Úvod do teoretické informatiky

Zdeněk Sawa

Katedra informatiky, FEI,
Vysoká škola báňská – Technická universita Ostrava
17. listopadu 15, Ostrava-Poruba 708 33
Česká republika

20. května 2014

Přednášející

Jméno: doc. Ing. Zdeněk Sawa, Ph.D.

E-mail: zdenek.sawa@vsb.cz

Místnost: A1024

Web: http://www.cs.vsb.cz/sawa/uti

Na těchto stránkách najdete:

- Informace o předmětu
- Učební texty
- Slidy z přednášek
- Zadání příkladů na cvičení
- Aktuální informace
- Odkaz na stránku s animacemi

Požadavky

- **Zápočet** (22 bodů):
  - Zápočtová písemka (22 bodů) — bude se psát na cvičení

  Minimum pro získání zápočtu je 7 bodů.

  Možnost opravy za 14 bodů.

- **Zkouška** (78 bodů)
  - Písemná zkouška skládající se ze tří částí po 26 bodech, přičemž z každé části je nutné získat nejméně 10 bodů.

Algoritmy a problémy

Algoritmus — mechanický postup, jak něco spočítat (může být vykonáván počítačem)

Algoritmy slouží k řešení různých **problémů**.

Příklad algoritmického problému:

Vstup: Přirozená čísla x a y.

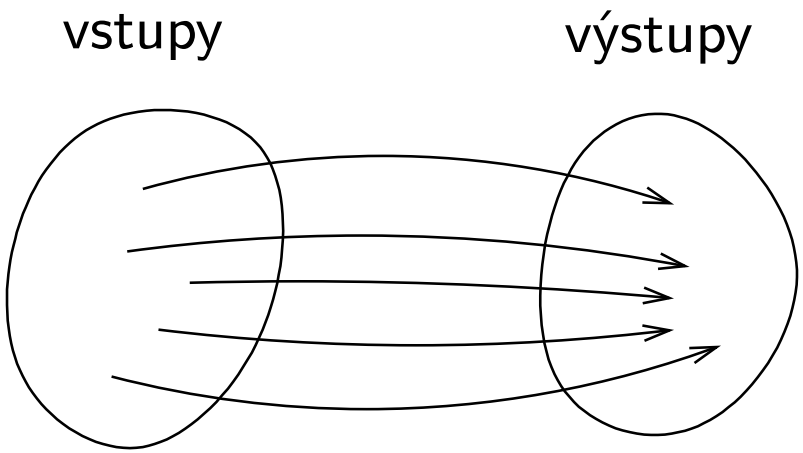
Výstup: Přirozené číslo z takové, že $z = x + y$.

Problémy

Problém

V zadání **problému** musí být určeno:

- co je množinou možných vstupů
- co je množinou možných výstupů
- jaký je vztah mezi vstupy a výstupy



Příklady problémů

Problém „Třídění“

Vstup: Sekvence prvků $a_1, a_2, \ldots, a_n$.

Výstup: Prvky sekvence $a_1, a_2, \ldots, a_n$ seřazené od nejmenšího po největší.

Příklad:

- Vstup: $8, 13, 3, 10, 1, 4$
- Výstup: $1, 3, 4, 8, 10, 13$

Poznámka: Konkrétní vstup nějakého problému se nazývá **instance** problému.

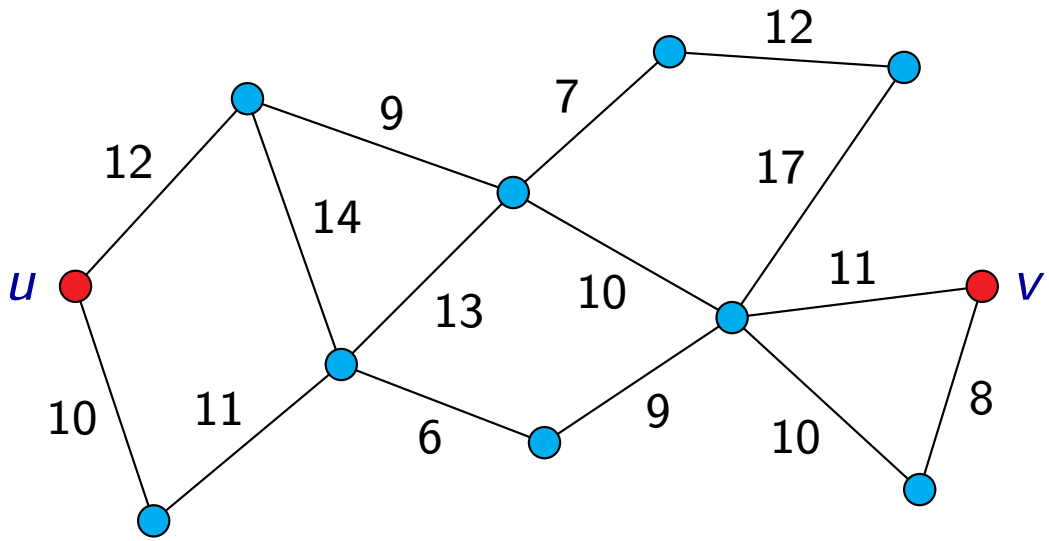
Příklady problémů

Problém „Hledání nejkratší cesty v (neorientovaném) grafu“

Vstup: Neorientovaný graf $G = (V, E)$ s ohodnocením hran, a dvojice vrcholů $u, v \in V$.

Výstup: Nejkratší cesta z vrcholu u do vrcholu v.

Příklad:



Algoritmy a problémy

Algoritmus **řeší** daný problém pokud:

- Se pro každý vstup po konečném počtu kroků zastaví.
- Pro každý vstup vydá správný výstup.

Korektnost algoritmu — ověření toho, že daný algoritmus skutečně řeší daný problém

Výpočetní složitost algoritmu:

- **časová složitost** — jak závisí doba výpočtu na velikosti vstupu
- **paměťová** (nebo též **prostorová**) **složitost** — jak závisí množství použité paměti na velikosti vstupu

Algoritmy a problémy

Teoretická informatika se překrývá se s mnoha dalšími oblastmi matematiky a informatiky:

- teorie grafů
- teorie čísel
- kryptografie
- výpočetní geometrie
- vyhledávání v textu, komprese dat
- teorie her
- ...

Další příklady problémů

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Výstup: ANO pokud je n prvočíslo, NE v opačném případě.

Poznámka: Přirozené číslo n je **prvočíslo**, pokud je větší než 1 a je dělitelné beze zbytku pouze čísly 1 a n.

Prvních několik prvočísel: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Rozhodovací problémy

Problémům, kde množina výstupů je $\{\text{Ano}, \text{Ne}\}$ se říká **rozhodovací problémy**.

Rozhodovací problémy jsou většinou specifikovány tak, že místo popisu toho, co je výstupem, je uvedena otázku.

Příklad:

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Otázka: Je n prvočíslo?

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.

Teorie vyčíslitelnosti — oblast teoretické informatiky, která se zabývá zkoumáním toho, které problémy jsou a které nejsou algoritmicky řešitelné.

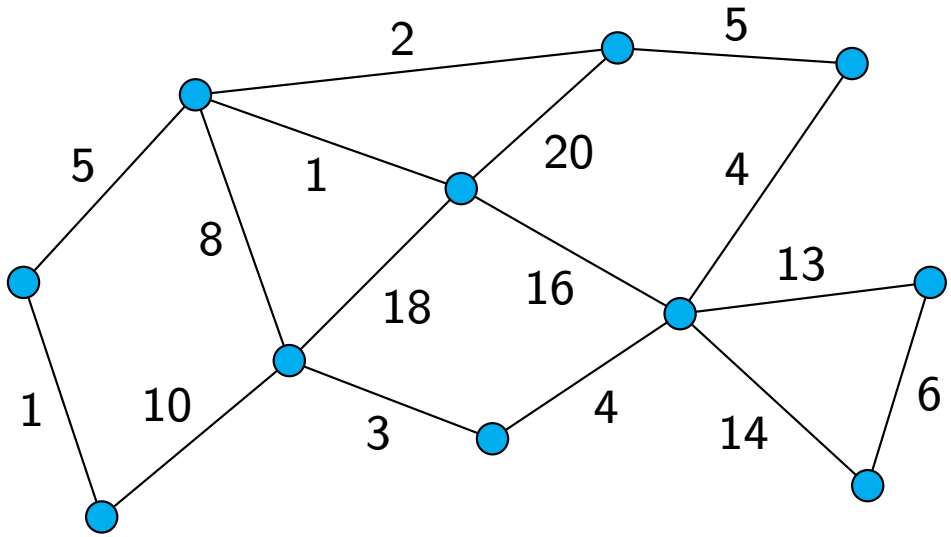
Teorie složitosti

Řada problémů je algoritmicky řešitelných, ale neexistují (nebo nejsou známy) efektivní algoritmy, které by je řešily:

TSP - Problém obchodního cestujícího

Vstup: Neorientovaný graf G s hranami ohodnocenými přirozenými čísly.

Výstup: Nejkratší uzavřená cesta, která projde všemi vrcholy a skončí v tom vrcholu, kde začíná.



Teoretická informatika

Některé další oblasti teoretické informatiky:

- teorie složitosti
- paralelní a distribuované algoritmy
- výpočetní modely

Teorie formálních jazyků

Oblast teoretické informatiky zabývajícími se otázkami týkajícími se **syntaxe**.

- **Jazyk** — množina slov
- **Slovo** — sekvence symbolů z určité abecedy
- **Abeceda** — množina **symbolů** (nebo též **znaků**)

Prostředky používané pro popis jazyků:

- gramatiky
- regulární výrazy
- automaty

Logika

Logika — obor zabývající se otázkami správného vyvozování a argumentace

- formulazace tvrzení přirozeného jazyka pomocí **formulí**
- zkoumání, kdy závěr vyplývá z daných předpokladů
- otázky týkající se důkazů a dokazatelnosti
- souvisí se zkoumáním základů matematiky a s teorií množin

Dva nejdůležitější typy logik:

- výroková logika
- predikátová logika

Výroková logika

Logické vyvozování

- *Jestliže měl vlak zpoždění a na nádraží nebyly taxíky, tak Honza přišel pozdě do práce.*
- *Honza nepřišel do práce pozdě.*
- *Vlak měl zpoždění.*

- *Na nádraží byly taxíky.*

- *Jestliže pršelo a Jana neměla s sebou deštník, tak zmokla.*
- *Jana nezmokla.*
- *Pršelo.*

- *Jana měla s sebou deštník.*

Logické vyvozování

p	Vlak měl zpoždění.	Pršelo.
q	Na nádraží byly taxíky.	Jana měla s sebou deštník.
r	Honza přišel pozdě do práce.	Jana zmokla.

Jestliže p a ne q, tak r.
Ne r.
p.

---

q.

Výroky

Příklady výroků:

- „*Jana zmokla.*“
- „*Jestliže pršelo a Jana neměla s sebou deštník, tak zmokla.*“
- „*Paříž je hlavním městem Japonska.*“
- „*Existuje nekonečně mnoho prvočísel.*“
- „$1 + 1 = 3$“
- „*Číslo* $\sqrt{2}$ *je iracionální.*“

Příklady formulací, které nejsou výroky:

- „*Součet čísel* 2 *a* 5*.*“
- „*V kolik hodin jede dneska rychlík z Ostravy do Prahy.*“
- „*Vzdálenost Země od Slunce.*“

Výroky

Příklady formulací, které nejsou přímo výroky, ale ze kterých dostaneme výroky dosazením konkrétních hodnot za proměnné:

- „$x > 5$“
- „$x + y = z$“
- „x *je největším prvkem množiny* A“

Toto ovšem jsou výroky:

- „*Existuje přirozené číslo* x *takové, že* $x > 5$.“
- „*Pro každé přirozené číslo* x *platí* $x > 5$.“

Logické spojky

Atomický výrok — nedá se rozložit na žádné menší výroky

„Jana zmokla.“

Složený výrok — složen z menších jednodušších výroků

„Jestliže pršelo a Jana neměla s sebou deštník, tak zmokla.“

Složeno z výroků:

- *„Pršelo.“*
- *„Jana měla s sebou deštník.“*
- *„Jana zmokla.“*

Logické spojky

Reprezentace výroků pomocí **formulí**:

Symbol	Log. spojka	Příklad použití	Neformální význam
$\neg$	negace	$\neg p$	„*není pravda* p“
$\wedge$	konjunkce	$p \wedge q$	„p *a* q“
$\vee$	disjunkce	$p \vee q$	„p *nebo* q“
$\rightarrow$	implikace	$p \rightarrow q$	„*jestliže* p, *pak* q“
$\leftrightarrow$	ekvivalence	$p \leftrightarrow q$	„p *právě tehdy, když* q“

Atomické výroky — p, q, r, ...
(případně s indexy — p_0, p_1, p_2, ...)

Logické spojky

„Jestliže pršelo a Jana neměla s sebou deštník, tak zmokla.“

Zápis pomocí formule:

$$(p \wedge \neg q) \rightarrow r$$

Atomické výroky:

- p — *„Pršelo.“*
- q — *„Jana měla s sebou deštník.“*
- r — *„Jana zmokla.“*

Pravdivostní hodnoty

pravda	nepravda
1	0
P	N
true	false
T	F
ano	ne
yes	no

Pro označení pravdivostních hodnot budeme používat 0 a 1.

Pravdivostní hodnoty se též označují jako hodnoty **booleovské**.

Negace

Negací výroku φ je výrok „*není pravda, že* φ“. Například negací výroku

„*číslo* 5 *je prvočíslo*“

je výrok

„*není pravda, že číslo* 5 *je prvočíslo*“

nebo

„*číslo* 5 *není prvočíslo*“.

Ve formulích se negace označuje symbolem “$\neg$”.

Formální zápis: $\neg\varphi$ příklad: $\neg p$

φ	$\neg\varphi$
0	1
1	0

Negace

Příklad: Negací výroku

„Karel je ze všech žáků ve třídě nejvyšší“

je třeba výrok

„Karel není ze všech žáků ve třídě nejvyšší“

nebo

„není pravda, že Karel je ze všech žáků ve třídě nejvyšší“.

Ne ovšem výrok

„Karel je ze všech žáků ve třídě nejmenší“.

Negace

Příklad:

„Není pravda, že číslo 5 *není kladné.“*

lze formalizovat jako

$\neg\neg p$ nebo $\neg(\neg p)$

Podobně

„není pravda, že není pravda, že číslo 5 *není kladné“*

je možné zapsat

$\neg\neg\neg p$ nebo $\neg(\neg(\neg p))$

p — *„číslo* 5 *je kladné“*

Konjunkce

Konjunkcí výroků φ a ψ je výrok „φ *a* ψ“.

Příklad: Konjunkcí výroků *„Kodaň je hlavním městem Dánska“* a „$2 + 2 = 4$“ je výrok

„Kodaň je hlavním městem Dánska a $2 + 2 = 4$*.“*

Ve formulích se konjunkce označuje pomocí symbolu „$\wedge$“.

$$p \wedge q$$

- p — *„Kodaň je hlavním městem Dánska“*
- q — „$2 + 2 = 4$“

Konjunkce

φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1

Příklady nepravdivých výroků:

- *„Helsinky jsou hlavním městem Itálie a Karlova univerzita byla založena v roce 1348.“*
- *„Asie je světadíl s největší rozlohou a* $3 + 5 = 14$*.“*
- *„Existuje jen konečně mnoho prvočísel a Plzeň je hlavním městem USA.“*

Konjunkce

Příklad:

„chtěl jsem přijet, ale ujel mi vlak“

lze formalizovat jako $p \wedge q$.

Příklad:

„číslo 12 je dělitelné čísly 3 a 4“

lze formalizovat jako $p \wedge q$, kde

- p — *„číslo 12 je dělitelné číslem 3“*
- q — *„číslo 12 je dělitelné číslem 4“*

Konjunkce

Příklady výroků, kde „a“ **nemá** význam konjunkce:

- *„Levý a pravý břeh řeky byly spojeny mostem.“*
- *„Číslo 3 je největším společným dělitelem čísel 12 a 15.“*
- *„Bod A leží na průsečíku přímek p_1 a p_2.“*

Konjunkce

Výrok „*ani* φ, *ani* ψ“ říká to samé, co

„*není pravda* φ *a není pravda* ψ“.

Je tedy možné ho formalizovat jako

$$\neg\varphi \wedge \neg\psi$$

Příklad: Výrok „*přímka* p *neprochází ani bodem* A *ani bodem* B“ je možné formalizovat jako

$$\neg r \wedge \neg s\,,$$

kde

- r — „*přímka* p *prochází bodem* A“
- s — „*přímka* p *prochází bodem* B“

Disjunkce

Disjunkcí výroků φ a ψ je výrok „φ *nebo* ψ“.

Příklad: Disjunkcí výroků *„velryby patří mezi savce“* a *„ČR leží v mírném podnebném pásu“* je výrok

„velryby patří mezi savce nebo ČR leží v mírném podnebném pásu“.

Ve formulích se disjunkce označuje symbolem “$\vee$”.

$$p \vee q$$

- p — *„velryby patří mezi savce“*
- q — *„ČR leží v mírném podnebném pásu“*

Disjunkce

Disjunkce je „*nebo*“ v **nevylučujícím** smyslu.

φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1

Implikace

Implikace — „*jestliže* φ, *pak* ψ“

- φ — předpoklad
- ψ — závěr

Příklad:

> „ *Jestliže se Petr dobře připravil na zkoušku, pak z této zkoušky dostal dobrou známku.*“

Implikace se označuje symbolem “$\rightarrow$”.

$$p \rightarrow q$$

- p — „*Petr se dobře připravil na zkoušku*“
- q — „*Petr dostal z této zkoušky dobrou známku*“

Implikace

φ	ψ	$\varphi \to \psi$
0	0	1
0	1	1
1	0	0
1	1	1

Poznámka: Formule $p \to q$ je pravdivá právě v těch případech, kdy je pravdivá formule

$$\neg p \vee q\,.$$

Implikace

Implikace **nevyjařuje** příčinnou souvislost.

Příklad:

- „*Jestliže je Washington hlavním městem USA, tak* $1+1=2$.“
- „*Jestliže je Washington hlavním městem USA, tak* $1+1=3$.“
- „*Jestliže je Tokio hlavním městem USA, tak* $1+1=2$.“
- „*Jestliže je Tokio hlavním městem USA, tak* $1+1=3$.“

Implikace

Příklady různých způsobů vyjádření tvrzení $p \longrightarrow q$ v přirozené řeči:

- „*pokud* p, *pak* q“
- „*když* p, *tak* q“
- „q, *pokud* p“
- „*z* p *plyne* q“
- „*za předpokladu* p *platí* q“
- „p *implikuje* q“
- „q *platí tehdy, když platí* p“
- „p *platí jen tehdy, když platí* q“
- „p *je postačující podmínka pro* q“
- „q *je nutná podmínka pro* p“

Implikace

Pokud platí $\varphi \to \psi$ a zároveň platí φ, je možné z toho vyvodit, že platí i ψ.

Příklad: Pokud platí

- *„jestliže je dnes úterý, pak je zítra středa“*
- *„dnes je úterý“*

lze z toho vyvodit, že platí

- *„zítra je středa“*

Ekvivalence

Ekvivalence — „φ *právě tehdy, když* ψ“

Příklad:

> „*Trojúhelník* ABC *má všechny tři strany stejně dlouhé právě tehdy, když má všechny tři úhly stejně velké.*“

Logická spojka ekvivalence se označuje symbolem “$\leftrightarrow$”

$$p \leftrightarrow q$$

- p — „*trojúhelník* ABC *má všechny tři strany stejně dlouhé*“
- q — „*trojúhelník* ABC *má všechny tři úhly stejně velké*“

Ekvivalence

φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1

Poznámka: Formule $p \leftrightarrow q$ říká v podstatě to samé co formule

$$(p \rightarrow q) \wedge (q \rightarrow p)$$

Ekvivalence

Alternativní způsoby vyjádření ekvivalence $p \leftrightarrow q$ v přirozené řeči:

- „p *tehdy a jen tehdy když* q“
- „p *je nutnou a postačující podmínkou pro to, aby platilo* q“

Ekvivalence se často používá v **definicích** nových pojmů:

Příklad:

- *„Trojúhelník je rovnoramenný právě tehdy, když alespoň dvě jeho strany jsou stejně dlouhé.“*
- *„Trojúhelník je rovnoramenný, jestliže alespoň dvě jeho strany jsou stejně dlouhé.“*

Formule výrokové logiky

- **Syntaxe** — jak vypadají formule výrokové logiky
- **Sémantika** — přiřazuje formulím a jednotlivým symbolům, které se v nich vyskytují, přesně definovaný význam

Syntaxe formulí výrokové logiky

Formule — posloupnosti symbolů z určité **abecedy**:

- **atomické výroky** — například symboly "p", "q", "r", apod.
- **logické spojky** — symboly "$\neg$", "$\wedge$", "$\vee$" "$\rightarrow$" a "$\leftrightarrow$"
- **závorky** — symboly "$($" a "$)$"

Ne každá posloupnost těchto symbolů je formulí.

Například toto formule není:

$$\wedge\vee)p\neg((\neg$$

Syntaxe formulí výrokové logiky

Definice

Dobře utvořené **formule výrokové logiky** jsou posloupnosti symbolů vytvořené podle následujících tří pravidel:

1. Jestliže p je atomický výrok, pak p je dobře utvořená formule.
2. Jestliže φ a ψ jsou dobře utvořené formule, pak i $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$ a $(\varphi \leftrightarrow \psi)$ jsou dobře utvořené formule.
3. Neexistují žádné další dobře utvořené formule než ty, které jsou vytvořené pomocí předchozích dvou pravidel.

Syntaxe formulí výrokové logiky

Příklady dobře utvořených formulí:

- q
- $(\neg q)$
- r
- $((\neg q) \rightarrow r)$
- p
- $(p \leftrightarrow r)$
- $(\neg(p \leftrightarrow r))$
- $(((\neg q) \rightarrow r) \wedge (\neg(p \leftrightarrow r)))$

Příklad sekvence symbolů, která není dobře utvořenou formulí:

- $(p \wedge \vee q)$

Syntaxe formulí výrokové logiky

Formule ψ je **podformulí** formule φ, jestliže platí alespoň jedna z následujících možností:

- Formule ψ je stejná jako formule φ (tj. jedná se o jednu a tutéž formuli).
- Pokud je formule φ tvaru $(\neg\chi)$, tak ψ je podformulí formule χ.
- Pokud je formule φ tvaru $(\chi_1 \wedge \chi_2)$, $(\chi_1 \vee \chi_2)$, $(\chi_1 \rightarrow \chi_2)$ nebo $(\chi_1 \leftrightarrow \chi_2)$, tak ψ je podformulí formule χ_1 nebo podformulí formule χ_2.

Příklad: Podformule formule $((\neg(p \wedge q)) \leftrightarrow r)$:

$$p \qquad q \qquad r \qquad (p \wedge q) \qquad (\neg(p \wedge q)) \qquad ((\neg(p \wedge q)) \leftrightarrow r)$$

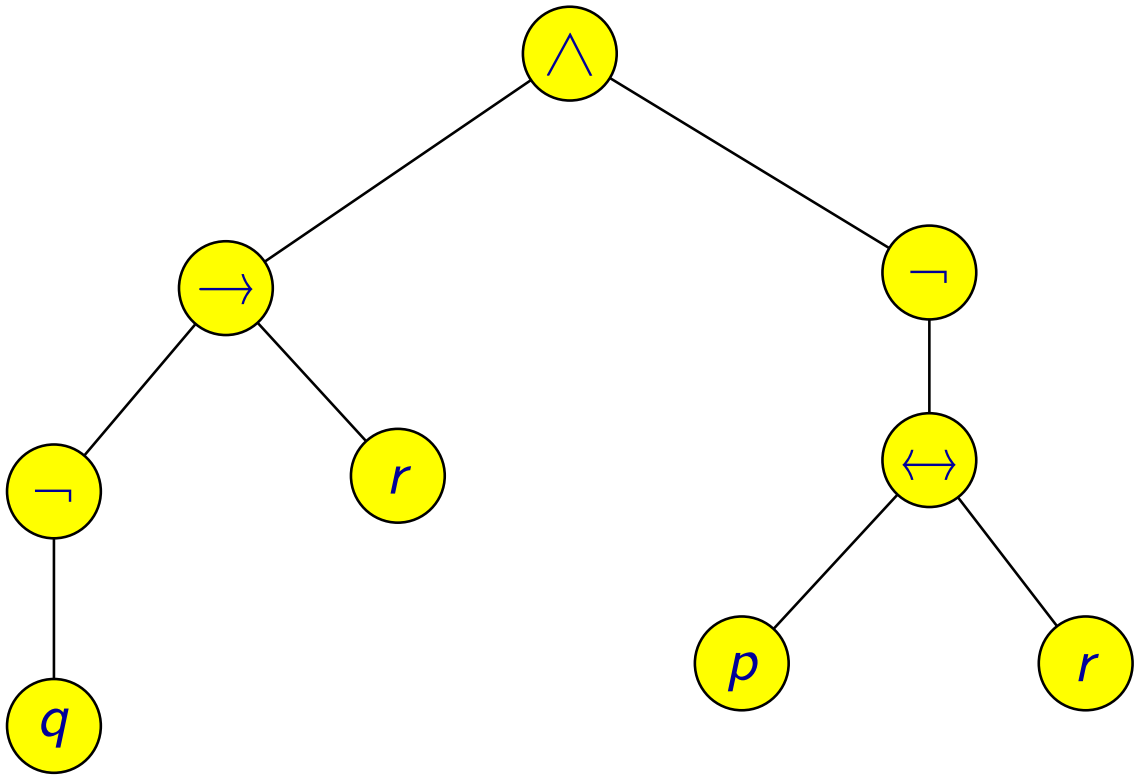
Syntaxe formulí výrokové logiky

Alternativní symboly pro logické spojky:

Log. spojka	Symbol	Alternativní symboly
negace	$\neg$	$\sim$
konjunkce	$\wedge$	$\&$
implikace	$\rightarrow$	$\Rightarrow$, $\supset$
ekvivalence	$\leftrightarrow$	$\Leftrightarrow$, $\equiv$

Syntaxe formulí výrokové logiky

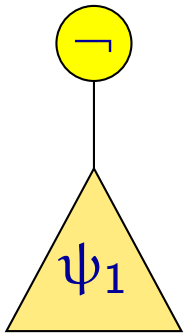
Abstraktní syntaktický strom formule $(((\neg q) \rightarrow r) \wedge (\neg(p \leftrightarrow r)))$:



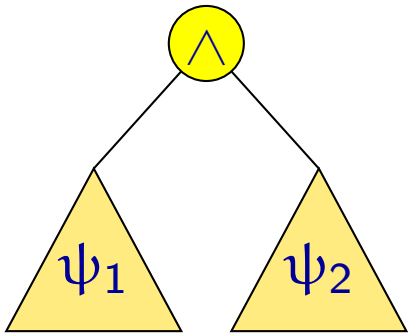
Syntaxe formulí výrokové logiky

p

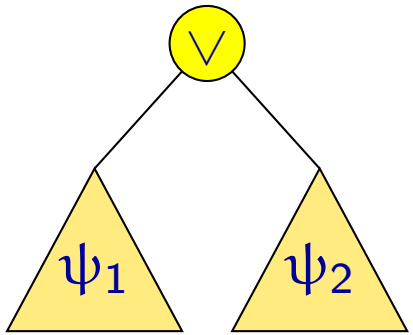
$(\neg\psi_1)$



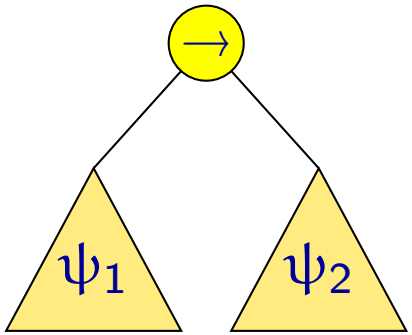
$(\psi_1 \wedge \psi_2)$



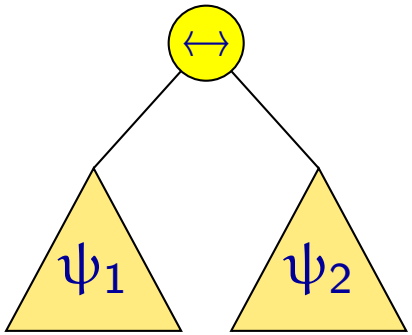
$(\psi_1 \vee \psi_2)$



$(\psi_1 \rightarrow \psi_2)$



$(\psi_1 \leftrightarrow \psi_2)$



Syntaxe formulí výrokové logiky

Arita logických spojek:

- **unární** spojka (arita 1): $\neg$
- **binární** spojky (arita 2): $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

Syntaxe formulí výrokové logiky

Konvence pro vypouštění závorek:

- Vnější závorky je možno vypustit.
- Priorita logických spojek (od nejvyšší po nejnižší):

$$\neg \qquad \wedge \qquad \vee \qquad \rightarrow \qquad \leftrightarrow$$

- Místo $\neg(\neg\varphi)$ je možno psát $\neg\neg\varphi$.

Příklad: Místo $((\neg p) \wedge (r \rightarrow (q \vee s)))$ je možno psát

$$\neg p \wedge (r \rightarrow q \vee s)$$

Poznámka: Další konvence budou uvedeny později.

Sémantika výrokové logiky

At — množina atomických výroků

Například

- $At = \{p, q, r\}$, nebo
- $At = \{p_0, p_1, p_2, \ldots\}$

Definice

Pravdivostní ohodnocení je přiřazení pravdivostních hodnot (tj. hodnot z množiny $\{0, 1\}$) všem atomickým výrokům z množiny At.
(Formálně je možné pravdivostní ohodnocení definovat jako funkci $v : At \to \{0, 1\}$.)

Příklad: Pravdivostní ohodnocení v pro $At = \{p, q, r\}$, kde

$$v(p) = 1 \qquad v(q) = 0 \qquad v(r) = 1$$

Sémantika výrokové logiky

Pokud je množina At konečná a obsahuje n atomických výroků, existuje celkem 2^n pravdivostních ohodnocení.

Příklad: $At = \{p, q, r\}$

$$
\begin{array}{lllll}
v_0\colon & v_0(p) = 0, & v_0(q) = 0, & v_0(r) = 0 \\
v_1\colon & v_1(p) = 0, & v_1(q) = 0, & v_1(r) = 1 \\
v_2\colon & v_2(p) = 0, & v_2(q) = 1, & v_2(r) = 0 \\
v_3\colon & v_3(p) = 0, & v_3(q) = 1, & v_3(r) = 1 \\
v_4\colon & v_4(p) = 1, & v_4(q) = 0, & v_4(r) = 0 \\
v_5\colon & v_5(p) = 1, & v_5(q) = 0, & v_5(r) = 1 \\
v_6\colon & v_6(p) = 1, & v_6(q) = 1, & v_6(r) = 0 \\
v_7\colon & v_7(p) = 1, & v_7(q) = 1, & v_7(r) = 1
\end{array}
$$

Sémantika výrokové logiky

Formule φ má při pravdivostním ohodnocení v pravdivostní hodnotu 1:

$$v \models \varphi$$

Formule φ má při pravdivostním ohodnocení v pravdivostní hodnotu 0:

$$v \not\models \varphi$$

Sémantika výrokové logiky

Definice

Pravdivostní hodnoty formulí výrokové logiky při daném pravdivostním ohodnocení v jsou definovány následujícím způsobem:

- Pro atomický výrok p platí $v \models p$ právě tehdy, když $v(p) = 1$. (Pokud je tedy $v(p) = 0$, tak $v \not\models p$.)
- $v \models \neg\varphi$ právě tehdy, když $v \not\models \varphi$.
- $v \models \varphi \wedge \psi$ právě tehdy, když $v \models \varphi$ a $v \models \psi$.
- $v \models \varphi \vee \psi$ právě tehdy, když $v \models \varphi$ nebo $v \models \psi$.
- $v \models \varphi \rightarrow \psi$ právě tehdy, když $v \not\models \varphi$ nebo $v \models \psi$.
- $v \models \varphi \leftrightarrow \psi$ právě tehdy, když $v \models \varphi$ a $v \models \psi$, nebo když $v \not\models \varphi$ a $v \not\models \psi$.

