu hlavu, přičemž jeden krok původního stroje je simulován jedním krokem simulujícího stroje.

- Existuje (komplikovaný a netriviální) způsob, jak k pásek simulovat pomocí dvou pásek (kde každá má jen jednu hlavu) takovým způsobem, že pokud výpočet původního stroje trvá t kroků, simulující stroj celkem provede nanejvýš $\mathcal{O}(t \log t)$ kroků.

Stroj RAM a Turingův stroj

Každý program v každém programovacím jazyce by mohl být realizován jako program stroje RAM.

Není složité (i když je to trochu pracné) si rozmyslet, že libovolný algoritmus prováděný strojem RAM je možné realizovat také Turingovým strojem.

Turingův stroj je schopen realizovat libovolný algoritmus, který by bylo možné zapsat jako program v nějakém programovacím jazyce.

Poznámka: A samozřejmě naopak je možné strojem RAM simulovat činnost libovolného Turingova stroje.

Turingův stroj simulující činnost stroje RAM

Při popisu toho, jak simulovat činnost stroje RAM pomocí Turingova stroje, budeme postupovat po menších krocích:

- Ukážeme, jak činnost stroje RAM ve variantě, kterou jsme si popsali, simulovat variantou stroje RAM s poněkud jednoduššími instrukcemi.
- Ukážeme, jak činnost této jednodušší varianty stroje RAM simulovat vícepáskovým Turingovým strojem.
- Už dříve jsme viděli, jak činnosti vícepáskového Turingovat stroje simulovat pomocí jednopáskového Turingovat stroje.

Jednodušší varianta stroje RAM

Tato jednodušší varianta stroje RAM bude mít kromě pracovní paměti tři **registry**:

- **registr A** — téměř všechny instrukce pracují s tímto registrem, výsledky všech operací se ukládají do tohoto registru

  Poznámka: Tento druh registru se často označuje jako **akumulátor**.

- **registr B** — tento registr slouží k uložení druhého operandu pro aritmetické instrukce (první operand je vždy v akumulátoru)

- **registr C** — tento registr slouží k uložení adresy, na kterou bude zapisovat instrukce store

Jednodušší varianta stroje RAM

Přehled instrukcí:

$A := c$	– přiřazení konstanty
$B := A$	– přiřazení do registru B
$C := A$	– přiřazení do registru C
$A := [A]$	– load (čtení z paměti)
$[C] := A$	– store (zápis do paměti)
$A := A \; op \; B$	– aritmetické instrukce, $op \in \{+, -, *, /\}$
if $(A \; rel \; 0)$ **goto** ℓ	– podmíněný skok, $rel \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– nepodmíněný skok
$A :=$ READ $()$	– čtení ze vstupu
WRITE (A)	– zápis na výstup
halt	– zastavení programu

Jednodušší varianta stroje RAM

Například instrukce

$$R_5 := 42$$

může být nahrazena posloupností instrukcí:

$$A := 5$$
$$C := A$$
$$A := 42$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$R_{12} := R_3$$

může být nahrazena posloupností instrukcí:

$$A := 12$$
$$C := A$$
$$A := 3$$
$$A := [A]$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$R_8 := [R_2]$$

může být nahrazena posloupností instrukcí:

$$A := 8$$
$$C := A$$
$$A := 2$$
$$A := [A]$$
$$A := [A]$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$[R_{15}] := R_9$$

může být nahrazena posloupností instrukcí:

$$A := 15$$
$$A := [A]$$
$$C := A$$
$$A := 9$$
$$A := [A]$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$R_7 := R_3 + R_6$$

může být nahrazena posloupností instrukcí:

$$A := 7$$
$$C := A$$
$$A := 6$$
$$A := [A]$$
$$B := A$$
$$A := 3$$
$$A := [A]$$
$$A := A + B$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$\textbf{if } (R_4 \geq R_{11}) \textbf{ goto } \ell$$

může být nahrazena posloupností instrukcí:

$$A := 11$$
$$A := [A]$$
$$B := A$$
$$A := 4$$
$$A := [A]$$
$$A := A - B$$
$$\textbf{if } (A \geq 0) \textbf{ goto } \ell$$

Jednodušší varianta stroje RAM

Například instrukce

$$R_{23} := \text{READ}()$$

může být nahrazena posloupností instrukcí:

$$A := 23$$
$$C := A$$
$$A := \text{READ}()$$
$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

$$\text{WRITE}\,(R_{17})$$

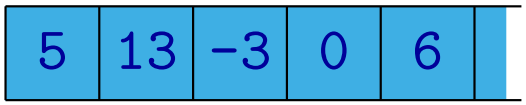
může být nahrazena posloupností instrukcí:

$$A := 17$$
$$A := [A]$$
$$\text{WRITE}\,(A)$$

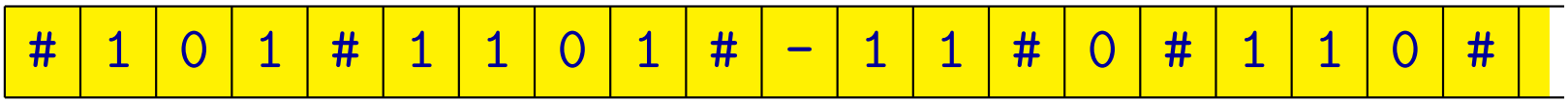
Turingův stroj simulující činnost stroje RAM

Turingův stroj pracuje se slovy nad nějakou abecedou, zatímco stroj RAM s čísly. Čísla ale můžeme zapisovat jako sekvence symbolů a naopak symboly nějaké abecedy můžeme zapisovat jako čísla.

Například následující vstup stroje RAM



může být v případě Turingova stroje reprezentován jako



Turingův stroj simulující činnost stroje RAM

Turingův stroj simulující činnost stroje RAM bude mít několik pásek:

- Pásku, na které bude uložen obsah pracovní paměti stroje RAM.
- Tři pásky, na kterých budou uloženy hodnoty registrů A, B a C.
  (Hodnoty registrů A, B a C budou na těchto páskách zapsány binárně bez vedoucích nul a zleva a zprava budou ohraničeny symboly #.)
- Pásku reprezentující vstupní pásku stroje RAM.
- Pásku reprezentující výstupní pásku stroje RAM.
- Jednu pomocnou pásku používanou při implementaci simulace jednotlivých instrukcí.

Turingův stroj simulující činnost stroje RAM

Turingův stroj si bude v řídící jednotce pamatovat, která instrukce stroje RAM se právě provádí.

Provedení většiny instrukcí není složité:

- $A := c$

  zapíše jednotlivé bity konstanty c na pásku registru A

- $B := A$ nebo $C := A$

  zkopíruje obsah pásky registru A na pásku registru B nebo C

- **goto** ℓ

  změní se jen stav řídící jednotky Turingova stroje

- **if** $(A \; rel \; 0)$ **goto** ℓ, kde $rel \in \{=, \neq, \leq, \geq, <, >\}$

  snadno se otestuje obsah registru A a podle výsledku se změní stav řídící jednotky Turingova stroje

Turingův stroj simulující činnost stroje RAM

- $A :=$ READ $()$

  zkopírování hodnoty (ohraničené znaky "#") ze vstupní pásky na pásku registru A

- WRITE (A)

  zkopírování hodnoty registru A na výstupní pásku.

- **halt**

  výpočet se zastaví

Turingův stroj simulující činnost stroje RAM

Také aritmetické instrukce jsou poměrně jednoduché, i když o něco složitější než předchozí instrukce:

- $A := A \; op \; B$, kde $op \in \{+, -, *, /\}$

  Příslušnou operaci (např. sčítání nebo odčítání) provede Turingův stroj bit po bitu, výsledek je ukládán do registru A.

Poznámka: Násobení a dělení je možné realizovat pomocí série sčítání, odčítání a bitových posunů.

Při implementaci násobení a dělení může být potřeba použít pomocnou pásku k ukládání mezivýsledků.

Turingův stroj simulující činnost stroje RAM

Asi nejsložitější je realizace pracovní paměti stroje RAM.

Jednou z možností je pamatovat si jen obsah těch buněk, se kterými stroj RAM v průběhu své činnosti někdy pracoval.

Příklad: Stroj RAM zatím pracoval jen s buňkami 2, 3 a 6:

- Buňka 2 obsahuje hodnotu 11.
- Buňka 3 obsahuje hodnotu -1.
- Buňka 6 obsahuje hodnotu 2.

Obsah pásky Turingova stroje reprezentující buňky paměti stroje RAM bude následující:

$	#	1	0	:	1	0	1	1	#	1	1	:	-	1	#	1	1	0	:	1	0	#	$	

Turingův stroj simulující činnost stroje RAM

Instrukce load, tj. $A := [A]$:

- Turingův stroj bude hledat příslušnou adresu uloženou v registru A na pásce reprezentující obsah paměti stroje RAM.
  (Pokud ji nenajde, přidá ji na konec, s tím, že obsahuje hodnotu 0.)
- Příslušnou hodnotu zkopíruje na pásku registru A.

Turingův stroj simulující činnost stroje RAM

Instrukce store, tj. $[C] := A$:

- Podobně jako u instrukce load se najde příslušné místo na pásce reprezentující pracovní paměť, kde se nachází obsah buňky, jejíž adresa je v registru C.
- Zbytek pásky s obsahem paměti stroje RAM se zkopíruje na pomocnou pásku.
- Na příslušné místo se zkopíruje obsah pásky registru A.
- Zbytek pásky, který byl zkopírován na pomocnou pásku, se zkopíruje zpět (za nově zapsanou hodnotu).

Turingův stroj simulující činnost stroje RAM

Není těžké si promyslet, že při výše popsané simulaci výpočtu stroje RAM Turingovým strojem, je počet kroků provedených Turingovým strojem polynomiální (zhruba kvadratický) vůči době výpočtu původního stroje RAM v logaritmické míře.

Poznámka: U strojů RAM, které nemají operaci násobení, je tento počet kroků simulujícího Turingova stroje polynomiální i vůči době výpočtu stroje RAM počítané v jednotkové míře.

Pokud původní stroj RAM provede t instrukcí, simulující Turingův stroj provede zhruba $\mathcal{O}(t^3)$ instrukcí.

Pásky, zásobníky a čítače

Všechny následující stroje mají konečnou řídící jednotku doplněnou o nějaký druh neomezeně velké paměti.

Tato paměť může být tvořena jednou nebo více strukturami, jako jsou třeba:

- **Páska** — čtení a zápis symbolu na aktuální pozici, posun hlavy doleva a doprava

  Poznámka: Páska může být jednostranně nebo oboustranně nekonečná.