Sémantika výrokové logiky

φ	$\neg\varphi$
0	1
1	0

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \rightarrow \psi$	$\varphi \leftrightarrow \psi$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Sémantika výrokové logiky

Příklad: $At = \{p, q, r\}$
pravdivostní ohodnocení v, kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

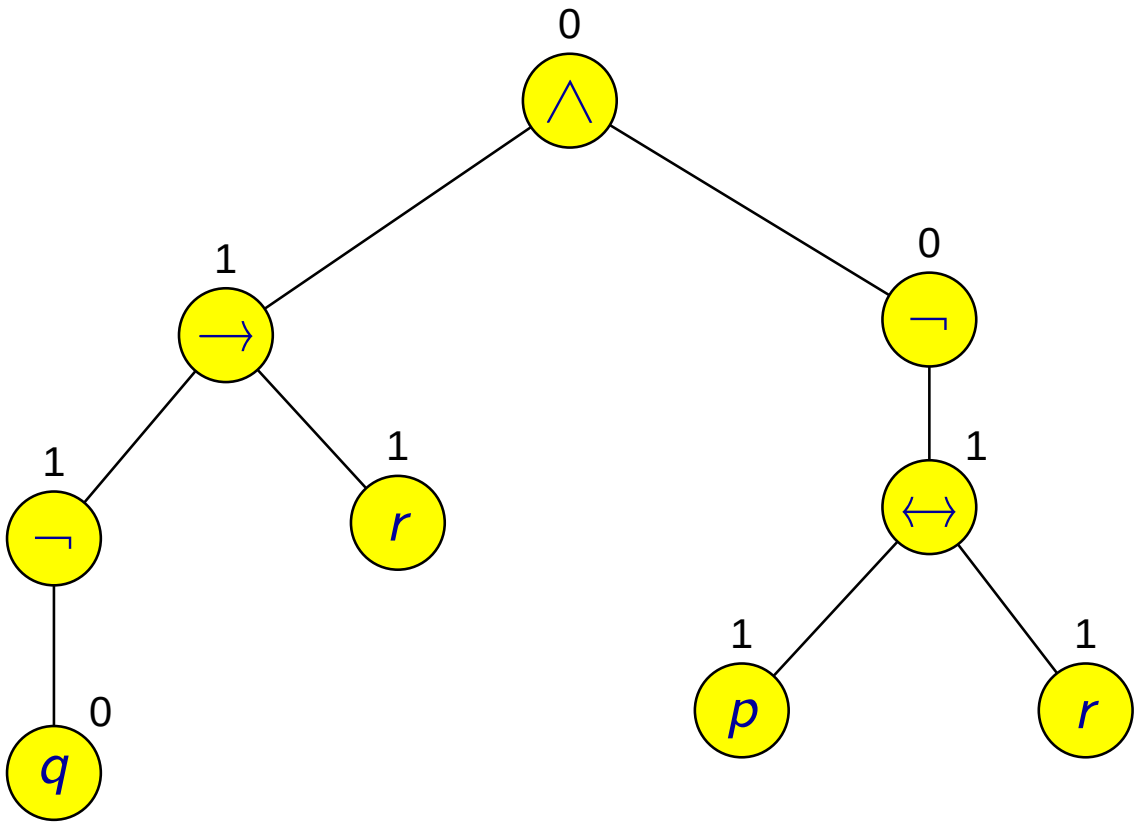
- $v \not\models q$
- $v \models \neg q$
- $v \models r$
- $v \models \neg q \rightarrow r$
- $v \models p$
- $v \models p \leftrightarrow r$
- $v \not\models \neg(p \leftrightarrow r)$
- $v \not\models (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$

p	q	r	$\neg q$	$\neg q \rightarrow r$	$p \leftrightarrow r$	$\neg(p \leftrightarrow r)$	φ
1	0	1	1	1	1	0	0

$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$

Sémantika výrokové logiky

$(\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$



Sémantika výrokové logiky

$\varphi := (\neg q \to r) \wedge \neg(p \leftrightarrow r)$

p	q	r	$\neg q$	$\neg q \to r$	$p \leftrightarrow r$	$\neg(p \leftrightarrow r)$	φ
0	0	0	1	0	1	0	0
0	0	1	1	1	0	1	1
0	1	0	0	1	1	0	0
0	1	1	0	1	0	1	1
1	0	0	1	0	0	1	0
1	0	1	1	1	1	0	0
1	1	0	0	1	0	1	1
1	1	1	0	1	1	0	0

Ohodnocení, při kterých je formule pravdivá se nazývají **modely**:

$$\begin{array}{lll} v_1\colon & v_1(p) = 0, & v_1(q) = 0, \quad v_1(r) = 1, \\ v_3\colon & v_3(p) = 0, & v_3(q) = 1, \quad v_3(r) = 1, \\ v_6\colon & v_6(p) = 1, & v_6(q) = 1, \quad v_6(r) = 0, \end{array}$$

Tautologie

Definice

Formule φ je **tautologie**, jestliže pro každé pravdivostní ohodnocení v platí $v \models \varphi$ (tj. pokud φ je pravdivá při každém pravdivostním ohodnocení).

Příklad: *„Jestliže venku prší, tak venku prší.“*

$$p \rightarrow p$$

Příklad: *„Dnes je pátek nebo dnes není pátek.“*

$$q \vee \neg q$$

Tautologie

Příklad komplikovanější tautologie:

$$(p \to q) \to ((p \to \neg q) \to \neg p)$$

p	q	$p \to q$	$\neg q$	$p \to \neg q$	$\neg p$	$(p \to \neg q) \to \neg p$	φ
0	0	1	1	1	1	1	1
0	1	1	0	1	1	1	1
1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	1

Tautologie

Důležité jsou zejména tautologie tvaru $\varphi \to \psi$ nebo $\varphi \leftrightarrow \psi$
— dají se použít pro logické vyvozování:

- Pokud platí $\varphi \to \psi$ a zároveň platí φ, musí platit i ψ.

  Speciálně, pokud $\varphi \to \psi$ je tautologie, z platnosti φ se dá vyvodit, že platí i ψ.

  Příklad: $(p \wedge q) \to p$ je tautologie.

  Pokud platí $p \wedge q$, tak platí i p.

  Příklad: $(p \to q) \to (\neg q \to \neg p)$ je tautologie.

  Pokud platí $p \to q$ a zároveň platí $\neg q$, tak platí $\neg p$.

Tautologie

- Pokud platí $\varphi \leftrightarrow \psi$ a zároveň platí φ, musí platit i ψ.

  Podobně, pokud platí $\varphi \leftrightarrow \psi$ a zároveň platí ψ, musí platit i φ.

  Příklad: $(\neg p \rightarrow q) \leftrightarrow (q \vee p)$ je tautologie.

  - Pokud platí $\neg p \rightarrow q$, tak musí platit i $q \vee p$.
  - Pokud platí $q \vee p$, tak musí platit i $\neg p \rightarrow q$.

Tautologie

Pokud v tautologii φ nahradíme jednotlivé atomické výroky libovolnými formulemi, dostaneme opět tautologii.

Příklad: Formule $p \to (p \vee q)$ je tautologie.

Pro libovolné formule ψ a χ proto platí, že

$$\psi \to (\psi \vee \chi)$$

je tautologie.

Náhrada atomických výroků:

- p nahradíme $q \vee \neg(r \to \neg s)$
- q nahradíme $\neg\neg(q \leftrightarrow p)$

Dostaneme tautologii

$$(q \vee \neg(r \to \neg s)) \to ((q \vee \neg(r \to \neg s)) \vee \neg\neg(q \leftrightarrow p))$$

Kontradikce

Definice

Formule φ je **kontradikce**, jestliže pro každé pravdivostní ohodnocení v platí $v \not\models \varphi$ (tj. pokud φ je při každém pravdivostním ohodnocení nepravdivá).

Příklad: *„Dnes je středa a dnes není středa.“*

$$p \wedge \neg p$$

- φ je tautologie právě tehdy, když $\neg\varphi$ je kontradikce
- φ je kontradikce právě tehdy, když $\neg\varphi$ je tautologie

Splnitelné formule

Definice

Formule φ je **splnitelná**, jestliže existuje alespoň jedno pravdivostní ohodnocení v, pro které je $v \models \varphi$.

- Formule je splnitelná právě tehdy, když není kontradikcí.
- Každá tautologie je splnitelná, ale ne každá splnitelná formule je tautologie.

Příklad: Formule, která je splnitelná, ale není tautologie:

$$(p \vee q) \rightarrow p$$

- Například při ohodnocení v_1, kde $v_1(p) = 1$ a $v_1(q) = 0$, je pravdivá.
- Při ohodnocení v_2, kde $v_2(p) = 0$ a $v_2(q) = 1$, je nepravdivá.

Splnitelné formule

- φ je tautologie právě tehdy, když $\neg\varphi$ není splnitelná
- φ je splnitelná právě tehdy, když $\neg\varphi$ není tautologie

- **Splnitelná formule**:
  - Abychom ukázali, že formule **je** splnitelná, stačí najít ohodnocení, při kterém je pravdivá.
  - Abychom ukázali, že formule **není** splnitelná, je třeba ukázat, že neexistuje žádné ohodnocení, při kterém by byla pravdivá.

Tautologie a kontradikce

- **Tautologie**:
  - Abychom ukázali, že formule **není** tautologie, stačí najít ohodnocení, při kterém není pravdivá.
  - Abychom ukázali, že formule **je** tautologie, je třeba ukázat, že neexistuje žádné ohodnocení, při kterém není pravdivá.

- **Kontradikce**:
  - Abychom ukázali, že formule **není** kontradikce, stačí najít ohodnocení, při kterém je pravdivá.
  - Abychom ukázali, že formule **je** kontradikce, je třeba ukázat, že neexistuje žádné ohodnocení, při kterém by byla pravdivá.

Sémantický spor

Při zdůvodňování toho, že formule φ je/není tautogie (resp. kontradikce, splnitelná) můžeme použít tabulkovou metodu.

Většinou není potřeba vytvářet celou tabulku, ale stačí se soutředit na „zajímavé“ případy.

- Můžeme si nakreslit syntaktický strom dané formule a zkusit přiřadit vrcholům hodnoty 0 a 1.

Například pro zjištění toho, zda formule je/není tautologie:

- Nejprve přiřadíme kořeni hodnotu 0.
- Postupně přiřazujeme dalším vrcholům hodnoty, které jsou vynuceny dříve přiřazenými hodnotami.
- Pokud se podaří celý strom konzistentně ohodotit, máme ohodnocení, při kterém formule není pravdivá.
- Pokud zjistíme, že žádné takové ohodnocení nemůže existovat, daná formule je tautologie.

Sémantický spor

- Stejné podstromy musí být ohodnoceny stejně — stejné podformule musí mít stejné pravdivostní hodnoty.
- Pokud má být nějaký vrchol ohodnocen 0 i 1, máme **spor** — takové ohodnocení nemůže existovat.
- Někdy je nutné se vracet a zkoušet více možností přiřazení — případy, kdy aktuálně přiřazené hodnoty nevynucují jednoznačné přiřazení hodnoty nějakému dosud neohodnocenému vrcholu.

Sémantický spor

Příklad:

- $(p \to (q \vee r)) \vee (p \to r)$ (není tautologie)

  ⟨*řešení na tabuli*⟩

- $(p \leftrightarrow (\neg q \vee r)) \to (\neg p \to q)$ (je tautologie)

  ⟨*řešení na tabuli*⟩

- $((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$ (není tautologie)

  ⟨*řešení na tabuli*⟩

Ekvivalence formulí

Definice

Formule φ a ψ jsou **logicky ekvivalentní**, jestliže pro každé pravdivostní ohodnocení v platí, že φ a ψ mají při ohodnocení v stejnou pravdivostní hodnotu, tj.

$$v \models \varphi \qquad \text{právě tehdy, když} \qquad v \models \psi.$$

To, že formule φ a ψ jsou logicky ekvivalentní, se označuje zápisem

$$\varphi \Leftrightarrow \psi.$$

Formule φ a ψ jsou logicky ekvivalentní právě tehdy, když $\varphi \leftrightarrow \psi$ je tautologie.

Ekvivalence formulí

Příklad: $\neg(p \to q) \Leftrightarrow p \wedge \neg q$

p	q	$p \to q$	$\neg(p \to q)$	$\neg q$	$p \wedge \neg q$
0	0	1	0	1	0
0	1	1	0	0	0
1	0	0	1	1	1
1	1	1	0	0	0

Ekvivalence formulí

Pro zdůvodnění toho, že formule φ a ψ **nejsou** ekvivalentní, stačí najít jedno ohodnocení v takové, že buď:

- $v \models \varphi$ a $v \not\models \psi$, nebo
- $v \not\models \varphi$ a $v \models \psi$.

Příklad: $p \vee (q \wedge r)$ není ekvivalentní $(p \vee q) \wedge r$

Ohodnocení v, kde:

- $v(p) = 1$
- $v(q) = 1$
- $v(r) = 0$

Při tomto ohodnocení platí $p \vee (q \wedge r)$, ale neplatí $(p \vee q) \wedge r$.

Některé důležité ekvivalence

– Ekvivalence týkající se negace:

$\neg\neg p \Leftrightarrow p$ *dvojitá negace*

– Ekvivalence týkající se konjunkce:

$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$ *asociativita*

$p \wedge q \Leftrightarrow q \wedge p$ *komutativita*

$p \wedge p \Leftrightarrow p$ *idempotence*

– Ekvivalence týkající se disjunkce:

$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ *asociativita*

$p \vee q \Leftrightarrow q \vee p$ *komutativita*

$p \vee p \Leftrightarrow p$ *idempotence*

Některé důležité ekvivalence

– Distributivní zákony pro $\wedge$ a $\vee$:

$$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$$
$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$$

– De Morganovy zákony:

$$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$$
$$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$$

– Ekvivalence týkající se implikace:

$$p \rightarrow q \Leftrightarrow \neg p \vee q$$
$$\neg(p \rightarrow q) \Leftrightarrow p \wedge \neg q$$

Některé důležité ekvivalence

– Ekvivalence týkající se spojky $\leftrightarrow$:

$$(p \leftrightarrow q) \leftrightarrow r \Leftrightarrow p \leftrightarrow (q \leftrightarrow r) \quad \textit{asociativita}$$

$$p \leftrightarrow q \Leftrightarrow q \leftrightarrow p \quad \textit{komutativita}$$

$$p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$$

$$p \leftrightarrow q \Leftrightarrow (p \vee \neg q) \wedge (\neg p \vee q)$$

$$p \leftrightarrow q \Leftrightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$$

Ekvivalence formulí

Řekněme, že formule φ a ψ jsou logicky ekvivalentní, tj.

$$\varphi \Leftrightarrow \psi\,.$$

Pokud ve φ a ψ nahradíme jednotlivé atomické výroky libovolnými formulemi (v obou stejně), dostaneme opět ekvivalentní formule.

Příklad: $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$

Pro libovolné formule χ_1 a χ_2 proto platí

$$\neg(\chi_1 \vee \chi_2) \Leftrightarrow \neg\chi_1 \wedge \neg\chi_2$$

Ekvivalence formulí

$$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$$

Náhrada atomických výroků:

- p nahradíme $q \vee \neg(r \rightarrow \neg s)$
- q nahradíme $\neg(q \leftrightarrow p)$

Dostaneme

$$\neg((q \vee \neg(r \rightarrow \neg s)) \vee \neg(q \leftrightarrow p)) \Leftrightarrow \neg(q \vee \neg(r \rightarrow \neg s)) \wedge \neg\neg(q \leftrightarrow p)$$

Ekvivalence formulí

Řekněme, že φ je formule a ψ nějaká její podformule.

Pokud nyní ve φ nahradíme nějaký výskyt podformule ψ formulí ψ' takovou, že $\psi \Leftrightarrow \psi'$, dostaneme tím z formule φ formuli φ' takovou, že

$$\varphi \Leftrightarrow \varphi' .$$

Příklad: Ve formuli

$$\neg((p \to q) \vee (\neg(p \to q) \to r))$$

nahradíme druhý výskyt podformule $p \to q$ ekvivalentní formulí $\neg p \vee q$.

Dostaneme

$$\neg((p \to q) \vee (\neg(\neg p \vee q) \to r))$$

Ekvivalentní úpravy

Pro libovolné formule φ, ψ a χ platí:

- $\varphi \Leftrightarrow \varphi$.
- Pokud $\varphi \Leftrightarrow \psi$, tak $\psi \Leftrightarrow \varphi$.
- Pokud $\varphi \Leftrightarrow \psi$ a $\psi \Leftrightarrow \chi$, tak $\varphi \Leftrightarrow \chi$.

Při zdůvodňování toho, že dané formule jsou ekvivalentní, můžeme postupovat po menších krocích:

Pokud například platí $\varphi_1 \Leftrightarrow \varphi_2$, $\varphi_2 \Leftrightarrow \varphi_3$, $\varphi_3 \Leftrightarrow \varphi_4$ a $\varphi_4 \Leftrightarrow \varphi_5$, můžeme z toho vyvodit, že platí

$$\varphi_1 \Leftrightarrow \varphi_5 .$$

Tento postup můžeme stručněji zapsat

$$\varphi_1 \Leftrightarrow \varphi_2 \Leftrightarrow \varphi_3 \Leftrightarrow \varphi_4 \Leftrightarrow \varphi_5$$

Ekvivalentní úpravy

Příklad: Zdůvodnění toho, že platí

$$(p \wedge q) \to r \;\Leftrightarrow\; p \to (q \to r)$$

$$\begin{aligned}
(p \wedge q) \to r &\Leftrightarrow \neg(p \wedge q) \vee r \\
&\Leftrightarrow (\neg p \vee \neg q) \vee r \\
&\Leftrightarrow \neg p \vee (\neg q \vee r) \\
&\Leftrightarrow \neg p \vee (q \to r) \\
&\Leftrightarrow p \to (q \to r)
\end{aligned}$$

Ekvivalentní úpravy

Každá formule se dá převést na ekvivalentní formuli, která obsahuje z logických spojek pouze "$\neg$", "$\wedge$" a "$\vee$".

- Spojku "$\leftrightarrow$" je možno odstranit pomocí libovolné z následujících tří ekvivalencí:
  - $p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$
  - $p \leftrightarrow q \Leftrightarrow (p \vee \neg q) \wedge (\neg p \vee q)$
  - $p \leftrightarrow q \Leftrightarrow (p \wedge q) \vee (\neg p \wedge \neg q)$
- Spojku "$\rightarrow$" je možno odstranit pomocí následující ekvivalence:
  - $p \rightarrow q \Leftrightarrow \neg p \vee q$

Příklad:

$$\begin{aligned}(\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r) &\Leftrightarrow (\neg\neg q \vee r) \wedge \neg(p \leftrightarrow r) \\ &\Leftrightarrow (\neg\neg q \vee r) \wedge \neg((p \wedge r) \vee (\neg p \wedge \neg r))\end{aligned}$$

Ekvivalentní úpravy

Každá formule se dá převést na ekvivalentní formuli, která obsahuje z logických spojek pouze "$\neg$", "$\wedge$" a "$\vee$", a kde jsou navíc negace aplikovány pouze na atomické výroky.

- Můžeme předpokládat, že formule obsahuje pouze "$\neg$", "$\wedge$" a "$\vee$".
- Negace můžeme postupně „zatlačit" k atomickým výrokům použitím následujících tří ekvivalencí:
  - $\neg\neg p \Leftrightarrow p$
  - $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$
  - $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$

Ekvivalentní úpravy

Příklad:

$$
\begin{aligned}
&(\neg\neg q \vee r) \wedge \neg((p \wedge r) \vee (\neg p \wedge \neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge \neg((p \wedge r) \vee (\neg p \wedge \neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge (\neg(p \wedge r) \wedge \neg(\neg p \wedge \neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge ((\neg p \vee \neg r) \wedge \neg(\neg p \wedge \neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge ((\neg p \vee \neg r) \wedge (\neg\neg p \vee \neg\neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge ((\neg p \vee \neg r) \wedge (p \vee \neg\neg r)) \\
&\quad \Leftrightarrow \ (q \vee r) \wedge ((\neg p \vee \neg r) \wedge (p \vee r))
\end{aligned}
$$

Logické konstanty

Pro některé účely se hodí zavést následující dvě speciální formule:

- $\top$ — formule, která je vždy pravdivá
- $\bot$ — formule, která je vždy nepravdivá

Pro každé pravdivostní ohodnocení v tedy platí:

- $v \models \top$ ($\top$ má tedy vždy pravdivostní hodnotu 1)
- $v \not\models \bot$ ($\bot$ má tedy vždy pravdivostní hodnotu 0)

Logické konstanty

Symboly $\top$ a $\bot$ je možné chápat jako „zkratky“:

- $\top$ zastupuje libovolnou tautologii (např. $p \rightarrow p$)
- $\bot$ zastupuje libovolnou kontradikci (např. $p \wedge \neg p$)

Alternativou by bylo rozšířit definici syntaxe a sémantiky výrokové logiky o příslušné položky.

Na $\top$ a $\bot$ se lze dívat jako na logické spojky s aritou 0.

Logické konstanty

Příklady ekvivalencí, které platí pro $\top$ a $\bot$ (a pro libovolné p):

$$\top \Leftrightarrow p \vee \neg p \qquad \bot \Leftrightarrow p \wedge \neg p$$
$$\neg\top \Leftrightarrow \bot \qquad \neg\bot \Leftrightarrow \top$$
$$p \wedge \top \Leftrightarrow p \qquad p \vee \bot \Leftrightarrow p$$
$$p \vee \top \Leftrightarrow \top \qquad p \wedge \bot \Leftrightarrow \bot$$

Ekvivalence formulí

Ekvivalentní formule **nemusí** nutně obsahovat stejné atomické výroky.

Příklad: $(q \to \neg\neg q) \wedge \neg p \;\Leftrightarrow\; p \to (r \wedge \neg r)$

$$
\begin{aligned}
(q \to \neg\neg q) \wedge \neg p &\Leftrightarrow (q \to q) \wedge \neg p \\
&\Leftrightarrow \top \wedge \neg p \\
&\Leftrightarrow \neg p \\
&\Leftrightarrow \neg p \vee \bot \\
&\Leftrightarrow p \to \bot \\
&\Leftrightarrow p \to (r \wedge \neg r)
\end{aligned}
$$

Například také libovolné dvě tautologie jsou spolu ekvivalentní.

Konjunkce a disjunkce více formulí

Díky asociativitě konjunkce platí například:

$$p \wedge ((q \wedge r) \wedge (s \wedge t)) \Leftrightarrow (p \wedge q) \wedge ((r \wedge s) \wedge t)$$

Obě tyto formule jsou také ekvivalentní formulím

- $p \wedge (q \wedge (r \wedge (s \wedge t)))$
- $(((p \wedge q) \wedge r) \wedge s) \wedge t$

Všechny výše uvedené formule jsou pravdivé právě tehdy, když jsou pravdivé všechny výroky p, q, r, s a t.

Konvence: Díky asociativitě konjunkce je možno vypustit závorky a psát

$$p \wedge q \wedge r \wedge s \wedge t$$

Konjunkce a disjunkce více formulí

Protože je konjunkce nejen asociativní, ale i komutativní, nezáleží na pořadí členů v takové komplikovanější konjunkci, např.:

$$r \wedge t \wedge q \wedge s \wedge p \quad \Leftrightarrow \quad p \wedge q \wedge r \wedge s \wedge t$$

Díky idempotenci není podstatné, kolikrát se v dané konjunkci který člen vyskytuje, např.:

$$p \wedge q \wedge p \quad \Leftrightarrow \quad q \wedge p \wedge q \wedge q$$

Konjunkce a disjunkce více formulí

Totéž, co platí pro konjunkci, platí i pro disjunkci, např.:

$$(p \vee q) \vee (r \vee q) \quad \Leftrightarrow \quad q \vee (p \vee (r \vee (r \vee r)))$$

Konvence: Místo $(p \vee q) \vee (r \vee (s \vee t))$ lze psát

$$p \vee q \vee r \vee s \vee t$$

Toto vše platí nejen pro atomické výroky, ale pro libovolné formule, např.:

- Místo $(\varphi_1 \wedge \varphi_2) \wedge (\varphi_3 \wedge (\varphi_4 \wedge \varphi_5))$ lze psát

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$$

Konjunkce a disjunkce více formulí

Konjunkcí n formulí $\varphi_1, \varphi_2, \ldots, \varphi_n$, kde $n \geq 0$, budeme rozumět formuli

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$$

Speciálně:

- Pro $n = 0$ bude touto konjunkcí formule $\top$.
- Pro $n = 1$ bude touto konjunkcí formule φ_1.

Disjunkcí n formulí $\varphi_1, \varphi_2, \ldots, \varphi_n$, kde $n \geq 0$, budeme rozumět formuli

$$\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$$

Speciálně:

- Pro $n = 0$ bude touto disjunkcí formule $\bot$.
- Pro $n = 1$ bude touto disjunkcí formule φ_1.

Konjunkce a disjunkce více formulí

Konjunkce $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$:

- Celá formule je pravdivá právě tehdy, když všechny formule φ_1, φ_2, $\ldots$, φ_n jsou pravdivé.
- Pokud je některá formule φ_i ekvivalentní $\bot$, je celá formule ekvivalentní $\bot$.
- Pokud je některá formule φ_i ekvivalentní negaci nějaké formule φ_j (tj. $\varphi_i \Leftrightarrow \neg\varphi_j$), pak celá formule je ekvivalentní $\bot$.
- Pokud je některá formule φ_i ekvivalentní $\top$, je možné formuli φ_i z celé formule vypustit.

Konjunkce a disjunkce více formulí

Disjunkce $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$:

- Celá formule je pravdivá právě tehdy, když alespoň jedna z formulí φ_1, φ_2, ..., φ_n je pravdivá.
- Pokud je některá formule φ_i ekvivalentní $\top$, je celá formule ekvivalentní $\top$.
- Pokud je některá formule φ_i ekvivalentní negaci nějaké formule φ_j (tj. $\varphi_i \Leftrightarrow \neg\varphi_j$), pak celá formule je ekvivalentní $\top$.
- Pokud je některá formule φ_i ekvivalentní $\bot$, je možné formuli φ_i z celé formule vypustit.

Normální formy formulí

- **Literál** — atomický výrok nebo jeho negace, např.

$$p \qquad \neg q \qquad \neg r$$

- **Elementární konjunkce** — konjunkce jednoho nebo více literálů, např.

$$(p \wedge \neg q) \qquad (r) \qquad (q \wedge \neg r \wedge p)$$

- **Elementární disjunkce** (**klauzule**) — disjunkce jednoho nebo více literálů, např.

$$(p \vee \neg q) \qquad (r) \qquad (q \vee \neg r \vee p)$$

Normální formy formulí

Příklad:

- Elementární konjunkce

$$(p \wedge \neg q \wedge r \wedge \neg s \wedge \neg t)$$

je **pravdivá** právě v těch pravdivostních ohodnoceních v, kde

$v(p) = 1 \qquad v(q) = 0 \qquad v(r) = 1 \qquad v(s) = 0 \qquad v(t) = 0$

- Elementární disjunkce

$$(p \vee \neg q \vee r \vee \neg s \vee \neg t)$$

je **nepravdivá** právě v těch pravdivostních ohodnoceních v, kde

$v(p) = 0 \qquad v(q) = 1 \qquad v(r) = 0 \qquad v(s) = 1 \qquad v(t) = 1$

Normální formy formulí

- **Disjunktivní normální forma** (**DNF**) — disjunkce jedné nebo více elementárních konjunkcí, např.

$$(p \wedge \neg q) \vee (\neg r) \vee (\neg r \wedge \neg p \wedge \neg q)$$

- **Konjunktivní normální forma** (**KNF**) — konjunkce jedné nebo více elementárních disjunkcí (klauzulí), např.

$$(p \vee \neg q) \wedge (\neg r) \wedge (\neg r \vee \neg p \vee \neg q)$$

Poznámka: Formule $\bot$ a $\top$ také budeme považovat za formule v DNF a KNF.

Normální formy formulí

Předpokládejme, že u formule φ v KNF žádná elementární disjunkce neobsahuje zároveň literály p a $\neg p$ (takové elementární disjunkce je možno vypustit).

Tato formule φ je **tautologie** právě tehdy, když je $\top$, tj. když neobsahuje žádné elementární disjunkce.

Předpokládejme, že u formule ψ v DNF žádná elementární konjunkce neobsahuje zároveň literály p a $\neg p$ (takové elementární konjunkce je možno vypustit).

Tato formule ψ je **kontradikce** právě tehdy, když je $\bot$, tj. když neobsahuje žádné elementární konjunkce.

Normální formy formulí

Převod formule do DNF a do KNF:

- Můžeme předpokládat, že formule obsahuje pouze atomické výroky, spojky "$\neg$" aplikované na atomické výroky a spojky "$\wedge$" a "$\vee$".
- Požadovaný tvar formule můžeme dosáhnout pomocí následujících ekvivalencí:
  - $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$ — při převodu do DNF
  - $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ — při převodu do KNF

Normální formy formulí

Příklad: Převod formule $q \wedge ((\neg p \vee \neg r) \wedge (p \vee r))$ do DNF:

$$
\begin{aligned}
& q \wedge ((\neg p \vee \neg r) \wedge (p \vee r)) \\
\Leftrightarrow\ & (q \wedge (\neg p \vee \neg r)) \wedge (p \vee r) \\
\Leftrightarrow\ & ((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge (p \vee r) \\
\Leftrightarrow\ & (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge p) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (((q \wedge \neg p) \wedge p) \vee ((q \wedge \neg r) \wedge p)) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (q \wedge \neg p \wedge p) \ \vee \ (q \wedge \neg r \wedge p) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (q \wedge \bot) \ \vee \ (q \wedge \neg r \wedge p) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & \bot \ \vee \ (q \wedge \neg r \wedge p) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (q \wedge \neg r \wedge p) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \ \vee \ (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & \quad \ldots
\end{aligned}
$$

Normální formy formulí

$$
\begin{aligned}
&\ldots \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (((q \wedge \neg p) \vee (q \wedge \neg r)) \wedge r) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (((q \wedge \neg p) \wedge r) \vee ((q \wedge \neg r) \wedge r)) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (q \wedge \neg p \wedge r) \vee (q \wedge \neg r \wedge r) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee (q \wedge \neg r \wedge r) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee (q \wedge \bot) \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r) \vee \bot \\
\Leftrightarrow\ & (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge r)
\end{aligned}
$$

Normální formy formulí

K dané tabulce pravdivostních hodnot můžeme snadno vyrobit příslušné formule v DNF a KNF:

p	q	r	φ
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

DNF:

$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge r)$

KNF:

$(p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$

Normální formy formulí

Pokud uvažujeme pevně danou **konečnou** množinu atomických výroků At:

- **Úplná disjunktivní normální forma** (**ÚDNF**) — formule v DNF, kde každá elementární konjunkce obsahuje každý atomický výrok z At právě jednou.

  Příklad: $(p \wedge \neg q \wedge \neg r) \vee (p \wedge q \wedge \neg r) \vee (\neg p \wedge q \wedge \neg r)$

- **Úplná konjunktivní normální forma** (**ÚKNF**) — formule v KNF, kde každá klauzule obsahuje každý atomický výrok z At právě jednou.

  Příklad: $(p \vee \neg q \vee \neg r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee q \vee \neg r)$

Poznámka: V příkladech je $At = \{p, q, r\}$.

Minimální množiny logických spojek

Z předchozího vidíme, že spojky "$\neg$", "$\wedge$" a "$\vee$" postačují na vytvoření formule pro jakoukoliv tabulku pravdivostních hodnot.

Ve skutečnosti stačí i některé menší množiny logických spojek:

- "$\neg$", "$\wedge$":
  $\varphi \vee \psi$ je možno vyjádřit jako $\neg(\neg\varphi \wedge \neg\psi)$
- "$\neg$", "$\vee$":
  $\varphi \wedge \psi$ je možno vyjádřit jako $\neg(\neg\varphi \vee \neg\psi)$
- "$\neg$", "$\rightarrow$":
  $\varphi \vee \psi$ je možno vyjádřit jako $\neg\varphi \rightarrow \psi$
  $\varphi \wedge \psi$ je možno vyjádřit jako $\neg(\varphi \rightarrow \neg\psi)$

Minimální množiny logických spojek

- "$\rightarrow$", "$\bot$":
  $\neg\varphi$ je možno vyjádřit jako $\varphi \rightarrow \bot$
  $\varphi \vee \psi$ je možno vyjádřit jako $(\varphi \rightarrow \bot) \rightarrow \psi$
  $\varphi \wedge \psi$ je možno vyjádřit jako $(\varphi \rightarrow (\psi \rightarrow \bot)) \rightarrow \bot$

- "$|$" — NAND — Shefferova funkce (též se označuje "$\uparrow$"):

φ	ψ	$\varphi \mid \psi$
0	0	1
0	1	1
1	0	1
1	1	0

  $\neg\varphi$ je možno vyjádřit jako $\varphi \mid \varphi$
  $\varphi \vee \psi$ je možno vyjádřit jako $(\varphi \mid \varphi) \mid (\psi \mid \psi)$
  $\varphi \wedge \psi$ je možno vyjádřit jako $(\varphi \mid \psi) \mid (\varphi \mid \psi)$

Minimální množiny logických spojek

- "$\downarrow$" — NOR — Peirceova funkce:

φ	ψ	$\varphi \downarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	0

$\neg\varphi$ je možno vyjádřit jako $\varphi \downarrow \varphi$
$\varphi \vee \psi$ je možno vyjádřit jako $(\varphi \downarrow \psi) \downarrow (\varphi \downarrow \psi)$
$\varphi \wedge \psi$ je možno vyjádřit jako $(\varphi \downarrow \varphi) \downarrow (\psi \downarrow \psi)$

Logické vyplývání

Definice

Formule ψ **logicky vyplývá** z formule φ, jestliže při každém pravdivostním ohodnocení v, při kterém platí φ, platí i ψ.

To, že ψ logicky vyplývá z φ se označuje zápisem

$$\varphi \models \psi.$$

- φ — předpoklad
- ψ — závěr

Formule ψ logicky vyplývá z φ (tj. $\varphi \models \psi$) právě tehdy, když $\varphi \rightarrow \psi$ je tautologie.

Logické vyplývání

Příklad: Formule $r \to p$ logicky vyplývá z formule $p \vee (q \wedge \neg r)$, tj.

$$p \vee (q \wedge \neg r) \models r \to p$$

	p	q	r	$p \vee (q \wedge \neg r)$	$r \to p$
	0	0	0	0	1
	0	0	1	0	0
$*$	0	1	0	1	1
	0	1	1	0	0
$*$	1	0	0	1	1
$*$	1	0	1	1	1
$*$	1	1	0	1	1
$*$	1	1	1	1	1

Logické vyplývání

Předpokladů může být libovolný počet:

Definice

Formule ψ **logicky vyplývá** z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$, jestliže při každém pravdivostním ohodnocení v, při kterém platí všechny tyto předpoklady, platí i ψ.