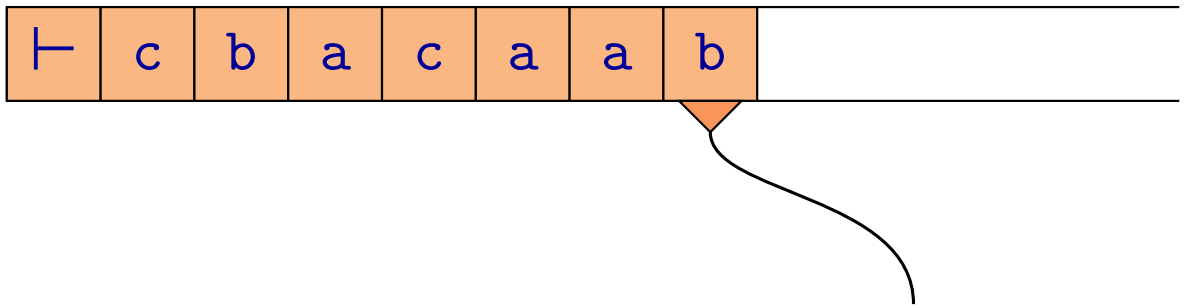
- **Zásobník** — push, pop, test prázdnosti zásobníku

- **Čítač** — hodnotou je přirozené číslo, operace přičtení nebo odečtení hodnoty jedna, test rovnosti nule

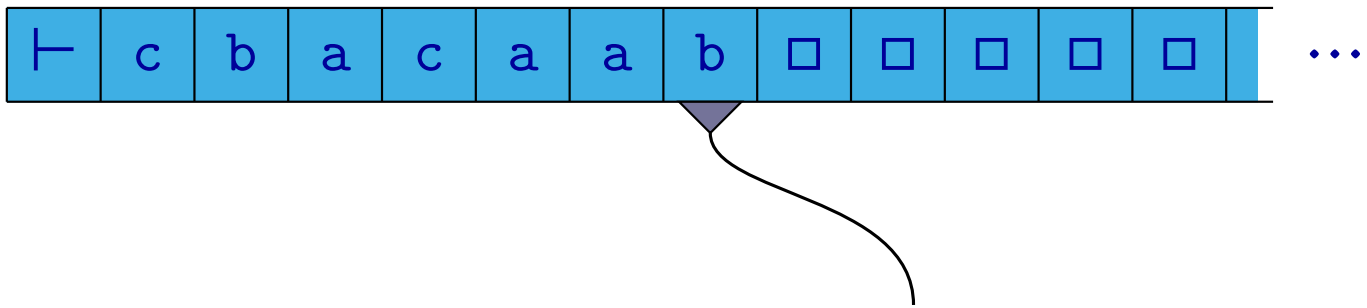
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



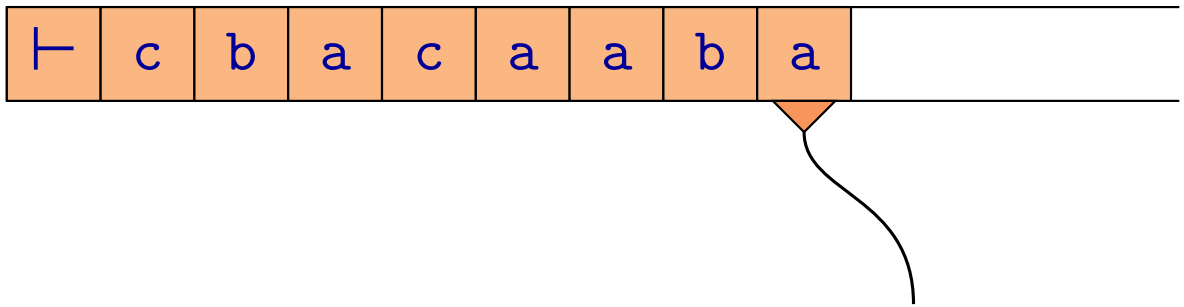
Páska:



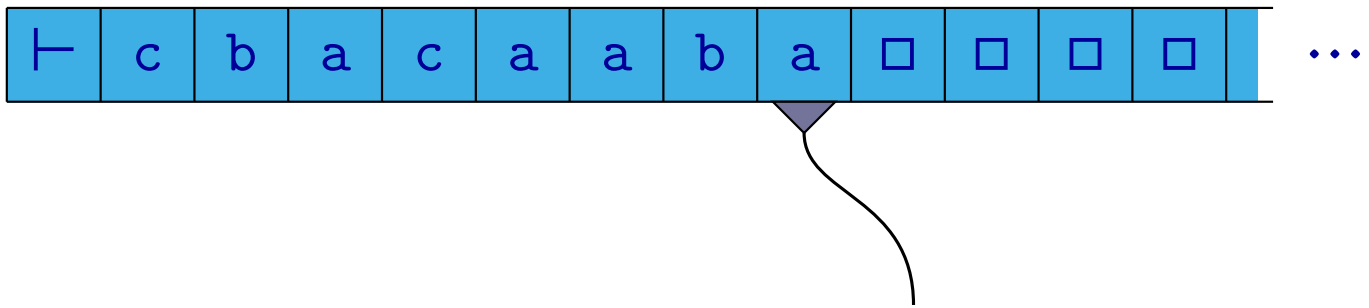
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



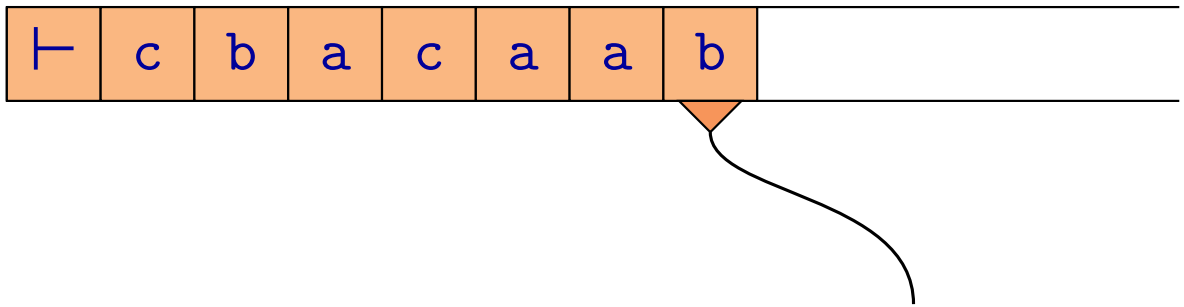
Páska:



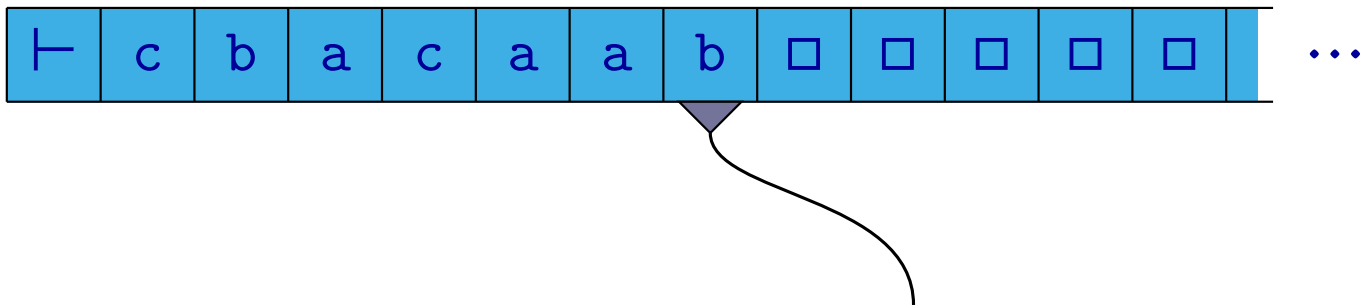
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



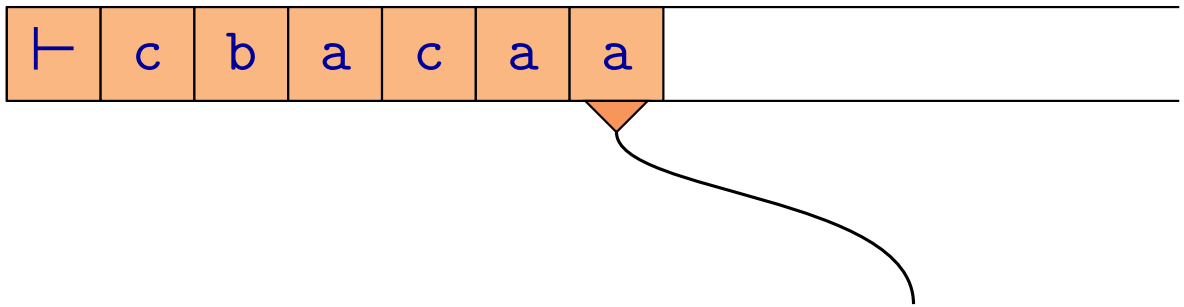
Páska:



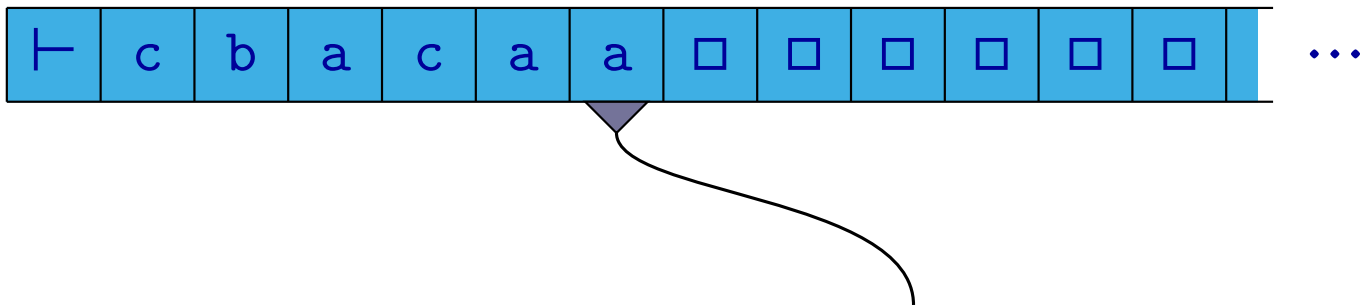
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



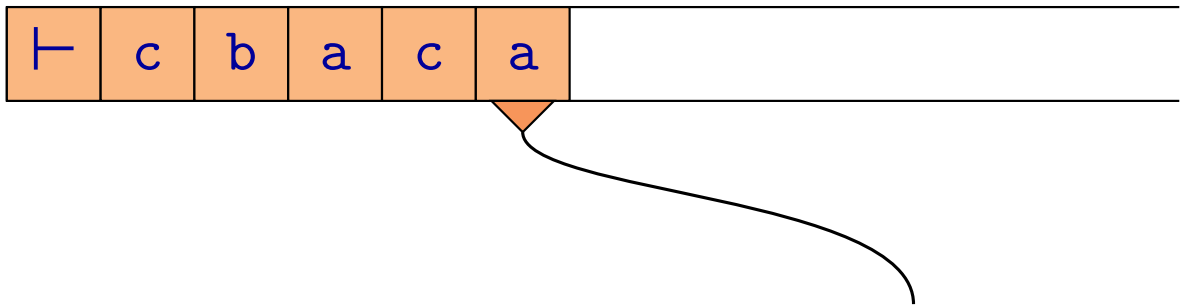
Páska:



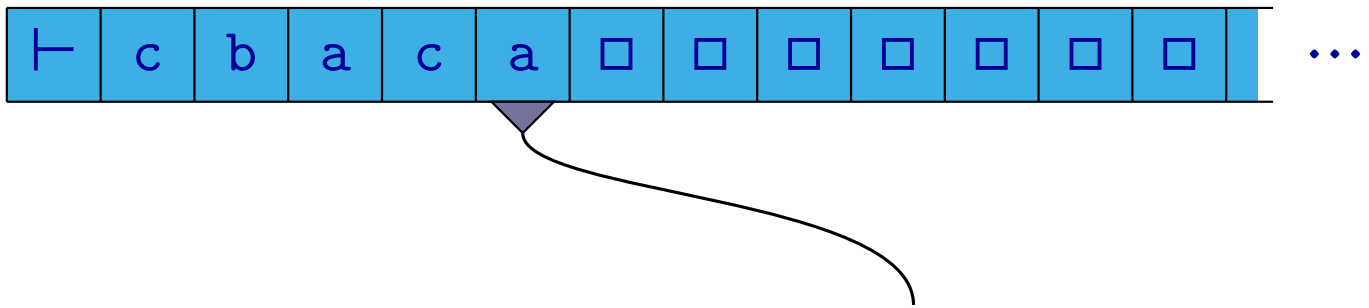
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:

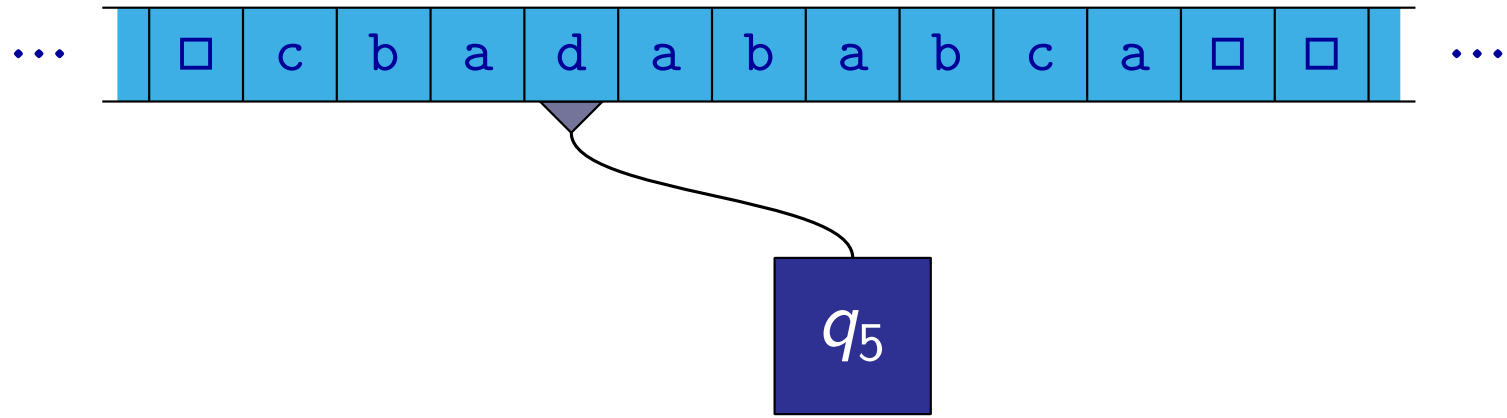


Páska:

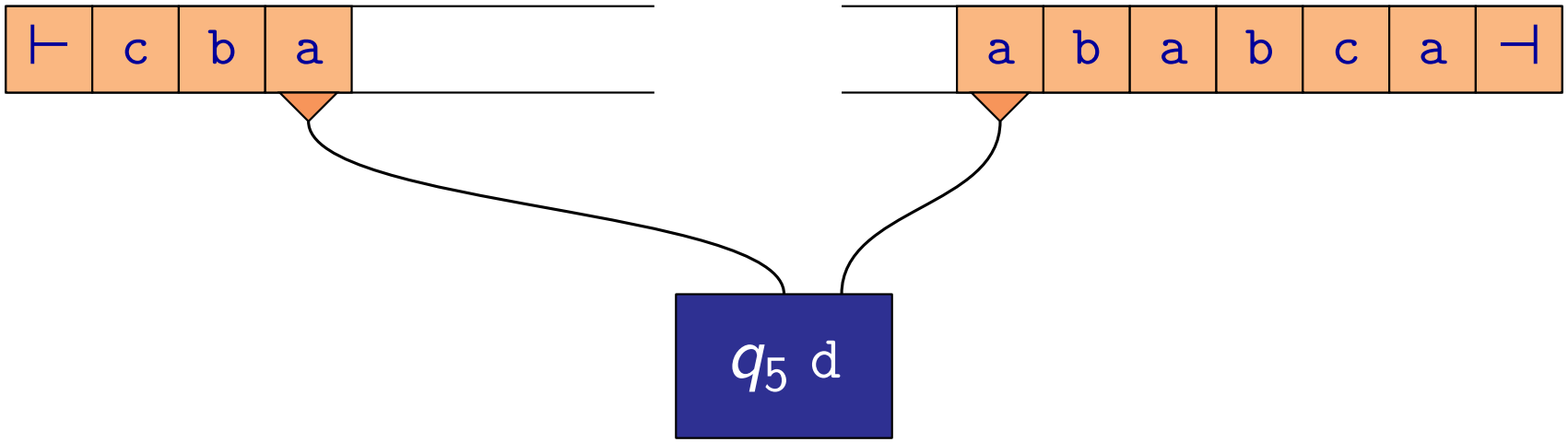


Zásobník

Oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků:

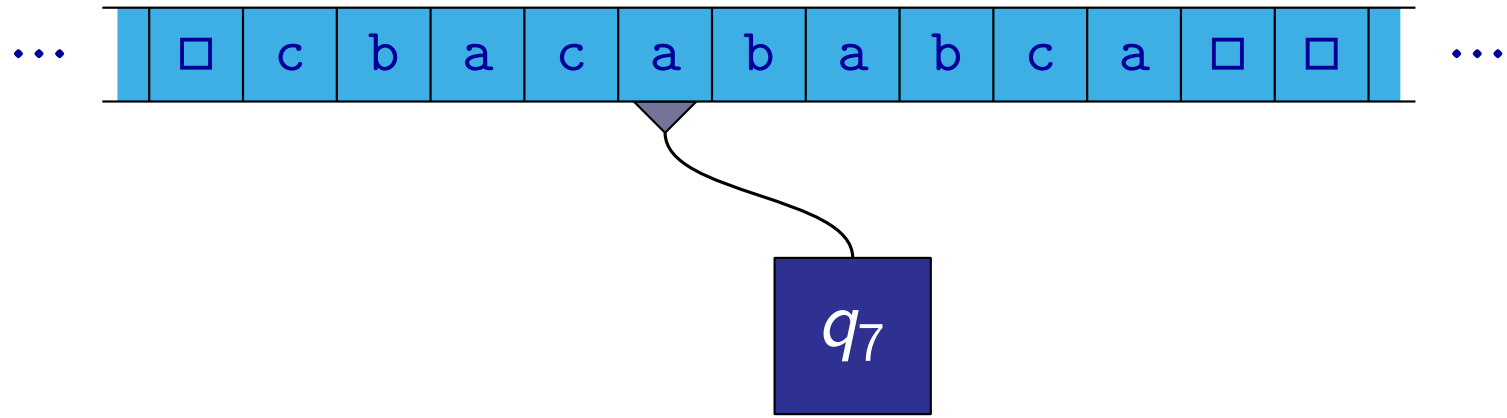


Stroj se dvěma zásobníky:

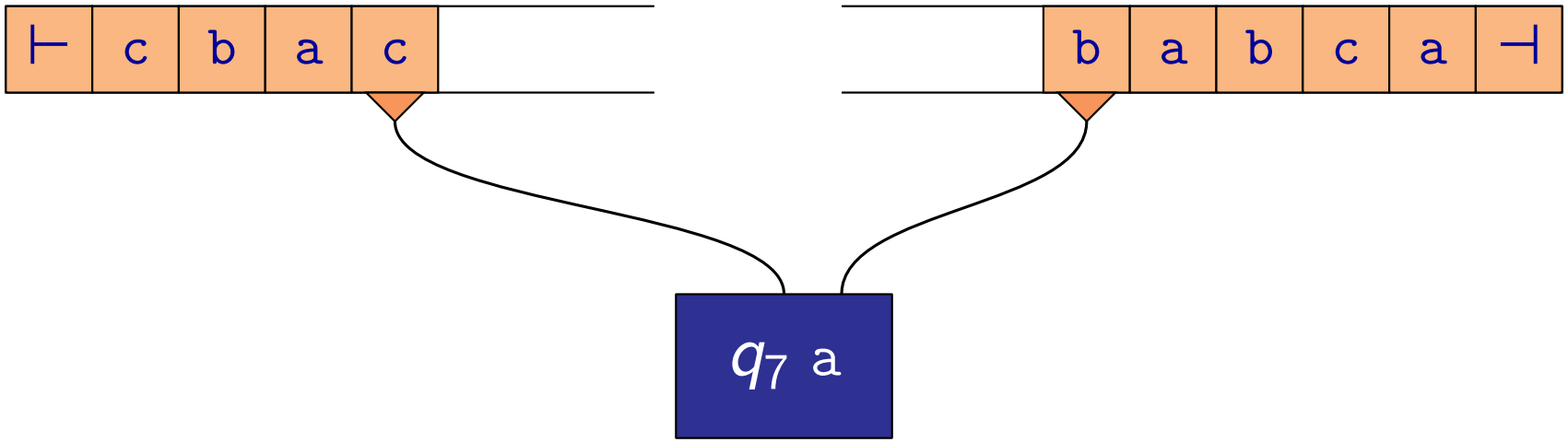


Zásobník

Oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků:



Stroj se dvěma zásobníky:



Čítač

Čítač — hodnotou čítače může být libovolně velké přirozené číslo, tj. prvek množiny $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.

Základní operace:

- zvýšení hodnoty o jedna:

$$x := x + 1$$

- snížení hodnoty o jedna:

$$x := x - 1$$

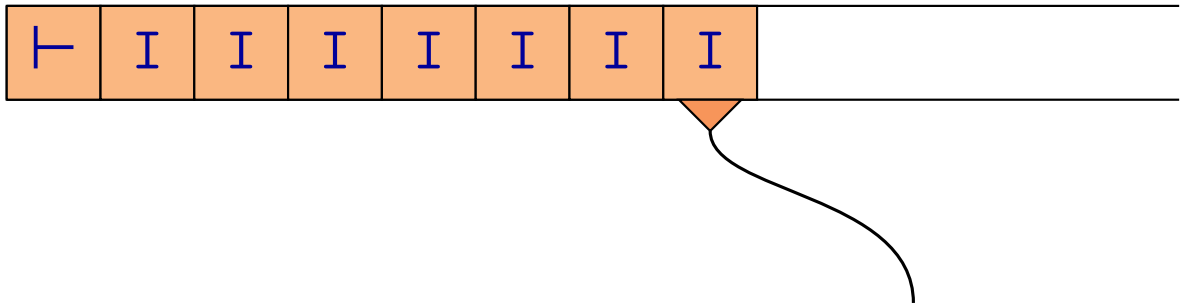
- test, jestli je hodnota čítače nula:

$$\textbf{if } (x = 0) \textbf{ goto } \ell$$

Čítač

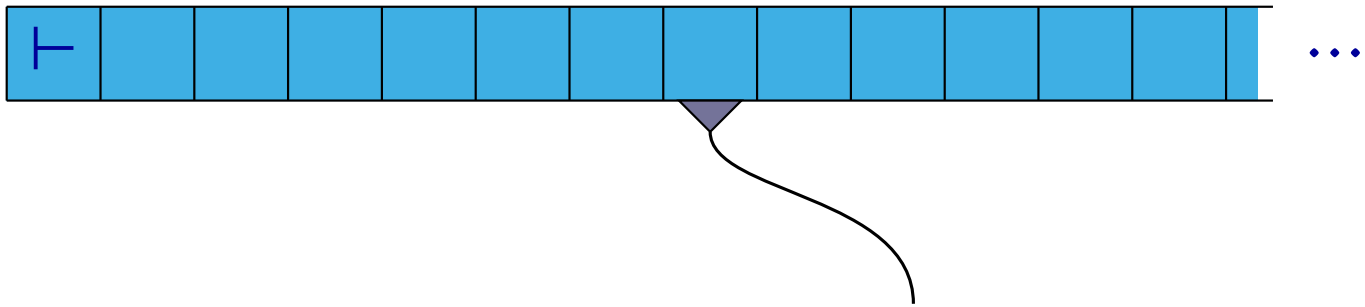
Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

Zásobník:



Čítač:

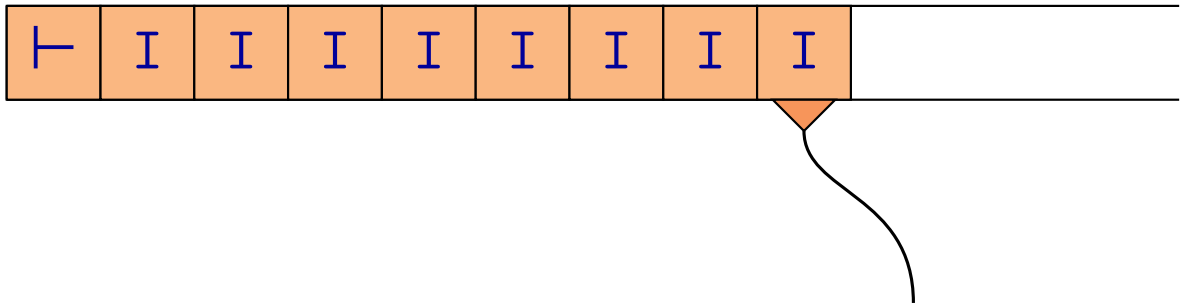
Páska:



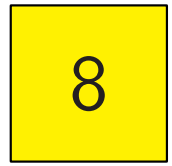
Čítač

Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

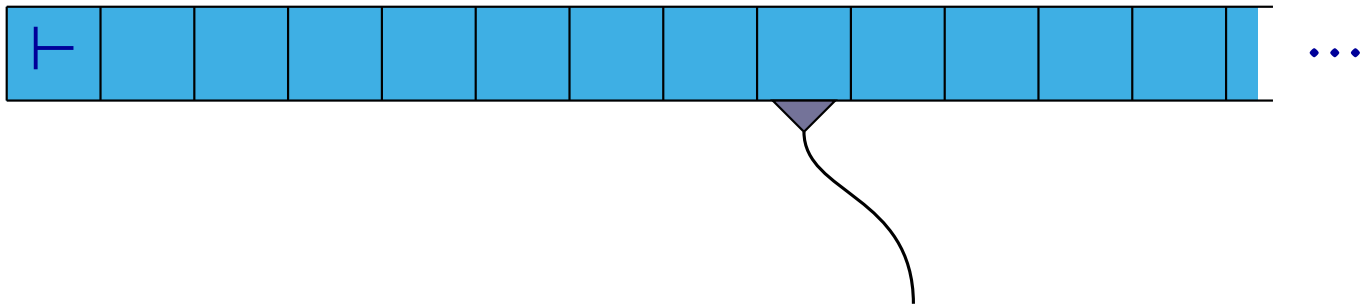
Zásobník:



Čítač:



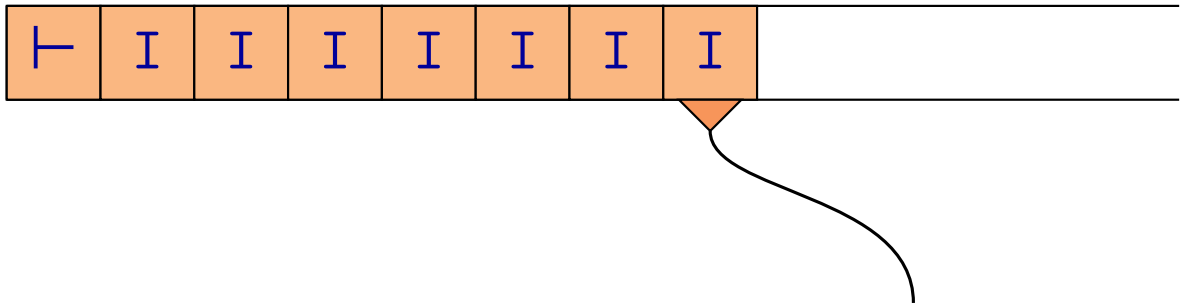
Páska:



Čítač

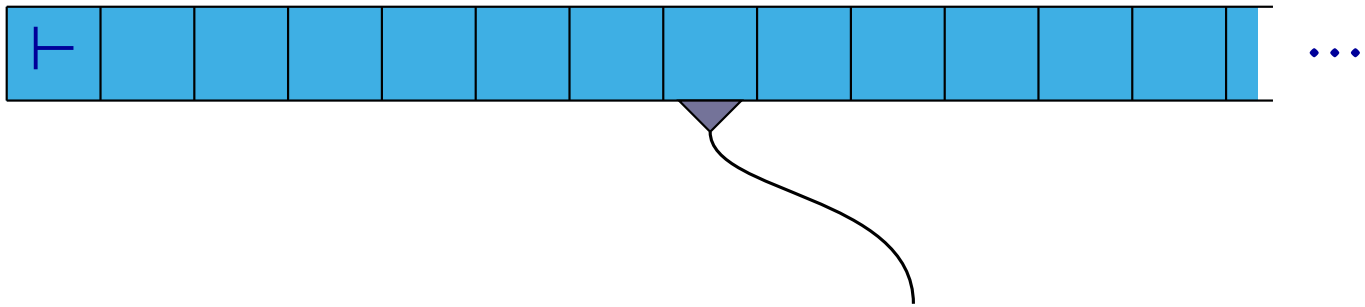
Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

Zásobník:



Čítač:

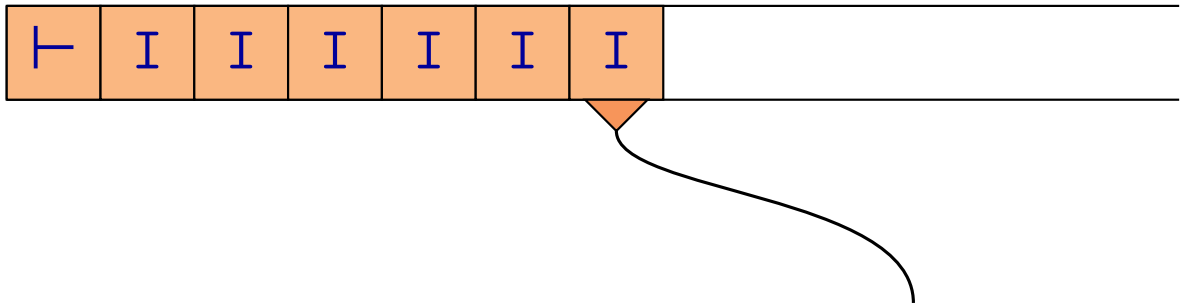
Páska:



Čítač

Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

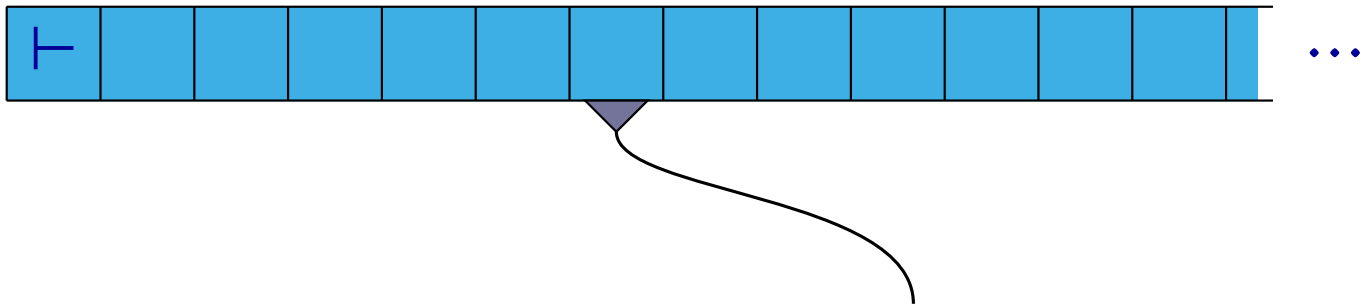
Zásobník:



Čítač:

6

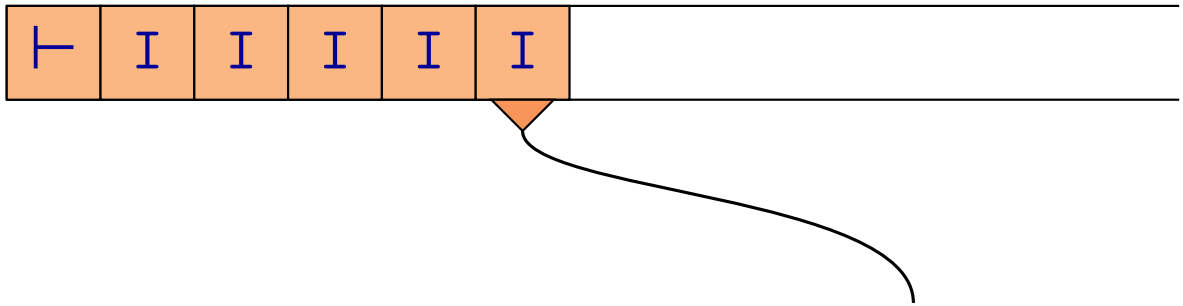
Páska:



Čítač

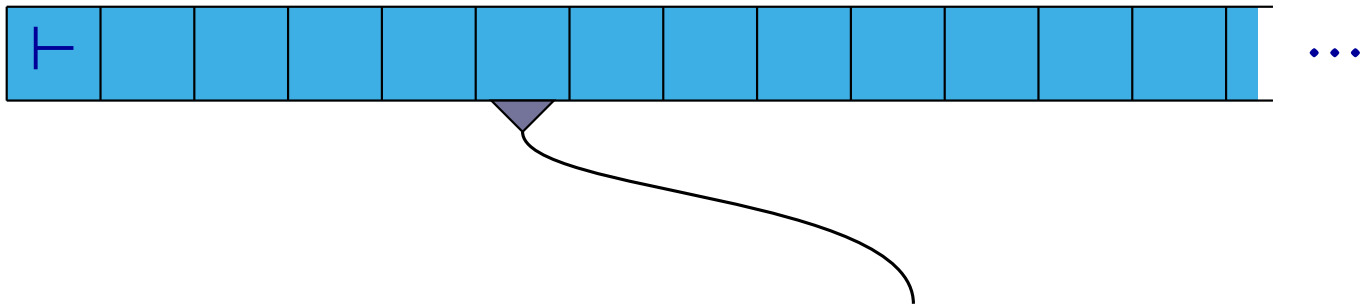
Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

Zásobník:



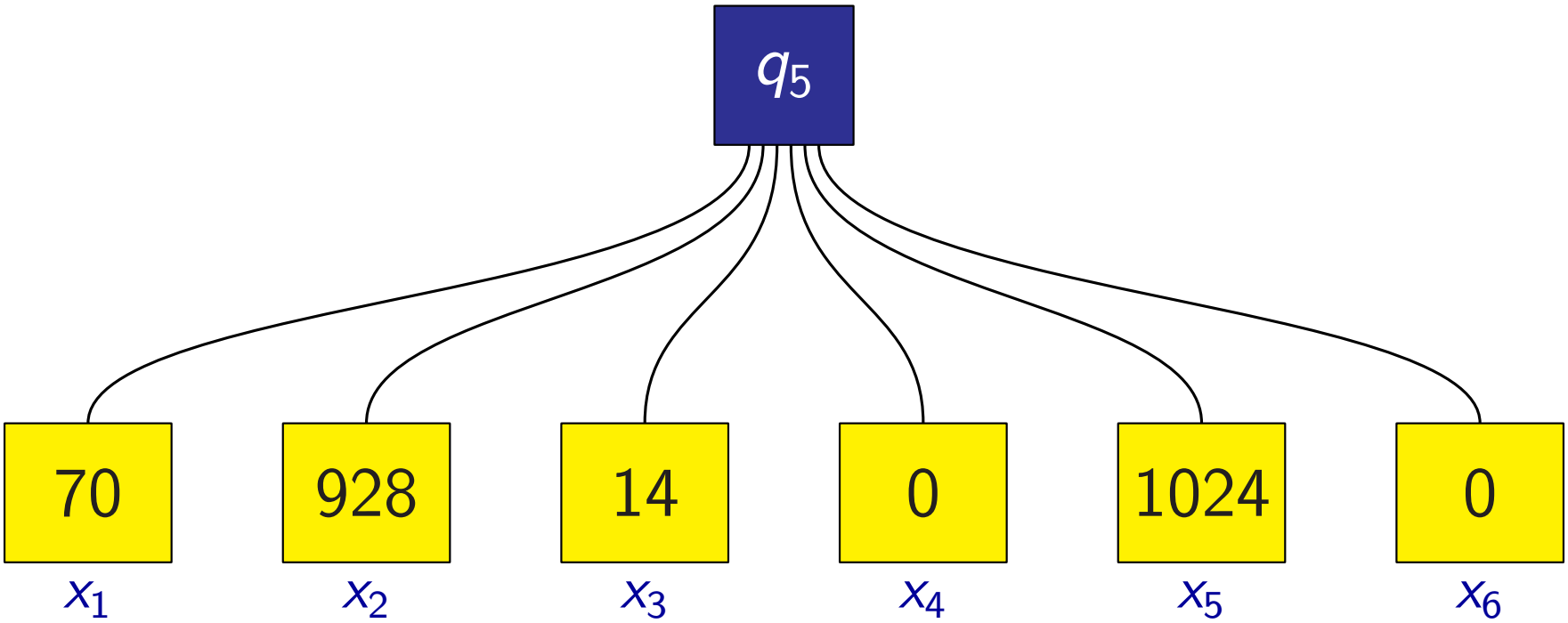
Čítač:

Páska:



Minského stroj

Minského stroj — stroj, který má konečnou řídící jednotku a konečný počet čítačů $x_1, x_2, \ldots, x_k$:



Poznámka: Pro označení čítačů budeme kromě symbolů $x_1, x_2, \ldots$ používat také symboly jako $x, y, z, \ldots$