To, že ψ logicky vyplývá z $\varphi_1, \varphi_2, \ldots, \varphi_n$ se označuje zápisem

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi.$$

- $\varphi_1, \varphi_2, \ldots, \varphi_n$ — předpoklady
- ψ — závěr

Logické vyplývání

Příklad:

- *Jestliže měl vlak zpoždění a na nádraží nebyly taxíky, tak Honza přišel pozdě do práce.*
- *Honza nepřišel do práce pozdě.*
- *Vlak měl zpoždění.*

- *Na nádraží byly taxíky.*

$$(p \wedge \neg q) \rightarrow r,\ \neg r,\ p \models q$$

Logické vyplývání

$$(p \wedge \neg q) \rightarrow r,\ \neg r,\ p \models q$$

	p	q	r	$(p \wedge \neg q) \rightarrow r$	$\neg r$	p	q
	0	0	0	1	1	0	0
	0	0	1	1	0	0	0
	0	1	0	1	1	0	1
	0	1	1	1	0	0	1
	1	0	0	0	1	1	0
	1	0	1	1	0	1	0
$*$	1	1	0	1	1	1	1
	1	1	1	1	0	1	1

Logické vyplývání

Pro konkrétnost řekněme, že máme celkem 4 předpoklady, ale podobně to platí i pro libovolný (konečný) počet předpokladů:

$$\varphi_1, \varphi_2, \varphi_3, \varphi_4 \models \psi$$

právě tehdy, když

$$\varphi_1, \varphi_2, \varphi_3 \models \varphi_4 \to \psi$$

právě tehdy, když

$$\varphi_1, \varphi_2 \models \varphi_3 \to (\varphi_4 \to \psi)$$

právě tehdy, když

$$\varphi_1 \models \varphi_2 \to (\varphi_3 \to (\varphi_4 \to \psi))$$

právě tehdy, když

$$\models \varphi_1 \to (\varphi_2 \to (\varphi_3 \to (\varphi_4 \to \psi)))$$

(tj. právě tehdy, když $\varphi_1 \to (\varphi_2 \to (\varphi_3 \to (\varphi_4 \to \psi)))$ je tautologie)

Logické vyplývání

Příklad: Skutečně platí

$$(p \wedge \neg q) \rightarrow r,\ \neg r,\ p \models q$$

(tj. závěr q logicky vyplývá z předpokladů $(p \wedge \neg q) \rightarrow r$, $\neg r$ a p), protože

$$((p \wedge \neg q) \rightarrow r) \rightarrow (\neg r \rightarrow (p \rightarrow q))$$

je tautologie.

(Lze ověřit například pomocí tabulkové metody nebo pomocí nalezení sémantického sporu.)

Logické vyplývání

Věta o dedukci

$$\varphi_1, \varphi_2, \ldots \varphi_n, \chi \models \psi$$

právě tehdy, když

$$\varphi_1, \varphi_2, \ldots \varphi_n \models \chi \rightarrow \psi$$

Logické vyplývání

$$\varphi_1, \varphi_2, \ldots \varphi_n \models \psi$$

právě tehdy, když

$$(\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n) \rightarrow \psi \text{ je tautologie}$$

Zdůvodnění (pro případ se čtyřmi předpoklady):

$$\begin{aligned}
& \varphi_1 \rightarrow (\varphi_2 \rightarrow (\varphi_3 \rightarrow (\varphi_4 \rightarrow \psi))) \\
\Leftrightarrow\ & \varphi_1 \rightarrow (\varphi_2 \rightarrow ((\varphi_3 \wedge \varphi_4) \rightarrow \psi)) \\
\Leftrightarrow\ & \varphi_1 \rightarrow ((\varphi_2 \wedge \varphi_3 \wedge \varphi_4) \rightarrow \psi)) \\
\Leftrightarrow\ & (\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4) \rightarrow \psi
\end{aligned}$$

Poznámka: Využívá toho, že platí $p \rightarrow (q \rightarrow r) \ \Leftrightarrow\ (p \wedge q) \rightarrow r$.

Logické vyplývání

$$\varphi_1, \varphi_2, \ldots \varphi_n \models \psi$$

právě tehdy, když

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \quad \Leftrightarrow \quad \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \wedge \psi$$

$$\varphi \quad \Leftrightarrow \quad \psi$$

právě tehdy, když

$$\varphi \models \psi \qquad \text{a} \qquad \psi \models \varphi$$

Logické vyplývání

Pokud je formule ψ **tautologie**, tak logicky vyplývá z jakékoliv množiny předpokladů, tj. pro jakoukoliv množinu předpokladů $\varphi_1, \varphi_2, \ldots \varphi_n$ platí

$$\varphi_1, \varphi_2, \ldots \varphi_n \models \psi$$

Speciálně, pokud je ψ tautologie, tak vyplývá i z prázdné množiny předpokladů:

$$\models \psi$$

Tautologie jsou jediné formule, které logicky vyplývají z prázdné množiny předpokladů.

Logické vyplývání

Řekněme, že platí

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \chi_1 \qquad \varphi_1, \varphi_2, \ldots, \varphi_n \models \chi_2$$

a zároveň $\chi_1, \chi_2 \models \psi$.

Pak platí i $\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi$.

Příklad:

- Pokud $\varphi_1, \varphi_2, \varphi_3 \models (q \vee \neg p)$ a $\varphi_1, \varphi_2, \varphi_3 \models \neg s$, tak

$$\varphi_1, \varphi_2, \varphi_3 \models (q \vee \neg p) \wedge \neg s,$$

protože $q \vee \neg p, \neg s \models (q \vee \neg p) \wedge \neg s$.

Logické vyplývání

Příklad:

- Pokud $\varphi_1, \varphi_2, \varphi_3 \models p \to q$ a $\varphi_1, \varphi_2, \varphi_3 \models p$, tak

  $$\varphi_1, \varphi_2, \varphi_3 \models q,$$

  protože $p \to q,\ p \models q$.

- Pokud $\varphi_1, \varphi_2, \varphi_3, \varphi_4 \models \neg p \to \neg q$, tak

  $$\varphi_1, \varphi_2, \varphi_3, \varphi_4 \models q \to p,$$

  protože $\neg p \to \neg p \models q \to p$.

Logické vyplývání

Při zdůvodnění toho, že závěr ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$ je možno postupovat po menších krocích.

Začneme s předpoklady, například:

$$\varphi_1,\ \varphi_2,\ \varphi_3$$

Postupně přidáváme další formule tak, aby každá nově přidaná formule logicky vyplývala z předchozích, například:

$$\varphi_1,\ \varphi_2,\ \varphi_3,\ \chi_1,\ \chi_2,\ \chi_3,\ \chi_4,\ \chi_5,\ \chi_6,\ \chi_7,\ \chi_8,\ \psi$$

Logické vyplývání

Předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ jsou **nekonzistentní** (**sporné**), jestliže neexistuje žádné pravdivostní ohodnocení v, při kterém by byly všechny tyto předpoklady pravdivé.

Předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ jsou nekonzistentní právě tehdy, když

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$$

je kontradikce.

Příklad: Předpoklady $p \rightarrow q$, $r \rightarrow p$, r, $\neg q$ jsou nekonzistentní.

Logické vyplývání

V případě, kdy předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ jsou nekonzistentní, tak $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$ je kontradikce, takže

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \;\Leftrightarrow\; \bot.$$

V tom případě tedy platí

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \;\Leftrightarrow\; \bot \;\Leftrightarrow\; \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \wedge \bot,$$

z čehož plyne, že

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \bot.$$

Logické vyplývání

Předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ jsou nekonzistentní právě tehdy, když

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \bot.$$

Protože $\bot \rightarrow \psi$ je tautologie (pro jakoukoliv formuli ψ), platí pro libovolnou formuli ψ v případě, kdy jsou předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ nekonzistentní

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi\,.$$

Z nekonzistentních předpokladů tedy logicky vyplývá jakákoliv formule.

Speciálně pro jakoukoliv formuli χ platí, že z nekonzistentních předpokladů vyplývá jak χ, tak $\neg\chi$.

Logické vyplývání

Předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ jsou nekonzistentní právě tehdy, když existuje nějaká formule χ taková, že

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \chi \qquad \text{a zároveň} \qquad \varphi_1, \varphi_2, \ldots, \varphi_n \models \neg\chi$$

Formule $\chi \to (\neg\chi \to \bot)$ je tautologie (pro libovolnou formuli χ).

Pokud tedy platí

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \chi \qquad \text{a zároveň} \qquad \varphi_1, \varphi_2, \ldots, \varphi_n \models \neg\chi$$

tak platí i

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \bot.$$

Naopak pokud platí za daných předpokladů $\bot$, tak za těchto předpokladů platí i χ a $\neg\chi$.

Princip důkazu sporem

$$\varphi_1, \varphi_2, \ldots \varphi_n \models \psi$$

právě tehdy, když

předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi$ jsou nekonzistentní

Předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi$ jsou nekonzistentní právě tehdy, když

$$\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi \models \bot,$$

což platí právě tehdy, když

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \neg\psi \to \bot.$$

Platí $\neg\psi \to \bot \Leftrightarrow \psi$ (protože $\neg p \to \bot \Leftrightarrow \neg\neg p \vee \bot \Leftrightarrow \neg\neg p \Leftrightarrow p$).

Rezoluční metoda

Rezoluční metoda je jedním z algoritmů pro zjištění toho, zda daný závěr vyplývá z daných předpokladů.

Řeší následující problém:

Vstup: Formule φ_1, φ_2, ..., φ_n, ψ.

Otázka: Platí $\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi$?

Poznámka: Dá se použít také pro zjištění, zda je daná formule tautologie, kontradikce nebo splnitelná.

Na použití různých variant rezoluční metody jsou založeny některé systémy pro automatické dokazování a také logický programovací jazyk Prolog.

Rezoluční metoda

- Pracuje s formulemi v KNF.
- Vytváří důkaz toho, že daný závěr plyne z předpokladů.
- Jedná se o důkaz sporem — postupně generuje formule, které vyplývají z předpokladů

$$\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi$$

- Výpočet může skončit dvěma různými způsoby:
  - Podaří se najít spor, tj. podaří se odvodit formuli $\bot$ — pak platí, že závěr ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$.
  - Nepodaří se odvodit $\bot$ a žádné další nové formule se nedají přidat — pak závěr ψ z daných předpokladů nevyplývá.

Rezoluční metoda

- Rezoluční metoda pracuje s formulemi, které mají podobu elementárních disjukcí, např.

$$(\neg p \vee q \vee \neg s \vee \neg t)$$

Těmto formulím se říká **klauzule**.

- Speciálním případem klauzule je **prázdná klauzule** $\bot$, která představuje nalezený spor.

- Algorimus začne svou činnost tím, že formule

$$\varphi_1, \varphi_2, \ldots, \varphi_n, \neg\psi$$

převede do KNF a vezme všechny klazule z takto vytvořených formulí jako počáteční množinu předpokladů

$$\chi_1, \chi_2, \ldots, \chi_m.$$

Rezoluční metoda

Při generování dalších klauzulí ke dříve přidaným klauzulím se používá tzv. **rezoluční pravidlo** (či **rezoluční princip**):

Pro libovolné formule φ, ψ a χ platí

$$\varphi \vee \psi, \neg\varphi \vee \chi \models \psi \vee \chi$$

V rezoluční metodě se tento princip používá jen pro klauzule.
V případě rezoluční metody jsou $\varphi \vee \psi$, $\neg\varphi \vee \chi$ a $\psi \vee \chi$ vždy klauzule a φ je navíc atomický výrok.

Příklad: Z klauzulí

$$p \vee \neg q \vee r \vee s \qquad \text{a} \qquad \neg r \vee t \vee \neg u$$

je možno vyvodit pomocí rezolučního pravidla klauzuli $p \vee \neg q \vee s \vee t \vee \neg u$.

Rezoluční metoda

Poznámky:

- Pořadí literálů v klauzulích není podstatné.
- Vícenásobné výskyty stejných literálů v téže klauzuli je možno odstranit.
- Pokud je nově vygenerovaná klauzule stejná, jako nějaká dříve přidaná klauzule (a liší se nanejvýš pořadím literálů), nemá smysl ji přidávat.
- Klauzule, které obsahují zároveň literály p a $\neg p$ jsou ekvivalentní $\top$ a je možné je odstranit.
- Klauzule je možno používat pro aplikaci rezolučního pravidla opakovaně (s jinými klauzulemi).

Rezoluční metoda

Speciální případy použití rezolučního pravidla:

- Jedna z klauzulí obsahuje jen jeden literál a druhá více než jeden literál:

  Z klauzulí

  $$\neg q \qquad\qquad p \vee q \vee \neg t$$

  je možno vyvodit klauzuli $p \vee \neg t$.

- Obě klauzule obsahují jeden literál:

  Z klauzulí

  $$p \qquad\qquad \neg p$$

  je možno vyvodit prázdnou klauzuli $\bot$, tj. spor.

Rezoluční metoda

Chceme ověřit platnost následujícího úsudku:

- *Není pravda, že Jana je ve škole a Petr není doma.*
- *Jana není ve škole nebo je všední den nebo prší.*
- *Jestliže je všední den, pak Petr není doma.*

- *Jestliže je Jana ve škole, pak prší.*

Jednotlivá tvrzení nejprve zformalizujeme pomocí formulí výrokové logiky:

$$\frac{\begin{array}{l}\neg(j \wedge \neg p) \\ \neg j \vee d \vee r \\ d \to \neg p\end{array}}{j \to r}$$

j – Jana je škole
p – Petr je doma
d – je všední den
r – prší

Rezoluční metoda

$$\frac{\begin{array}{l}\neg(j \wedge \neg p)\\ \neg j \vee d \vee r\\ d \rightarrow \neg p\end{array}}{j \rightarrow r}$$

Jednotlivé předpoklady převedeme do KNF:

- $\neg(j \wedge \neg p) \Leftrightarrow \neg j \vee p$
- $\neg j \vee d \vee r$
- $d \rightarrow \neg p \Leftrightarrow \neg d \vee \neg p$

Závěr znegujeme a převedeme do KNF:

- $\neg(j \rightarrow r) \Leftrightarrow j \wedge \neg r$

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1. $\neg j \vee p$ – předpoklad 1
2. $\neg j \vee d \vee r$ – předpoklad 2
3. $\neg d \vee \neg p$ – přespoklad 3
4. j – 1. klauzule z negovaného závěru
5. $\neg r$ – 2. klauzule z negovaného závěru

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1.	$\neg j \vee p$	– předpoklad 1
2.	$\neg j \vee d \vee r$	– předpoklad 2
3.	$\neg d \vee \neg p$	– přespoklad 3
4.	j	– 1. klauzule z negovaného závěru
5.	$\neg r$	– 2. klauzule z negovaného závěru
6.	p	– rezoluce: 1,4

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1.	$\neg j \vee p$	– předpoklad 1
2.	$\neg j \vee d \vee r$	– předpoklad 2
3.	$\neg d \vee \neg p$	– přespoklad 3
4.	j	– 1. klauzule z negovaného závěru
5.	$\neg r$	– 2. klauzule z negovaného závěru
6.	p	– rezoluce: 1,4
7.	$d \vee r$	– rezoluce: 2,4

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1.	$\neg j \vee p$	– předpoklad 1
2.	$\neg j \vee d \vee r$	– předpoklad 2
3.	$\neg d \vee \neg p$	– přespoklad 3
4.	j	– 1. klauzule z negovaného závěru
5.	$\neg r$	– 2. klauzule z negovaného závěru
6.	p	– rezoluce: 1,4
7.	$d \vee r$	– rezoluce: 2,4
8.	$\neg d$	– rezoluce: 3,6

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1.	$\neg j \vee p$	– předpoklad 1
2.	$\neg j \vee d \vee r$	– předpoklad 2
3.	$\neg d \vee \neg p$	– přespoklad 3
4.	j	– 1. klauzule z negovaného závěru
5.	$\neg r$	– 2. klauzule z negovaného závěru
6.	p	– rezoluce: 1,4
7.	$d \vee r$	– rezoluce: 2,4
8.	$\neg d$	– rezoluce: 3,6
9.	r	– rezoluce: 7,8

Rezoluční metoda

Sepíšeme si jednotlivé klauzule:

1.	$\neg j \vee p$	– předpoklad 1
2.	$\neg j \vee d \vee r$	– předpoklad 2
3.	$\neg d \vee \neg p$	– přespoklad 3
4.	j	– 1. klauzule z negovaného závěru
5.	$\neg r$	– 2. klauzule z negovaného závěru
6.	p	– rezoluce: 1,4
7.	$d \vee r$	– rezoluce: 2,4
8.	$\neg d$	– rezoluce: 3,6
9.	r	– rezoluce: 7,8
10.	$\bot$	– rezoluce: 5,9

Byl odvozen spor, takže závěr skutečně z daných předpokladů vyplývá.

Rezoluční metoda

Poznámky:

- Na rezoluční metodu se lze dívat jako na vytváření jediné „obří“ formule v KNF, která je ekvivalentní formuli

$$\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n \wedge \neg\psi,$$

  a která vzniká postupným přidáváním jednotlivých klauzulí.

- Pokud se nepodaří vygenerovat spor, lze odvozené klauzule použít k nalezení příkladu pravdivostního ohodnocení v, při kterém platí předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$ a neplatí závěr ψ.

Rezoluční metoda

- Lze postupovat i přímou metodou, kdy se začne jen z předpokladů

$$\varphi_1, \varphi_2, \ldots, \varphi_n$$

a snažíme se postupně vygenerovat všechny klauzule závěru ψ.

Tento postup však nezaručuje, že se tyto klauzule podaří vygenerovat ve všech případech, kdy závěr ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$.