Minského stroj

Na Minského stroj se můžeme dívat jako na program tvořený posloupností instrukcí následujících pěti typů:

- zvýšení hodnoty daného čítače o jedna:

$$x_i := x_i + 1$$

- snížení hodnoty daného čítače o jedna:

$$x_i := x_i - 1$$

- test, jestli je hodnota daného čítače nula:

$$\textbf{if } (x_i = 0) \textbf{ goto } \ell$$

- nepodmíněný skok:

$$\textbf{goto } \ell$$

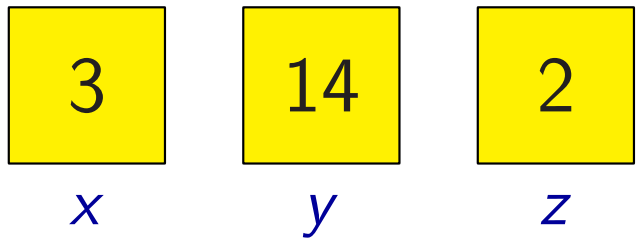
- zastavení programu:

$$\textbf{halt}$$

Minského stroj

Vynulování čítače x:

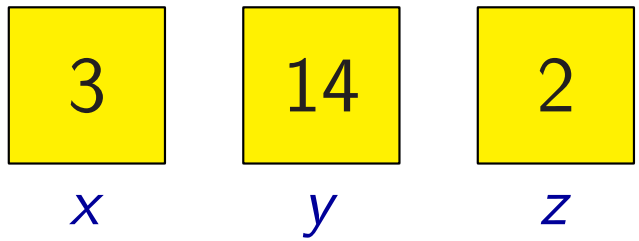
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

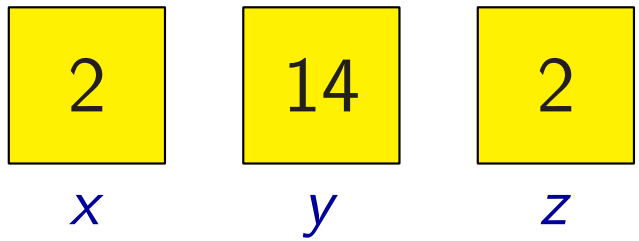
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

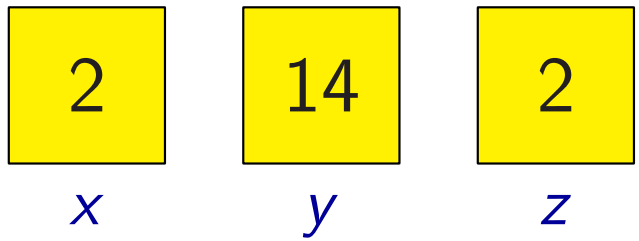
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

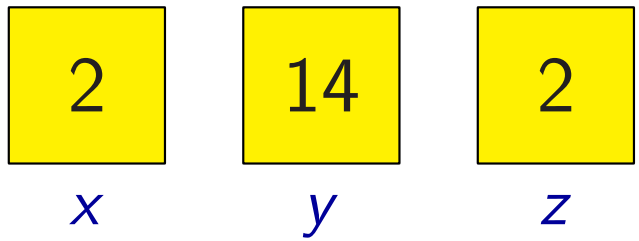
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

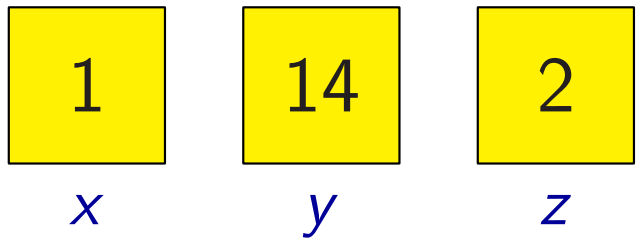
L_1 : **if** $(x = 0)$ **goto** L_2
$\quad x := x - 1$
$\quad$ **goto** L_1
L_2 : $\quad \ldots$



Minského stroj

Vynulování čítače x:

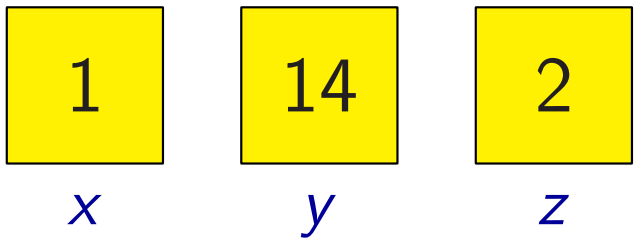
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

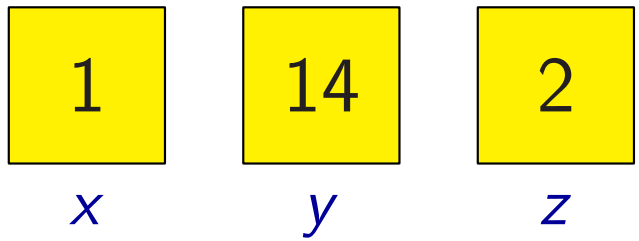
Vynulování čítače x:

L_1 : **if** $(x = 0)$ **goto** L_2

$x := x - 1$

goto L_1

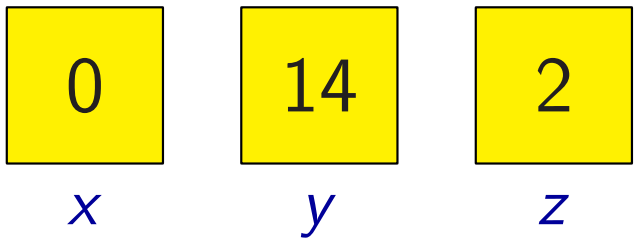
L_2 : ...



Minského stroj

Vynulování čítače x:

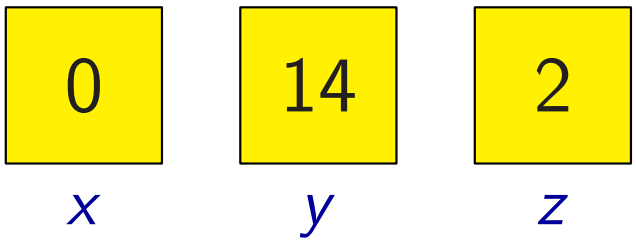
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

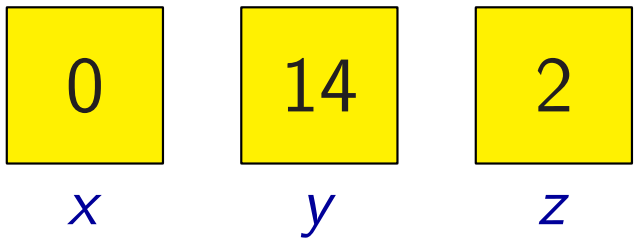
L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : ...



Minského stroj

Vynulování čítače x:

L_1 : **if** $(x = 0)$ **goto** L_2
$x := x - 1$
goto L_1
L_2 : $\ldots$



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

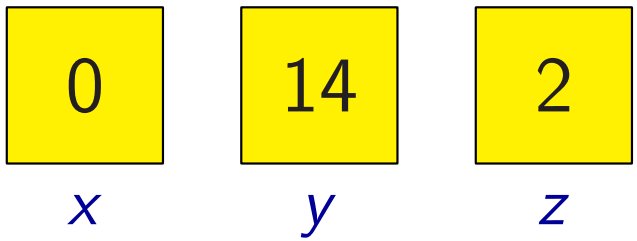
L_2 : **if** $(z = 0)$ **goto** L_3

$z := z - 1$

$y := y + 1$

goto L_1

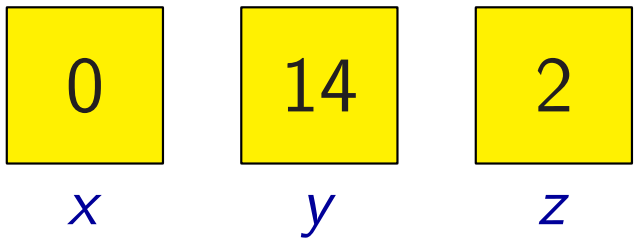
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

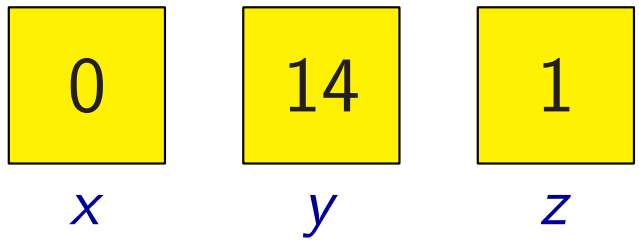
L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

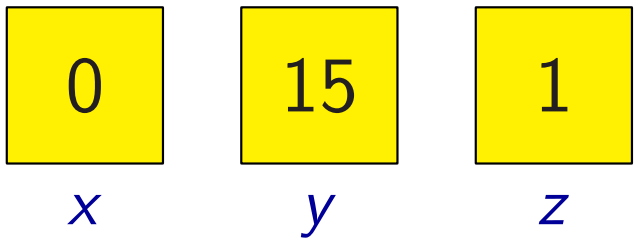
L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

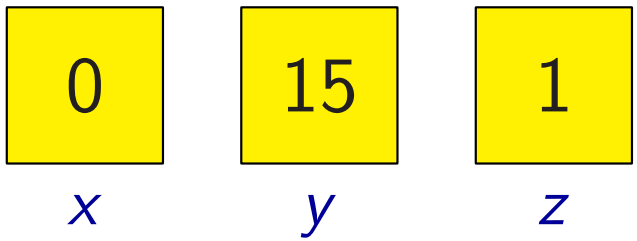
L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

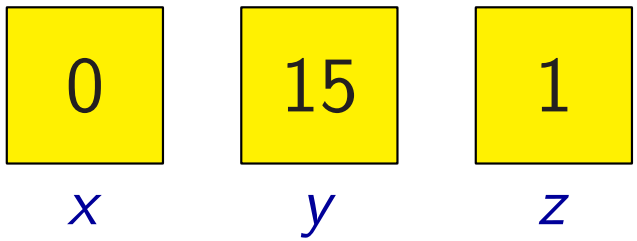
L_2 : **if** $(z = 0)$ **goto** L_3

→ $z := z - 1$

$y := y + 1$

goto L_1

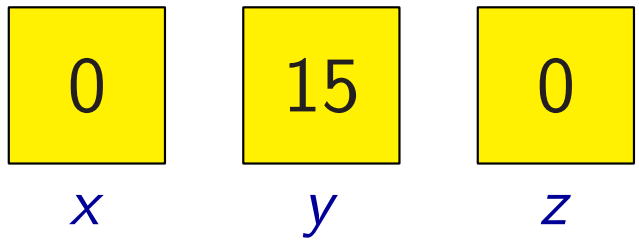
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

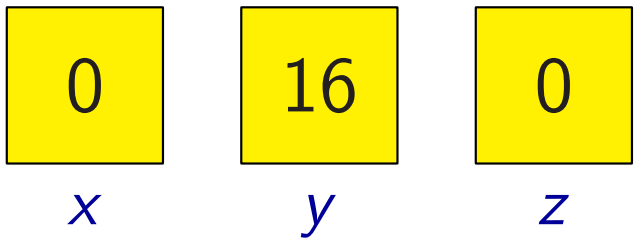
L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

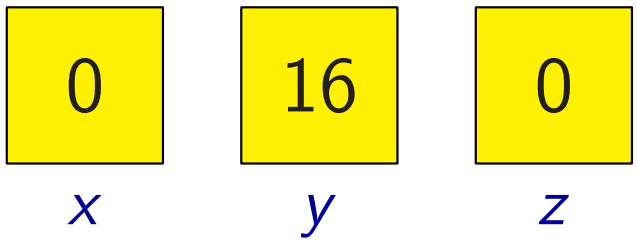
L_2 : **if** $(z = 0)$ **goto** L_3

$z := z - 1$

$y := y + 1$

goto L_1

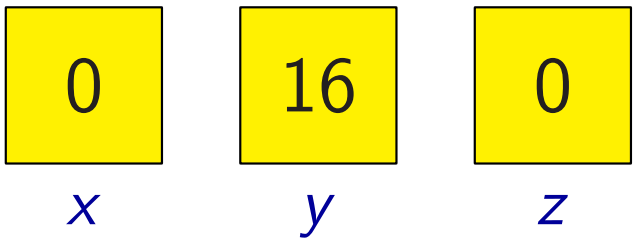
L_3 : ...



Minského stroj

Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** $(z = 0)$ **goto** L_3
$z := z - 1$
$y := y + 1$
goto L_1
L_3 : $\ldots$



Minského stroj

Vynásobení hodnoty čítače x číslem 5:

$$
\begin{aligned}
L_1 &: \textbf{if } (x = 0) \textbf{ goto } L_2 \\
&\quad x := x - 1 \\
&\quad y := y + 1 \\
&\quad y := y + 1 \\
&\quad y := y + 1 \\
&\quad y := y + 1 \\
&\quad y := y + 1 \\
&\quad \textbf{goto } L_1 \\
L_2 &: \textbf{if } (y = 0) \textbf{ goto } L_3 \\
&\quad y := y - 1 \\
&\quad x := x + 1 \\
&\quad \textbf{goto } L_2 \\
L_3 &: \quad \ldots
\end{aligned}
$$

Minského stroj

Vydělení hodnoty čítače x číslem 5 a zjištění zbytku po dělení:

$$
\begin{aligned}
L_1 : \ & \textbf{if } (x = 0) \textbf{ goto } M_0 \\
& x := x - 1 \\
& \textbf{if } (x = 0) \textbf{ goto } M_1 \\
& x := x - 1 \\
& \textbf{if } (x = 0) \textbf{ goto } M_2 \\
& x := x - 1 \\
& \textbf{if } (x = 0) \textbf{ goto } M_3 \\
& x := x - 1 \\
& \textbf{if } (x = 0) \textbf{ goto } M_4 \\
& x := x - 1 \\
& y := y + 1 \\
& \textbf{goto } L_1
\end{aligned}
$$

Minského stroj

Zásobník je možné simulovat pomocí dvou čítačů — hodnota jednoho čítače reprezentuje obsah zásobníku jako číslo, jehož zápis v číselné soustavě o základu $k = |\Gamma| + 1$ (kde Γ je zásobníková abeceda) odpovídá obsahu zásobníku.

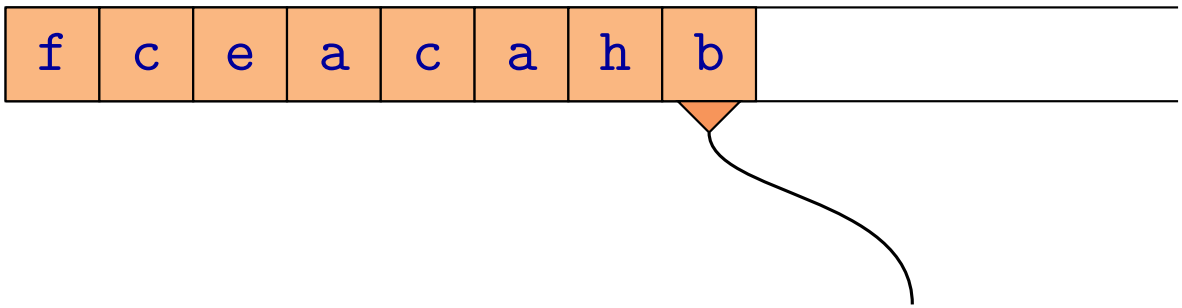
- Symbol na vrcholu zásobníku — zbytek po dělení číslem k
- Pop — vydělit číslem k
- Push — vynásobit číslem k a přičíst kód příslušného symbolu

Druhý čítač slouží jako pomocný při provádění výše uvedených operací.

Minského stroj

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9

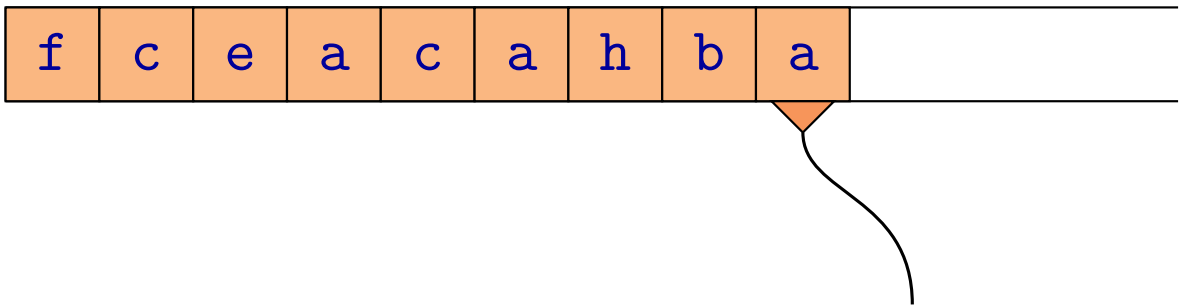


63513182

Minského stroj

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9

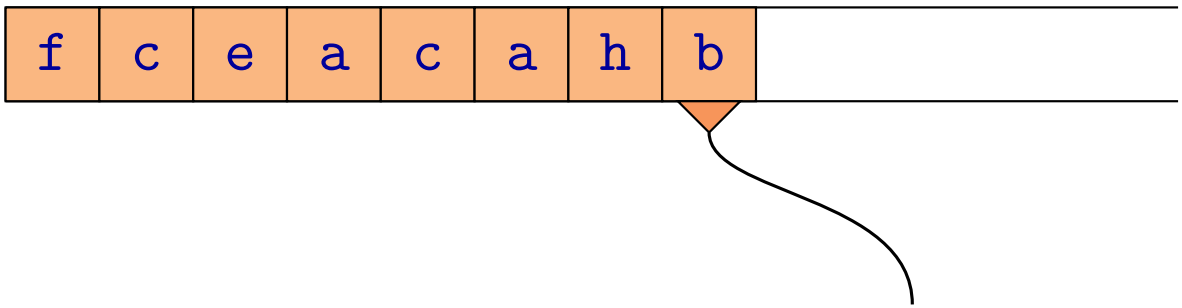


635131821

Minského stroj

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9

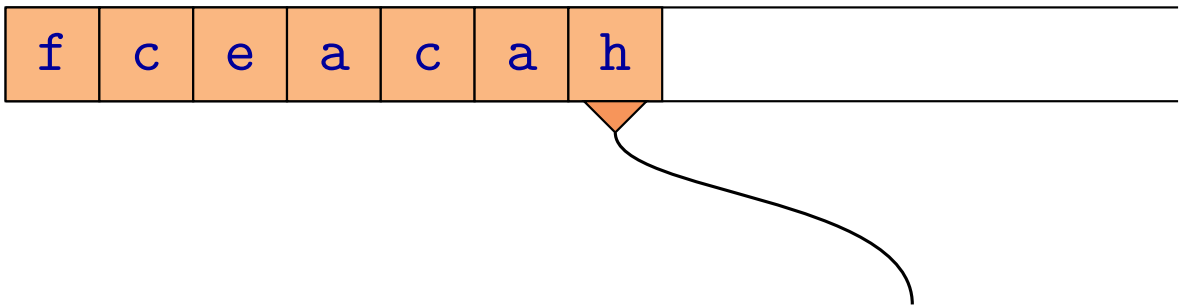


63513182

Minského stroj

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9

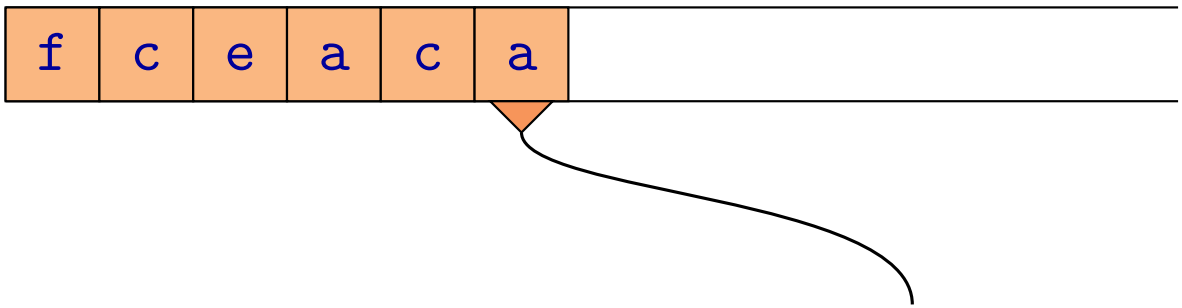


6351318

Minského stroj

Příklad:

$\texttt{a} \leftrightarrow 1$
$\texttt{b} \leftrightarrow 2$
$\texttt{c} \leftrightarrow 3$
$\texttt{d} \leftrightarrow 4$
$\texttt{e} \leftrightarrow 5$
$\texttt{f} \leftrightarrow 6$
$\texttt{g} \leftrightarrow 7$
$\texttt{h} \leftrightarrow 8$
$\texttt{i} \leftrightarrow 9$



635131

Minského stroj

Připomeňme, že oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků.

V Minského stroji může být obsah každého z těchto zásobníků reprezentován jemu odpovídajícím čítačem.

Navíc potřebujeme ještě jeden pomocný čítač pro implementaci operací násobení a dělení na těchto čítačích reprezentujících obsahy zásobníků.

Vidíme, že Turingův stroj s k páskami je možné simulovat Minského strojem s $2k + 1$ čítači.

Minského stroj

Libovolný konečný počet čítačů je možné simulovat pomocí dvou čítačů:

- Jeden čítač (označme jej C) reprezentuje hodnoty všech čítačů — např. hodnoty tří čítačů x, y, z mohou být v čítači C reprezentovány jako číslo $2^x 3^y 5^z$.
- Druhý čítač je používán jako pomocný při provádění operací násobení a dělení na čítači C.
- Přičtení jedničky k čítači x je simulováno jako vynásobení čítače C hodnotou 2, přičtení jedničky k čítači y jako vynásobení hodnotou 3, atd.
- Analogicky je odečtení jedničky od čítače x simulováno pomocí vydělení čítače C hodnotou 2, odečtení jedničky od čítače y vydělením hodnotou 3, atd.
- Test podmínky $x = 0$ odpovídá testu, že hodnota C není dělitelná dvěma, atd.

Minského stroj

Vidíme, že činnost libovolného Turingova stroje je možné simulovat Minského strojem s dvěma čítači.

Tato simulace je však mimořádně neefektivní:

- Už simulace pásky Turingova stroje pomocí tří čítačů vyžaduje exponenciálně větší počet kroků, než kolik by jich vykonal tento Turingův stroj.
- Simulace činnosti těchto tří čítačů pomocí dvou čítačů tento počet kroků dále exponenciálně zvyšuje.

Výpočet Turingova stroje jako data

Uvažujme nyní libovolné Turingovy stroje — tj. s libovolným počtem pásek, hlav, atd.

Není těžké si rozmyslet, že s konfiguracemi daného stroje $\mathcal{M}$ můžeme pracovat jako s daty.

Například jsme viděli, že konfigurace jednopáskového Turingova stroje $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ můžeme kódovat jako slova nad abecedou $\Delta = \Gamma \cup (Q \times \Gamma)$:

- V zápisu konfigurace je právě jeden symbol z množiny $(Q \times \Gamma)$ — reprezentuje stav řídící jednotky a pozici hlavy.
- Zbylé symboly (z množiny Γ) reprezentují obsah pásky.

Podobně můžeme navrhnout nějaký konkrétní způsob kódování konfigurací ve formě dat (např. ve formě slov nad nějakou abecedou) i pro další typy Turingových strojů.

Výpočet Turingova stroje jako data

Na **výpočet** (resp. na popis tohoto výpočtu) daného stroje $\mathcal{M}$ nad nějakým vstupem w tak rovněž můžeme nahlížet jako na určitý druh dat.

Výpočet je možné reprezentovat jako posloupnost konfigurací oddělených nějakým speciální symbolem, například $\# \notin \Delta$.