Příklad: Klauzuli $p \vee q$ tímto způsobem nelze vygenerovat z předpokladu p, i když platí

$$p \models p \vee q.$$

Predikátová logika

Predikátová logika

- *Ryby jsou obratlovci žijící ve vodě.*
- *Kapři jsou ryby.*
- *Existuje alespoň jeden kapr.*

- *Existuje alespoň jeden obratlovec žijící ve vodě.*

- *Trojúhelníky jsou konvexní mnohoúhelníky.*
- *Rovnostranné trojúhelníky jsou trojúhelníky.*
- *Existuje alespoň jeden rovnostranný trojúhelník.*

- *Existuje alespoň jeden konvexní mnohoúhelník.*

Predikátová logika

- *Ryby jsou obratlovci žijící ve vodě.*
- *Kapři jsou ryby.*
- *Existuje alespoň jeden kapr.*

- *Existuje alespoň jeden obratlovec žijící ve vodě.*

Použití **proměnných**:

- *Pro každé x platí, že pokud x je ryba, tak x je obratlovec a x žije ve vodě.*
- *Pro každé x platí, že pokud x je kapr, tak x je ryba.*
- *Existuje alespoň jedno x takové, že x je kapr.*

- *Existuje alespoň jedno x takové, že x je obratlovec a x žije ve vodě.*

Predikátová logika

- *Trojúhelníky jsou konvexní mnohoúhelníky.*
- *Rovnostranné trojúhelníky jsou trojúhelníky.*
- *Existuje alespoň jeden rovnostranný trojúhelník.*

- *Existuje alespoň jeden konvexní mnohoúhelník.*

Použití **proměnných**:

- *Pro každé x platí, že pokud x je trojúhelník, tak x je mnohoúhelník a x je konvexní.*
- *Pro každé x platí, že pokud x je rovnostranný trojúhelník, tak x je trojúhelník.*
- *Existuje alespoň jedno x takové, že x je rovnostranný trojúhelník.*

- *Existuje alespoň jedno x takové, že x je mnohoúhelník a x je konvexní.*

Predikátová logika

- *Pro každé x platí, že pokud x má vlastnost P, tak x má vlastnost Q a x má vlastnost R.*
- *Pro každé x platí, že pokud x má vlastnost S, tak x má vlastnost P.*
- *Existuje alespoň jedno x takové, že x má vlastnost S.*

- *Existuje alespoň jedno x takové, že x má vlastnost Q a x má vlastnost R.*

P	je ryba	je trojúhelník
Q	je obratlovec	je mnohoúhelník
R	žije ve vodě	je konvexní
S	je kapr	je rovnostranný trojúhelník

Predikátová logika

- *Pro každé x platí, že pokud $P(x)$, tak $Q(x)$ a $R(x)$.*
- *Pro každé x platí, že pokud $S(x)$, tak $P(x)$.*
- *Existuje x takové, že $S(x)$.*

- *Existuje x takové, že $Q(x)$ a $R(x)$.*

$P(x)$	x je ryba	x je trojúhelník
$Q(x)$	x je obratlovec	x je mnohoúhelník
$R(x)$	x žije ve vodě	x je konvexní
$S(x)$	x je kapr	x je rovnostranný trojúhelník

Predikátová logika

- *Pro každé* x *platí* $(P(x) \rightarrow (Q(x) \wedge R(x)))$.
- *Pro každé* x *platí* $(S(x) \rightarrow P(x))$.
- *Existuje* x *takové, že* $S(x)$.

- *Existuje* x *takové, že* $(Q(x) \wedge R(x))$.

$P(x)$	x je ryba	x je trojúhelník
$Q(x)$	x je obratlovec	x je mnohoúhelník
$R(x)$	x žije ve vodě	x je konvexní
$S(x)$	x je kapr	x je rovnostranný trojúhelník

Predikátová logika

- $\forall x(P(x) \rightarrow (Q(x) \wedge R(x)))$
- $\forall x(S(x) \rightarrow P(x))$
- $\exists x\, S(x)$

- $\exists x(Q(x) \wedge R(x))$

$P(x)$	x je ryba	x je trojúhelník
$Q(x)$	x je obratlovec	x je mnohoúhelník
$R(x)$	x žije ve vodě	x je konvexní
$S(x)$	x je kapr	x je rovnostranný trojúhelník

- $\forall$ — univerzální kvantifikátor („*pro každé*“)
- $\exists$ — existenční kvantifikátor („*existuje*“)

Predikátová logika

Formule predikátové logiky vyjadřují tvrzení o objektech, které mají nějaké vlastnosti a které jsou v určitých vzájemných vztazích.

Interpretace či **interpretační struktura** — konkrétní soubor těchto objektů, jejich vlastností a vztahů.

Universum — soubor všech objektů v dané interpretaci

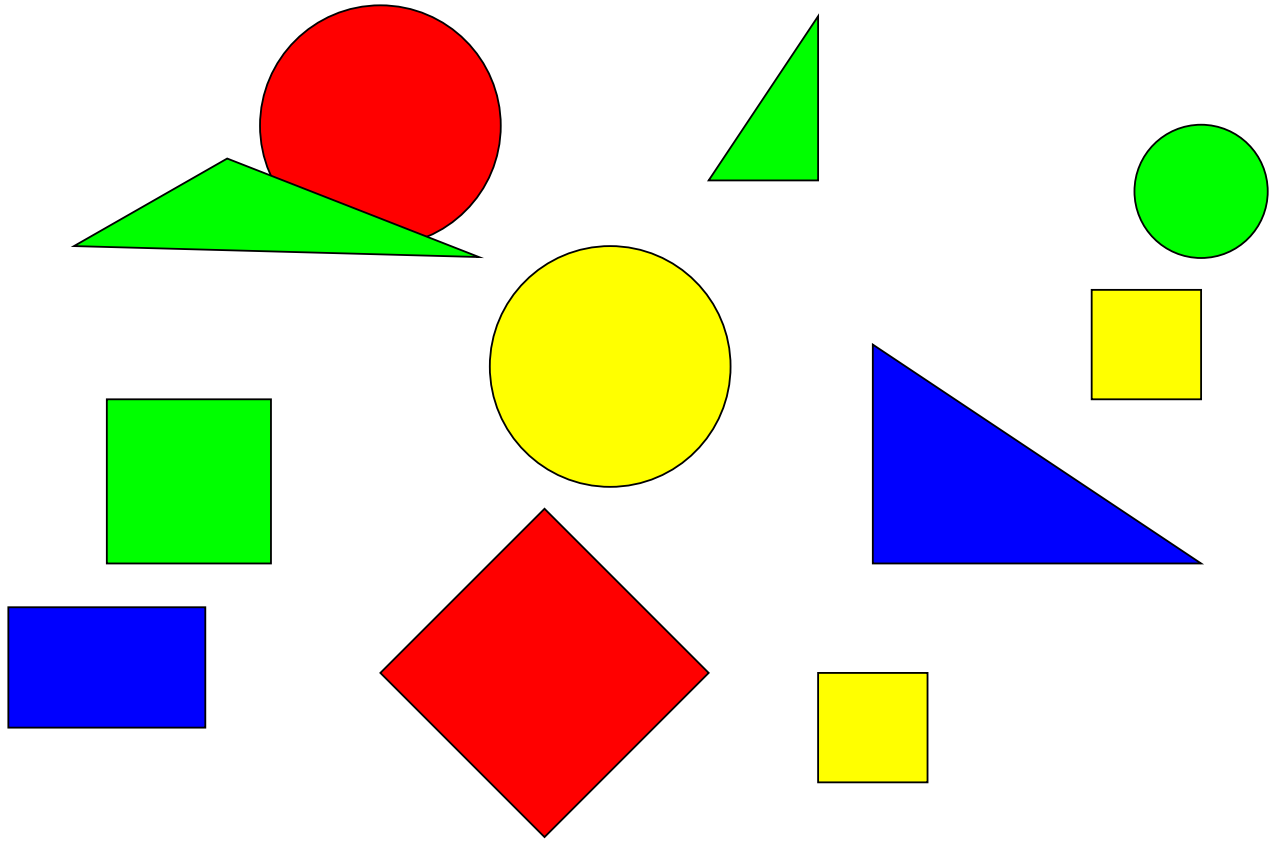
- Universem může být libovolná **neprázdná** množina.
- Objektům z tohoto universa se říká **prvky** universa.

Valuace — přiřazení prvků universa proměnným

Pravdivost formulí závisí na dané interpretaci a valuaci.

Predikátová logika

Příklad universa:



Predikátová logika

Další příklady univers:

- Nějaká přesně vymezená množina lidí, například množina všech obyvatel daného domu („*Jan Novák*“, „*František Vomáčka*“, . . .)
- Množina všech knih v dané knihovně.
- Množina přirozených čísel $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$.
- Množina všech bodů v rovině.
- Množina $\{a, b, c, d, e\}$.
- Množina $\{a\}$.

Proměnné

Proměnné — x, y, z, ..., případně s indexy — x_0, x_1, x_2, ...

Předpokládáme, že k dispozici je nekonečný počet proměnných.

Valuace — přiřazení prvků universa proměnným

Příklad:

- Universum — nějaká množina lidí; valuace v, kde:
  $v(x) =$ „*František Vomáčka*“
  $v(y) =$ „*Pavla Nováková*“
  ...
- Universum — množina přirozených čísel $\mathbb{N} = \{0, 1, 2, \ldots\}$; valuace v, kde

$$v(x) = 57 \qquad v(y) = 3 \qquad v(z) = 57 \qquad \ldots$$

Predikáty

Predikáty — P, Q, R, ...

- **Unární predikáty** — reprezentují **vlastnosti** prvků universa

  Příklad: Predikát P reprezentující vlastnost „*být modrý*“:

  $$P(x) \quad — \quad \text{„}x \text{ je modrý“}$$

  Unární predikát přiřazuje prvkům universa pravdivostní hodnoty.

  Např. hodnota $P(x)$ může být:

  - 1 — prvek přiřazený proměnné x má danou vlastnost P (tj. je modrý)
  - 0 — prvek přiřazený proměnné x nemá danou vlastnost P (tj. není modrý)

Predikáty

- **Binární predikáty** — reprezentují **vztahy** mezi dvojicemi prvků universa

  Příklad: Predikát R reprezentující vztah „*být rodičem*“:

  $$R(x, y) \quad — \quad \text{„}x \text{ je rodičem } y\text{“}$$

  Binární predikát přiřazuje pravdivostní hodnoty dvojicím prvků universa.

  Např. hodnota $R(x, y)$ může být:

  - 1 — když x a y jsou v daném vztahu (tj. x je rodičem y)
  - 0 — když x a y v daném vztahu nejsou (tj. x není rodičem y)

Predikáty

Můžeme uvažovat i predikáty libovolné jiné arity.

Například:

- **Ternární** predikát T (tj. predikát s aritou 3) reprezentující vztah mezi rodiči a dítětem:

  $$T(x, y, z)$$

  — x a y jsou rodiči dítěte z, přičemž x je jeho matka a y je jeho otec

- Na **nulární** predikáty (tj. predikáty s aritou 0) se můžeme dívat jako na atomické výroky, které se nevztahují k prvkům universa.

Formule predikátové logiky

Atomická formule — predikát aplikovaný na nějaké proměnné

Příklad:

- P — unární predikát reprezentující vlastnost „*být modrý*“
- Q — unární predikát reprezentující vlastnost „*být čtverec*“
- R — binární predikát reprezentující vztah „*překrývá*“

$$
\begin{array}{rcl}
P(x) & — & \text{„}x \text{ je modrý“} \\
P(y) & — & \text{„}y \text{ je modrý“} \\
Q(y) & — & \text{„}y \text{ je čtverec“} \\
R(z,x) & — & \text{„}z \text{ překrývá } x\text{“} \\
R(y,y) & — & \text{„}y \text{ překrývá sám sebe“}
\end{array}
$$

Poznámka: Později pojem atomické formule poněkud rozšíříme.

Formule predikátové logiky

Z jednodušších formulí je možno vytvářet složitější formule pomocí **logických spojek** ("$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$", "$\leftrightarrow$") stejně jako ve výrokové logice.

Příklad:

- P — unární predikát reprezentující vlastnost „*být modrý*“
- Q — unární predikát reprezentující vlastnost „*být čtverec*“
- R — binární predikát reprezentující vztah „*překrývá*“

„*Jestliže x je modrý čtverec nebo y nepřekrývá x, tak z není čtverec.*“

$$\big((P(x) \wedge Q(x)) \vee \neg R(y,x)\big) \rightarrow \neg Q(z)$$

Formule predikátové logiky

Z jednodušších formulí je možno vytvářet složitější formule pomocí **logických spojek** ("$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$", "$\leftrightarrow$") stejně jako ve výrokové logice.

Příklad:

- P — unární predikát reprezentující vlastnost *„být žena“*
- Q — unární predikát reprezentující vlastnost *„mít tmavé vlasy“*
- R — binární predikát reprezentující vztah *„být rodičem“*

„Jestliže x je žena s tmavými vlasy nebo y není rodičem x, tak z nemá tmavé vlasy.“

$$\big((P(x) \wedge Q(x)) \vee \neg R(y,x)\big) \rightarrow \neg Q(z)$$

Formule predikátové logiky

Z jednodušších formulí je možno vytvářet složitější formule pomocí **logických spojek** ("$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$", "$\leftrightarrow$") stejně jako ve výrokové logice.

Příklad:

- P — unární predikát reprezentující vlastnost „*být sudý*“
- Q — unární predikát reprezentující vlastnost „*být prvočíslo*“
- R — binární predikát reprezentující vztah „*být větší*“

„*Jestliže x je sudé prvočíslo nebo y není větší než x, tak z není prvočíslo.*“

$$\big((P(x) \wedge Q(x)) \vee \neg R(y,x)\big) \rightarrow \neg Q(z)$$

Kvantifikátory

Univerzální kvantifikátor — symbol "$\forall$"

Jestliže φ je formule reprezentující určité tvrzení, tak

$$\forall x \varphi$$

je formule reprezentující tvrzení

„pro každé x platí φ“.

Příklad: P — *„být čtverec“*

$$\forall x\, P(x)$$

- *„Pro každé x platí, že x je čtverec.“*
- *„Každé x je čtverec.“*
- *„Všechny prvky jsou čtverce.“*

Kvantifikátory

Příklad:

- *„Pro každé x platí, že pokud x je čtverec, tak x je zelený.“*
- *„Všechny čtverce jsou zelené.“*

$$\forall x(P(x) \rightarrow Q(x))$$

- P — *„být čtverec“* (arita 1)
- Q — *„být zelený“* (arita 1)

Kvantifikátory

Příklad:

- *„Jestliže pro každé x platí, že x je čtverec nebo x je zelený, tak pro každé y platí, že y je trojúhelník.“*
- *„Jestliže je každý objekt čtverec nebo je zelený, tak jsou všechny prvky trojúhelníky.“*

$$\forall x(P(x) \vee Q(x)) \rightarrow \forall y T(y)$$

- P — *„být čtverec“* (arita 1)
- Q — *„být zelený“* (arita 1)
- T — *„být trojúhelník“* (arita 1)

Kvantifikátory

Zásadní rozdíl mezi následujícími formulemi:

- $P(x)$ — „x *je čtverec*“

  Mluví o **jednom** konkrétním prvku přiřazeném proměnné x.

  Pravdivost tohoto tvrzení závisí na konkrétním prvku přiřazeném proměnné x, tj. na konkrétní valuaci.

- $\forall x P(x)$ — „*každé* x *je čtverec*“ (tj. „*všechny prvky jsou čtverce*“)

  Mluví o **všech** prvcích universa.

  Pravdivost tohoto tvrzení nezávisí na konkrétní valuaci.

Kvantifikátory

Příklad:

- „*Jestliže je x prvočíslo, pak je liché.*“

$$P(x) \to L(x)$$

- „*Pro každé x platí, že pokud je x prvočíslo, pak je liché*“.
(Tj. „*všechna prvočísla jsou lichá*“.)

$$\forall x(P(x) \to L(x))$$

Predikáty:

- P — „*být prvočíslo*“ (arita 1)
- L — „*být lichý*“ (arita 1)

Kvantifikátory

Příklad:

- *„Pro každé y platí, že pokud y je zelený, tak x překrývá y.“*
- *„Objekt x překrývá všechny zelené objekty.“*

$$\forall y(G(y) \rightarrow R(x, y))$$

Predikáty:

- R — *„překrývá“* (arita 2)
- G — *„je zelený“* (arita 1)

Kvantifikátory

Příklad:

- „*Pro každé x platí, že pro každé y platí, že pokud x je rodičem y, tak x má rád y.*“
- „*Pro každé x a y platí, že pokud x je rodičem y, tak x má rád y.*“
- „*Pro každé dva prvky x, y platí, že pokud x je rodičem y, tak x má rád y.*“

$$\forall x \forall y (R(x, y) \rightarrow S(x, y))$$

Predikáty:

- R — „*je rodičem*“ (arita 2)
- S — „*má rád*“ (arita 2)

Kvantifikátory

Existenční kvantifikátor — symbol "$\exists$"

Jestliže φ je formule reprezentující určité tvrzení, tak

$$\exists x \varphi$$

je formule reprezentující tvrzení

„existuje x, pro které platí φ“.

Příklad: P — *„být čtverec“*

$$\exists x\, P(x)$$

- *„Existuje x, pro které platí, že x je čtverec.“*
- *„Existuje x takové, že x je čtverec.“*
- *„Existuje alespoň jeden čtverec.“*

Kvantifikátory

Příklad:

- „*Existuje x, pro které platí, že x je čtverec a x je zelený.*“
- „*Existuje x takové, že x je čtverec a x je zelený.*“
- „*Pro nějaké x platí, že x je čtverec a x je zelený.*“
- „*Existuje zelený čtverec.*“
- „*Některé čtverce jsou zelené.*“
- „*Alespoň jedno x je zelený čtverec.*“

$$\exists x(P(x) \wedge Q(x))$$

Predikáty:

- P — „*být čtverec*“ (arita 1)
- Q — „*být zelený*“ (arita 1)

Kvantifikátory

Příklad:

- *„Existuje x takové, že pro každé y je x větší než y.“*

$$\exists x \forall y \, P(x, y)$$

- *„Pro každé y existuje x takové, že x je větší než y.“*

$$\forall y \exists x \, P(x, y)$$

P — *„být větší než“* (arita 2)

Syntaxe formulí predikátové logiky

Abeceda:

- **logické spojky** — "$\neg$", "$\wedge$", "$\vee$" "$\rightarrow$" a "$\leftrightarrow$"
- **kvantifikátory** — "$\forall$", "$\exists$"
- **pomocné symboly** — "$($", "$)$", "$,$"
- **proměnné** — "x", "y", "z", ..., "x_0", "x_1", "x_2", ...

- **predikátové symboly** — například symboly "P", "Q", "R", apod. (u každého symbolu musí být navíc stanovena příslušná arita)
- ...

Poznámka: Další typy symbolů budou uvedeny později.

Syntaxe formulí predikátové logiky

Definice

Dobře utvořené **atomické formule** predikátové logiky jsou formule tvaru:

- $P(x_1, x_2, \ldots, x_n)$, kde P je predikátový symbol s aritou n a $x_1, x_2, \ldots, x_n$ jsou (ne nutně různé) proměnné.
- ...

Poznámka: Tato definice není úplná. Později bude poněkud zobecněna a rozšířena o další položky.

Příklad:

$$P(x, y) \qquad R(z, z, z) \qquad S(y)$$

Syntaxe formulí predikátové logiky

Definice

Dobře utvořené **formule predikátové logiky** jsou posloupnosti symbolů vytvořené podle následujících pravidel:

1. Dobře utvořené atomické formule jsou dobře utvořené formule.
2. Jestliže φ a ψ jsou dobře utvořené formule, pak i $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$ a $(\varphi \leftrightarrow \psi)$ jsou dobře utvořené formule.
3. Jestliže φ je dobře utvořená formule a x je proměnná, tak $\forall x \varphi$ a $\exists x \varphi$ jsou dobře utvořené formule.
4. Neexistují žádné další dobře utvořené formule než ty, které jsou vytvořené pomocí předchozích pravidel.

Syntaxe formulí predikátové logiky

Pojmy jako

- podformule
- abstraktní syntaktický strom

jsou zavedeny v predikátové logice podobně jako ve výrokové logice (pouze rozšířeny o konstrukce, které jsou v predikátové logice navíc).

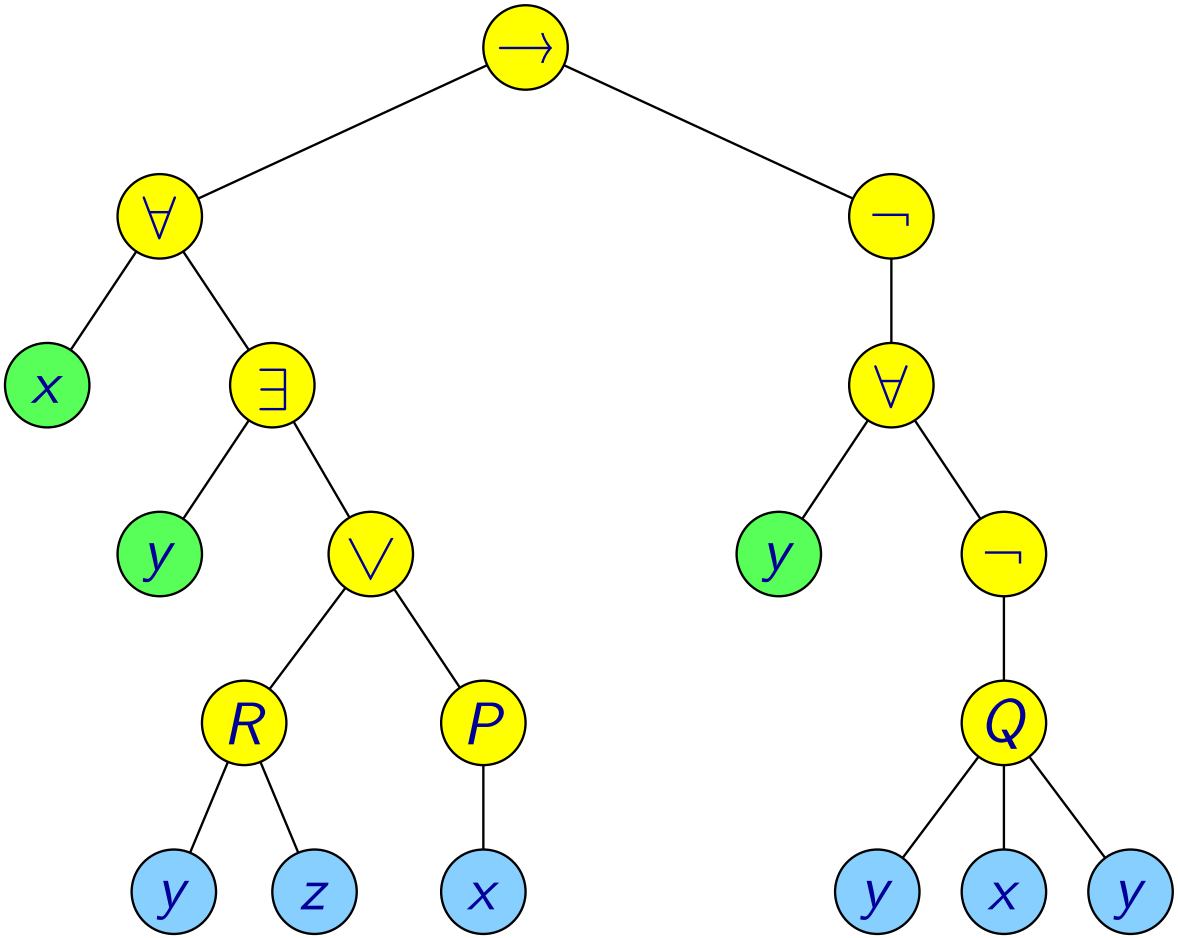
Konvence pro vypouštění závorek:

- Stejné jako ve výrokové logice.
- Kvantifikátory ("$\forall$" a "$\exists$") mají stejnou prioritu jako negace ("$\neg$"), tj. nejvyšší prioritu.

Syntaxe formulí predikátové logiky

Abstraktní syntaktický strom formule

$$\forall x \exists y (R(y,z) \vee P(x)) \rightarrow \neg \forall y \neg Q(y,x,y)$$



Volné a vázané výskyty proměnných

Každý výskyt proměnné x v podformuli tvaru $\exists x\varphi$ nebo $\forall x\varphi$ je **vázaný**.

Výskyt proměnné, který není vázaný, je **volný**.

Příklad: Formule

$$\forall x \exists y (R(y,z) \vee P(x)) \rightarrow \neg \forall y \neg Q(y,x,y)$$

- y v podformuli $R(y,z)$ — vázaný výskyt $(\exists y)$
- z v podformuli $R(y,z)$ — volný výskyt
- x v podformuli $P(x)$ — vázaný výskyt $(\forall x)$
- oba výskyty y v podformuli $Q(y,x,y)$ — vázané výskyty $(\forall y)$
- x v podformuli $Q(y,x,y)$ — volný výskyt

Volné a vázané výskyty proměnných

Množinu proměnných, které se vyskytují jako **volné** proměnné ve formuli φ, budeme označovat $\mathit{free}(\varphi)$.

Příklad:

- Pokud φ je formule $P(x, y)$, tak $\mathit{free}(\varphi) = \{x, y\}$.
- Pokud ψ je formule $\exists x \exists y P(x, y)$, tak $\mathit{free}(\psi) = \emptyset$.
- Pokud χ je formule

$$\forall x \exists y (R(y, z) \vee P(x)) \rightarrow \neg \forall y \neg Q(y, x, y)$$

tak $\mathit{free}(\chi) = \{x, z\}$.

Volné a vázané výskyty proměnných

Množinu volných proměnných $\mathit{free}(\varphi)$ je možné popsat následující induktivní definicí:

- $\mathit{free}(P(x_1, x_2, \ldots, x_n)) = \{x_1, x_2, \ldots, x_n\}$
  (kde P je predikátový symbol)
- $\mathit{free}(\neg\varphi) = \mathit{free}(\varphi)$
- $\mathit{free}(\varphi \wedge \psi) = \mathit{free}(\varphi) \cup \mathit{free}(\psi)$
  (pro formule tvaru $\varphi \vee \psi$, $\varphi \rightarrow \psi$ a $\varphi \leftrightarrow \psi$ je to podobné)
- $\mathit{free}(\forall x \varphi) = \mathit{free}(\varphi) - \{x\}$ (kde x je proměnná)
- $\mathit{free}(\exists x \varphi) = \mathit{free}(\varphi) - \{x\}$ (kde x je proměnná)

Sémantika predikátové logiky

Formule jsou vyhodnocovány při dané **interpretaci** (**interpretační struktuře**) a **valuaci**.

To, že formule φ je pravdivá (tj. má pravdivostní hodnotu 1) v interpretaci $\mathcal{A}$ při valuaci v, budeme označovat

$$\mathcal{A}, v \models \varphi$$

To, že formule φ je nepravdivá (má pravdivostní hodnotu 0) v interpretaci $\mathcal{A}$ při valuaci v, budeme označovat $\mathcal{A}, v \not\models \varphi$.

Sémantika predikátové logiky

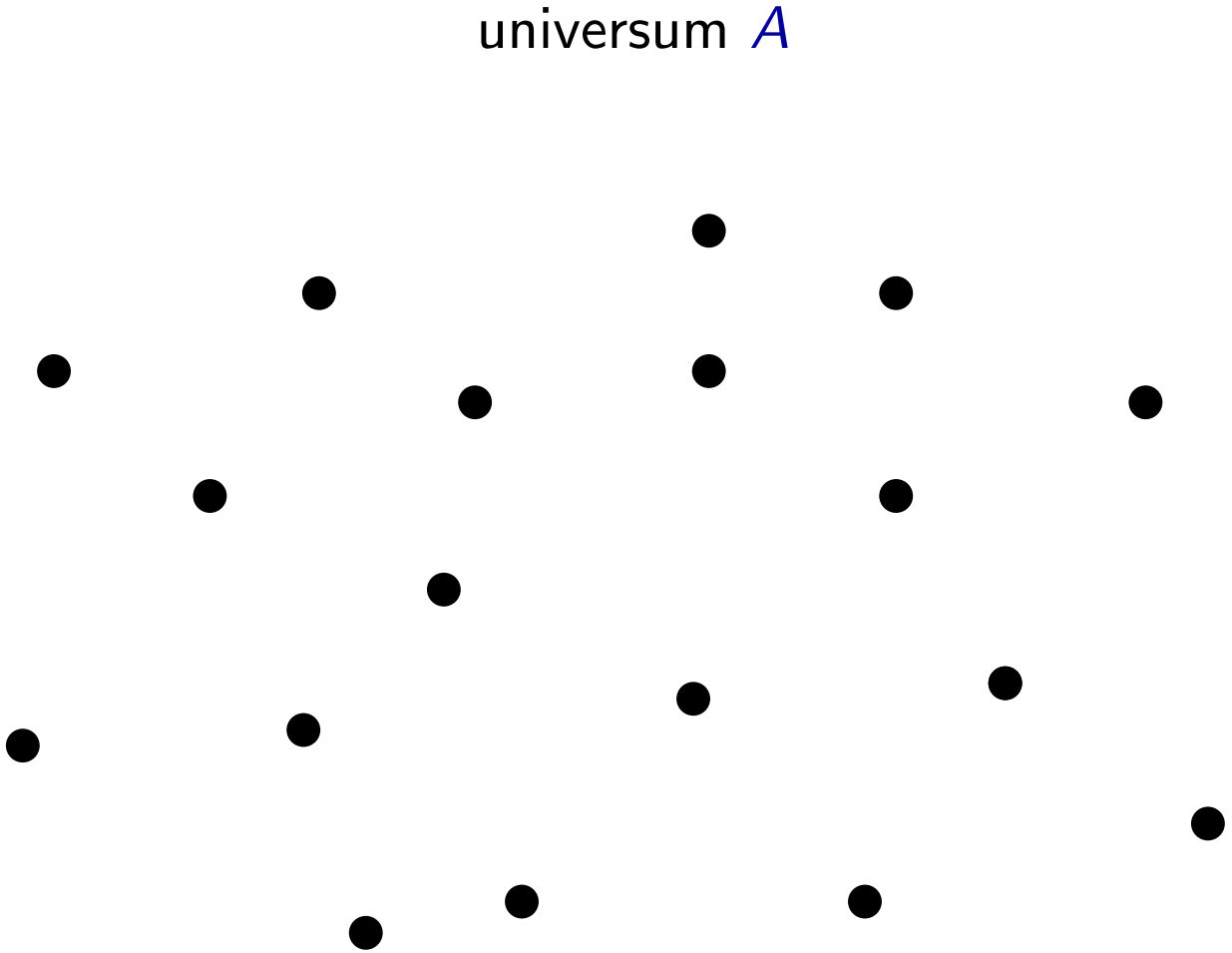
Interpretace $\mathcal{A}$ je struktura skládající se z následujících položek:

- **Universum** A — libovolná neprázdná množina
- Každému unárnímu predikátovému symbolu P je přiřazena podmnožina množiny A (tj. unární relace na A) — označme ji $P^{\mathcal{A}}$.
  (Platí tedy $P^{\mathcal{A}} \subseteq A$.)
- Každému binárnímu predikátovému symbolu Q je přiřazena binární relace na A — označme ji $Q^{\mathcal{A}}$.
  (Platí tedy $Q^{\mathcal{A}} \subseteq A \times A$.)
- Podobné to bude pro predikátové symboly s dalšími aritami (3, 4, 5, ...).

Poznámka: Tato definice není úplná a bude později doplněna o další položky.