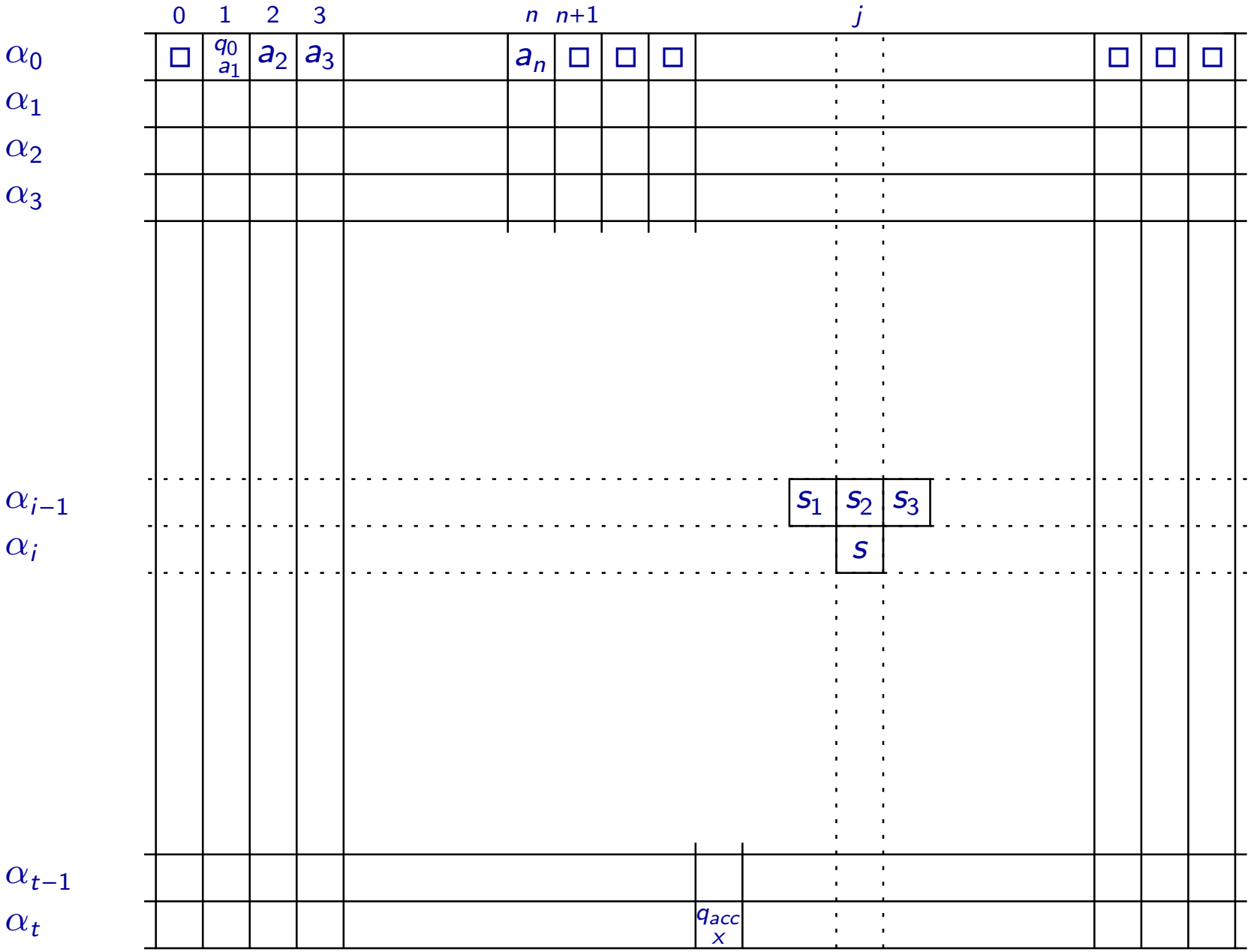
Ve výpočtu

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

se dvě po sobě jdoucí kofigurace α_{i-1} a α_i, tj. takové, že

$$\alpha_{i-1} \longrightarrow \alpha_i$$

mezi sebou liší vždy jen v několika málo místech.


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

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)

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

Problém je **částečně rozhodnutelný**, jestliže existuje algoritmus, který:

- Pokud dostane jako vstup instanci, pro kterou je odpověď ANO, tak se po konečném počtu kroků zastaví a vypíše `"ANO"`.
- Pokud dostane jako vstup instanci, pro kterou je odpověď NE, tak se buď zastaví a vypíše `"NE"` nebo se nikdy nezastaví.

Je očividné, že například HP (Halting problem) je částečně rozhodnutelný.

Některé problémy však nejsou ani částečně rozhodnutelné.

Postova věta

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.

Postova věta

Jestliže problém P i jeho doplňkový problém jsou částečně rozhodnutelné, pak je problém P rozhodnutelný.

Převody mezi problémy

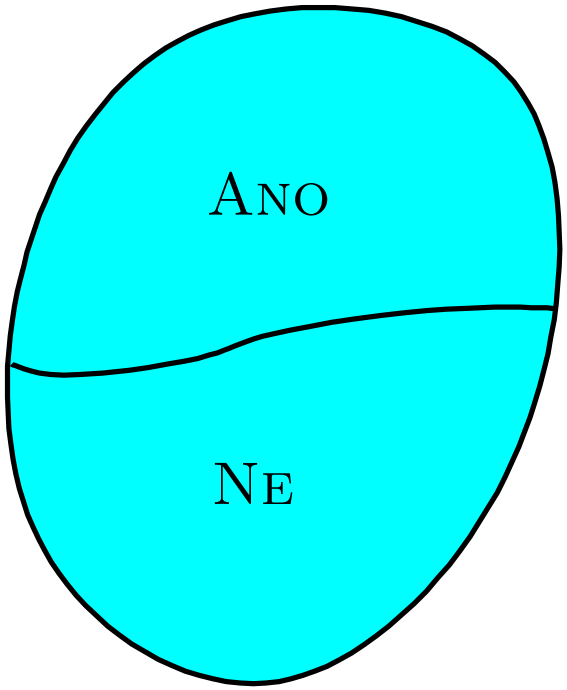
Pokud máme o nějakém (rozhodovacím) problému dokázáno, že je nerozhodnutelný, můžeme ukázat nerozhodnutelnost dalších problémů pomocí redukcí (převodů) mezi problémy.

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

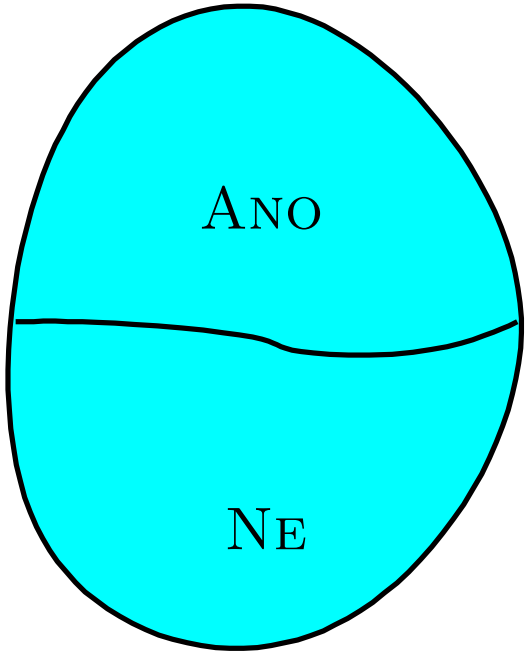
- Jako vstup může dostat libovolnou instanci problému P_1.
- K instanci problému P_1, kterou dostane jako vstup (označme ji w), vyprodukuje jako svůj výstup instanci problému P_2 (označme ji $Alg(w)$).
- Platí, že pro vstup w je v problému P_1 odpověď ANO právě tehdy, když pro vstup $Alg(w)$ je v problému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

Převody mezi problémy

vstupy problému P_1



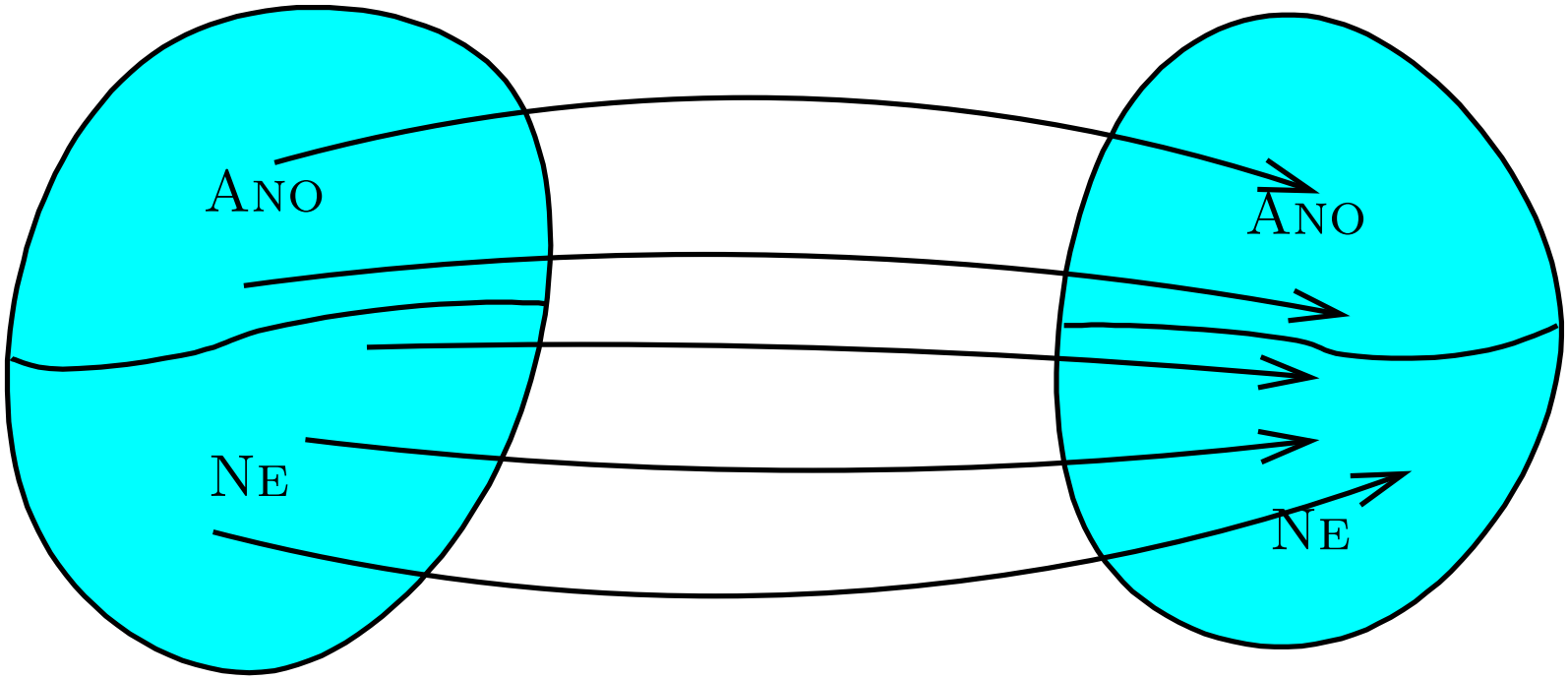
vstupy problému P_2



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 se Halting problem nejčastěji 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?

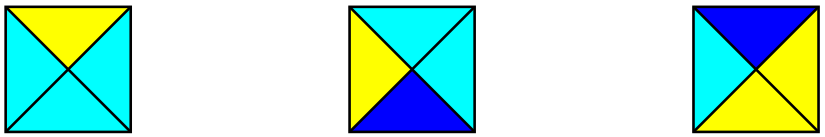
Poznámka: Také zde může být výhodné používat různé zjednodušující předpoklady, např.:

- 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:

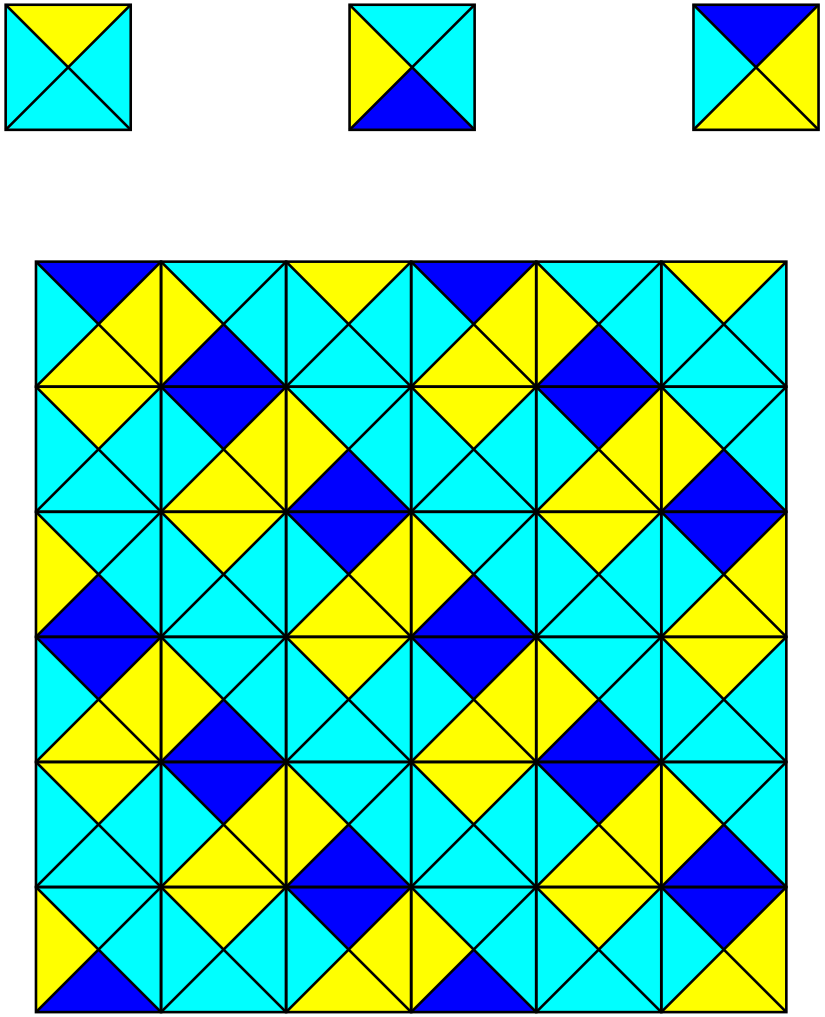


Otázka je, zda je možné použitím daných typů kachliček pokrýt celou nekonečnou rovinu tak, aby všechny kachličky spolu sousedily stejnými barvami.

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 ?

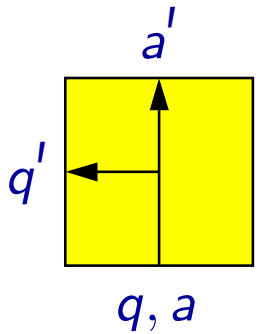
Popíšeme algoritmus, který:

- 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).

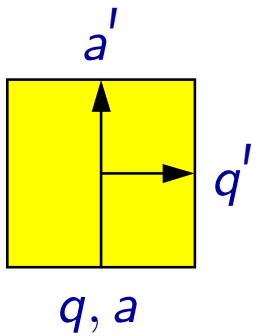
Kachličkování roviny

Algoritmus pro daný stroj $\mathcal{M}$ vyrobí kachličky následujícím způsobem (v následujícím popisu kachliček jsou místo barev použity šipky a značení pomocí prvků z množin Q, Γ a $(Q \times \Gamma)$ — nahrazení těchto šipek a značení barvami je přímočaré):

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

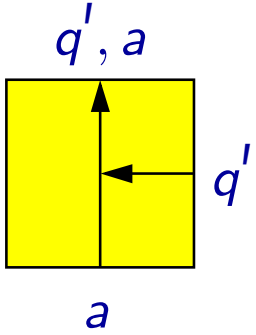


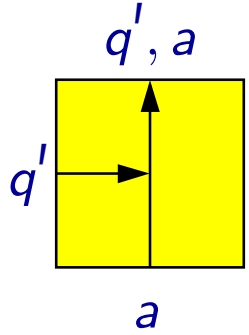
- Pro každé $q, q' \in Q$ a $a, a' \in \Gamma$, kde $\delta(q, a) = (q', a', +1)$, přidáme následující typ kachličky:



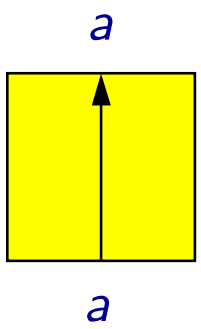
Kachličkování roviny

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



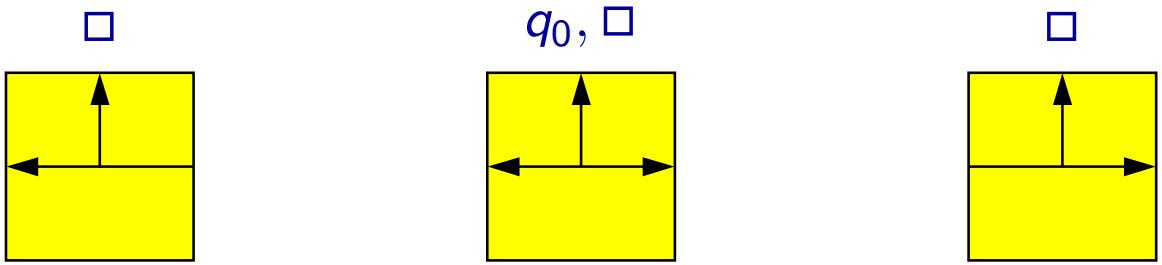


- Pro každé $a \in \Gamma$ přidáme následující typ kachliček:



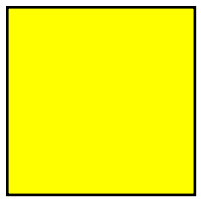
Kachličkování roviny

- 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.)

- 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.
- ...

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 stroje $\mathcal{M}$ 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:

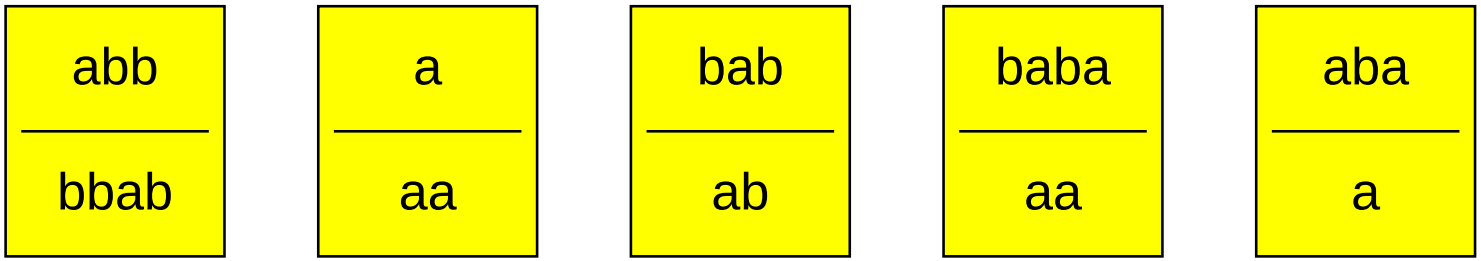
Vstup: Množina typů kachliček $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$.

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

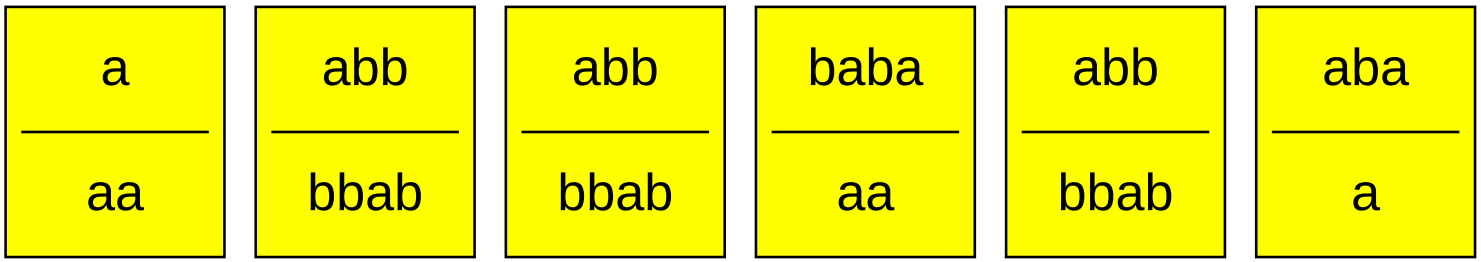
Důkaz je technicky komplikovanější, ale je také postaven na podobných myšlenkách, jaké byly uvedeny výše — tj. kódování výpočtu Turingova stroje, kdy je rovinu možné pokrýt jen v případě, že je výpočet daného stroje nekonečný.

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

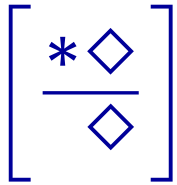
$$\left[\frac{*a_1 * a_2 * \cdots * a_k}{b_1 * b_2 * \cdots * b_\ell *}\right]$$

- 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 mohou být použity jako predikátové symboly pouze $=$ a $<$, 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:

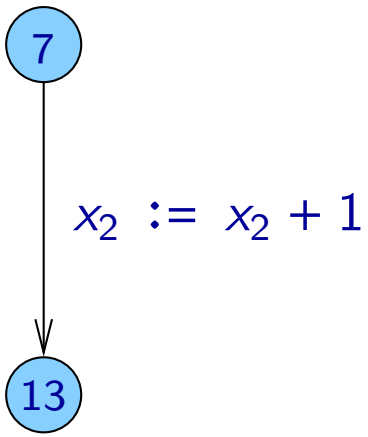
$$q = s$$

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:

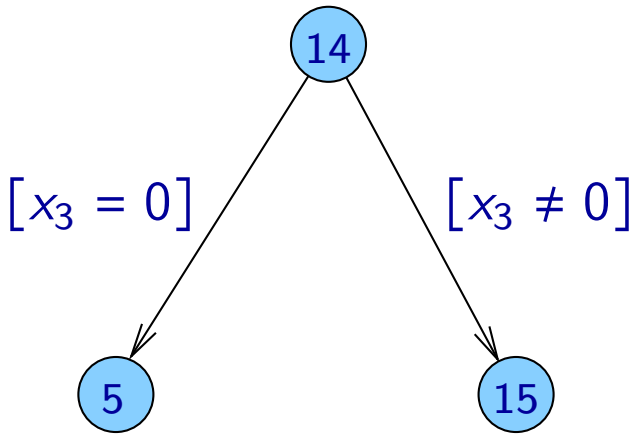
- $\textsc{Step}(q, v_1, v_2, v_3, q', v_1', v_2', v_3')$ — stroj $\mathcal{M}$ může přejít jedním krokem z konfigurace (q, v_1, v_2, v_3) do konfigurace (q', v_1', v_2', v_3')

  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 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 $d = p^i$

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

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

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

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.

Částečná rozhodnutelnost

Připomeňme, že daný rozhodnovací problém P je **částečně rozhodnutelný**, pokud 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 nikdy nezastaví.

Poznámka: Obecně je také možné, že pro některé vstupy, kde je správná odpověď NE, se může algoritmus $\mathcal{A}$ zastavit a dát tuto správnou odpověď.

Pro jednoduchost budeme ale předpokládat, že pro vstupy, kde je odpověď NE, se algoritmus $\mathcal{A}$ nikdy nezastaví, tj. že místo vrácení odpovědi NE vždy skočí do nekonečné smyčky.

Částečná rozhodnutelnost

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.

$$\text{odpověď pro } x \in In \text{ je ANO}$$
$$\Longleftrightarrow$$
$$\text{existuje nějaký } \textbf{skutečný svědek } w \in \mathcal{W}$$

Částečná rozhodnutelnost

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

- 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

Ještě jiná charakterizace částečně rozhodnutelných problémů vypadá takto:

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

- Ignoruje svůj vstup.
- 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

Pokud máme takový algoritmus $\mathcal{A}$ generující (potenciáně 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

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

- 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 vstup z množiny In.
  - Jakmile některý ze simulovaných výpočtů skončí s odpovědí ANO, algoritmus 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

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 **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.
- 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.