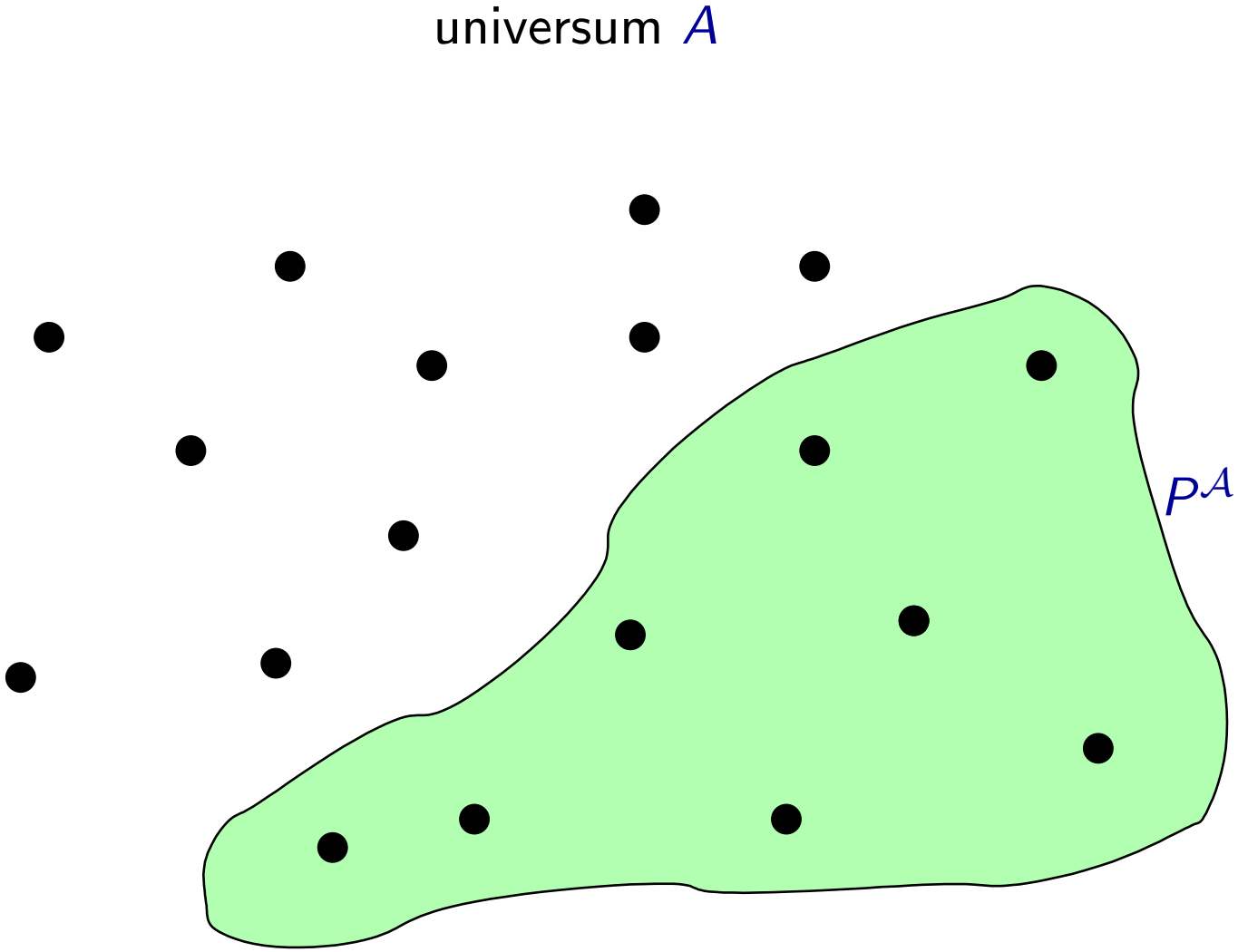
Sémantika predikátové logiky

Příklad interpretace $\mathcal{A}$:



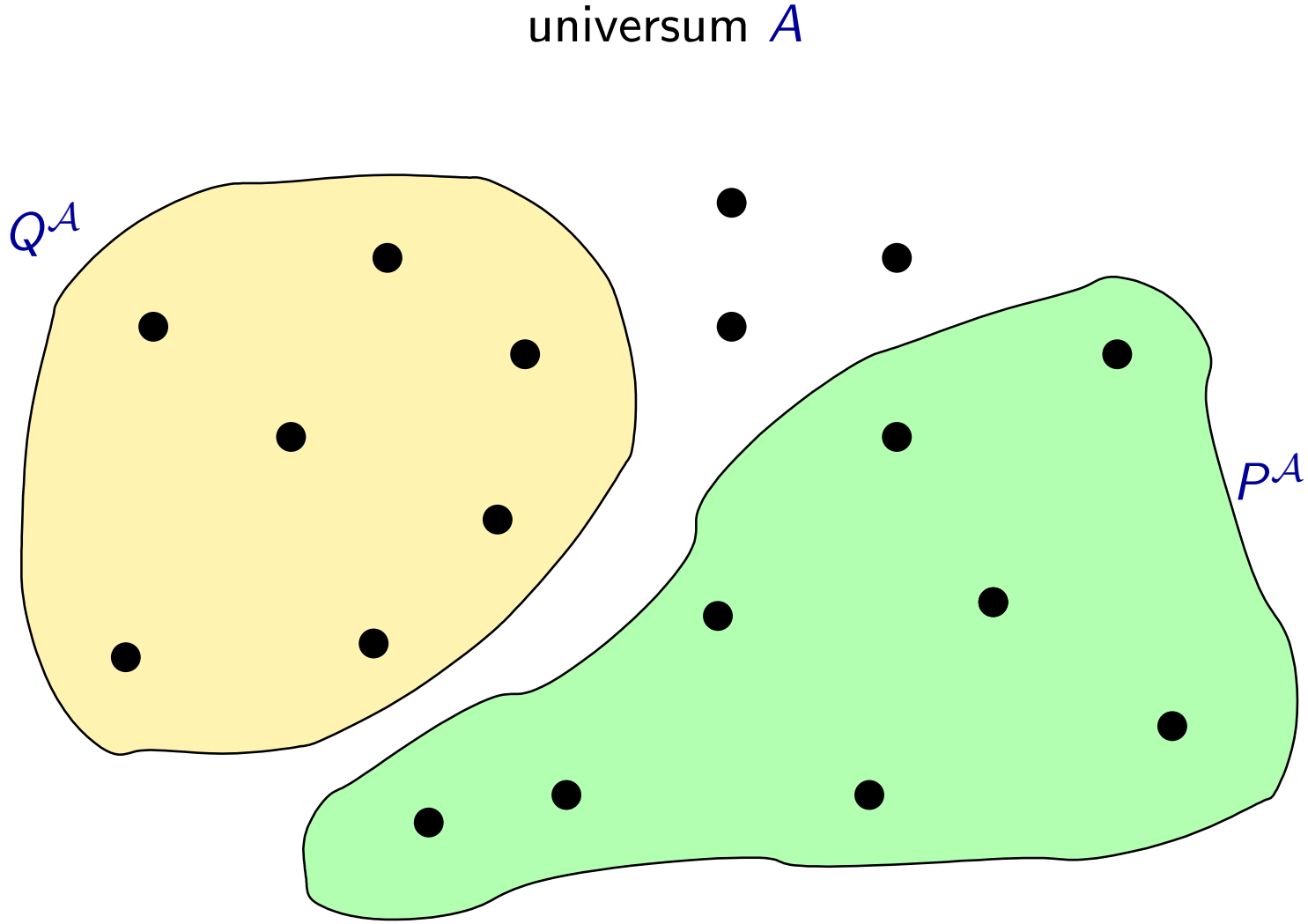
Sémantika predikátové logiky

Příklad interpretace $\mathcal{A}$:



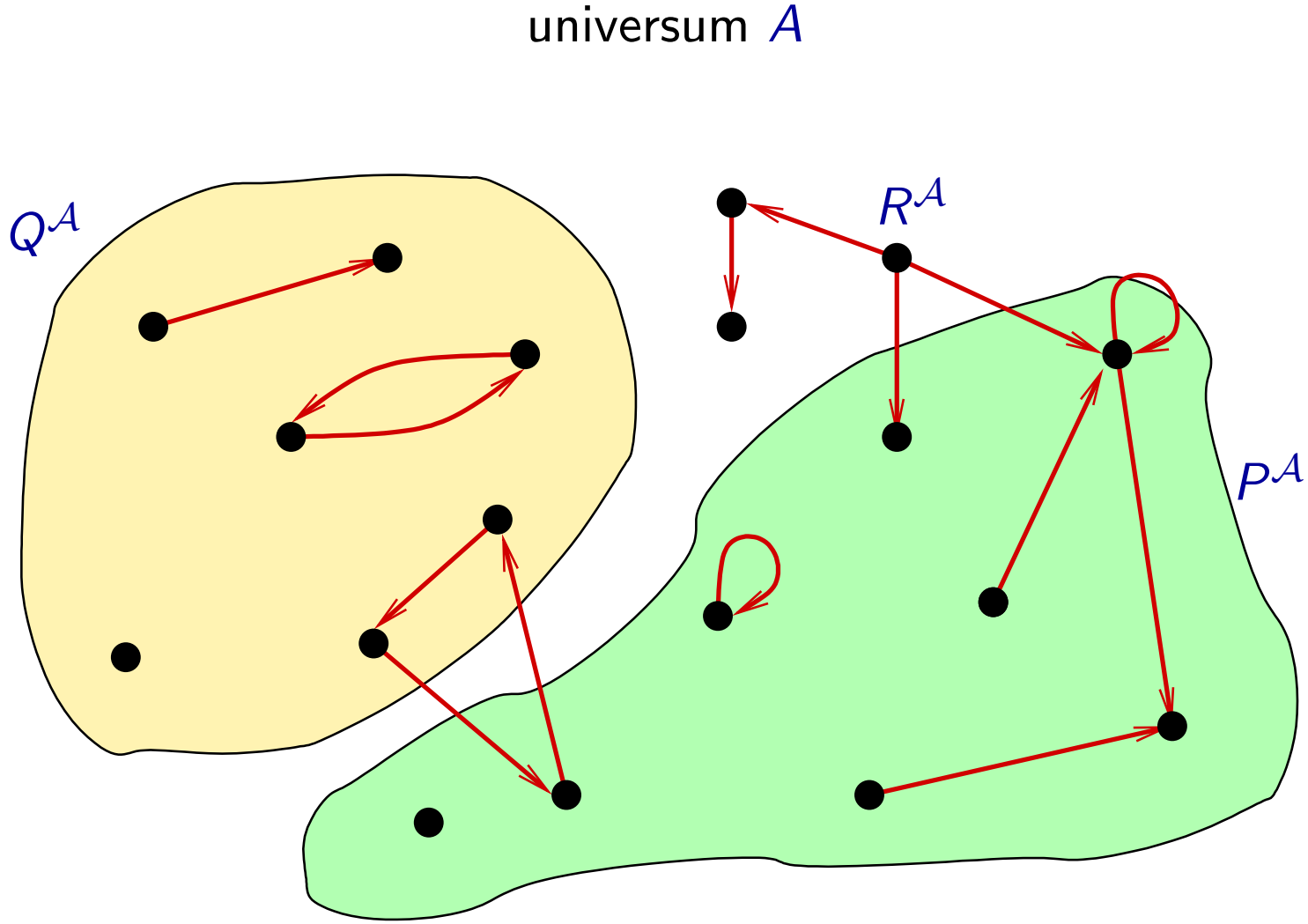
Sémantika predikátové logiky

Příklad interpretace $\mathcal{A}$:



Sémantika predikátové logiky

Příklad interpretace $\mathcal{A}$:



Sémantika predikátové logiky

Jiný příklad interpretace $\mathcal{A}$:

- universum $A = \{a, b, c, d, e, f, g\}$
- $P^{\mathcal{A}} = \{b, d, e\}$
- $Q^{\mathcal{A}} = \{a, b, e, g\}$
- $R^{\mathcal{A}} = \{(a, b), (a, e), (a, g), (b, b), (c, e), (f, c), (f, g), (g, a), (g, g)\}$

Sémantika predikátové logiky

Označme množinu všech proměnných Var, tj.

$$Var = \{x, y, z, \ldots, x_0, x_1, x_2, \ldots\}$$

Při dané interpretaci $\mathcal{A}$ s universem A je **valuace** v libovolná funkce

$$v : Var \to A$$

přiřazující jednotlivým proměnným prvky universa.

Poznámka: Jak uvidíme, ve skutečnosti jsou pro určení pravdivosti formule φ důležité hodnoty přiřazené valuací v proměnným z množiny $free(\varphi)$.

Hodnoty přiřazené valuací v ostatním proměnným z tohoto hlediska důležité nejsou.

Sémantika predikátové logiky

Vezměme si libovolnou interpretaci $\mathcal{A}$ s universem A a libovolnou valuaci v.

Řekněme, že x je proměnná (tj. $x \in Var$) a a je prvek universa (tj. $a \in A$).

Zápis

$$v[x \mapsto a]$$

označuje valuaci $v' : Var \to A$, která se s valuací v shoduje v hodnotách všech proměnných jiných než x, a kde x nabývá hodnoty a.

Tj. pro každou proměnnou y (kde $y \in Var$) platí

$$v'(y) = \begin{cases} a & \text{pokud } y = x \\ v(y) & \text{jinak} \end{cases}$$

Sémantika predikátové logiky

Příklad:

- universum $A = \{a, b, c, d, e, f, g, \ldots\}$

  valuace v:

  $v(x_0) = c \qquad v(x_1) = e \qquad v(x_2) = b \qquad v(x_3) = e \qquad \ldots$

  valuace $v[x_2 \mapsto g]$:

  $v(x_0) = c \qquad v(x_1) = e \qquad v(x_2) = g \qquad v(x_3) = e \qquad \ldots$

Sémantika predikátové logiky

Definice

Předpokládejme danou interpretaci $\mathcal{A}$ s universem A a valuaci v, přiřazující proměnným prvky z universa A.

Pravdivost formulí predikátové logiky v interpretaci $\mathcal{A}$ a valuaci v je definována následovně:

- Pro predikát P s aritou n platí $\mathcal{A}, v \models P(x_1, x_2, \ldots, x_n)$ právě tehdy, když $(v(x_1),\ v(x_2),\ \ldots\ , v(x_n)) \in P^{\mathcal{A}}$.
- $\mathcal{A}, v \models \neg\varphi$ právě tehdy, když $\mathcal{A}, v \not\models \varphi$.
- $\mathcal{A}, v \models \varphi \wedge \psi$ právě tehdy, když $\mathcal{A}, v \models \varphi$ a $\mathcal{A}, v \models \psi$.
- $\mathcal{A}, v \models \varphi \vee \psi$ právě tehdy, když $\mathcal{A}, v \models \varphi$ nebo $\mathcal{A}, v \models \psi$.
- $\mathcal{A}, v \models \varphi \rightarrow \psi$ právě tehdy, když $\mathcal{A}, v \not\models \varphi$ nebo $\mathcal{A}, v \models \psi$.
- $\mathcal{A}, v \models \varphi \leftrightarrow \psi$ právě tehdy, když $\mathcal{A}, v \models \varphi$ a $\mathcal{A}, v \models \psi$, nebo když $\mathcal{A}, v \not\models \varphi$ a $\mathcal{A}, v \not\models \psi$.
- ...

Sémantika predikátové logiky

Definice (pokrač.)

- ...
- $\mathcal{A}, v \models \forall x \varphi$ právě tehdy, když pro **každé** $a \in A$ platí $\mathcal{A}, v[x \mapsto a] \models \varphi$.
- $\mathcal{A}, v \models \exists x \varphi$ právě tehdy, když **existuje** $a \in A$ takové, že $\mathcal{A}, v[x \mapsto a] \models \varphi$.

Poznámka: Ty interpretace, ve kterých daná formule platí, se označují jako její **modely**.

Vyhodnocování pravdivosti formulí jako hra

Vezměme si formuli tvaru

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \exists x_{n-1} \forall x_n \varphi,$$

kde se nějak libovolně střídají kvantifikátory a kde φ neobsahuje žádné kvantifikátory.

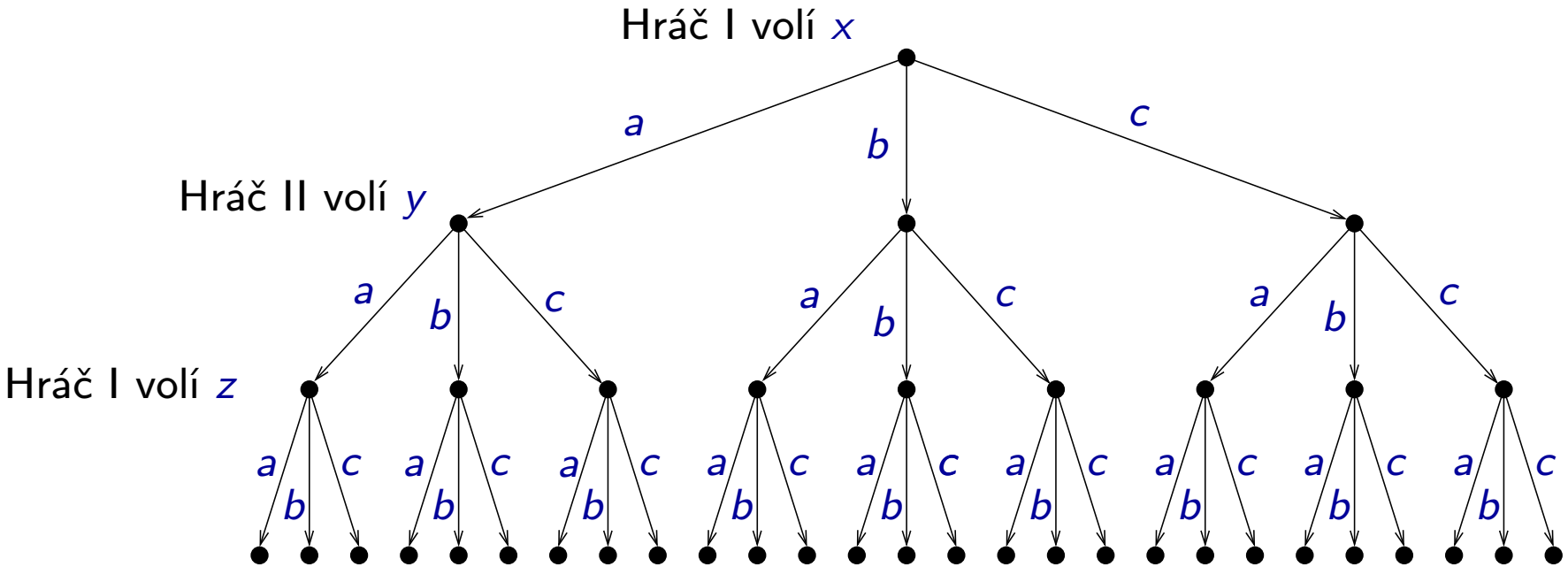
Na vyhodnocování pravdivosti formulí tohoto tvaru v dané interpretaci $\mathcal{A}$ a valuaci v lze nahlížet jako na určitý druh hry:

- Hrají dva hráči — **Hráč I** a **Hráč II**.
- Hráč I se snaží ukázat, že formule platí.
- Hráč II se snaží ukázat, že formule neplatí.
- Hráč I volí hodnoty proměnných vázaných existenčním kvantifikátorem ($\exists$).
- Hráč II volí hodnoty proměnných vázaných univerzálním kvantifikátorem ($\forall$).

Vyhodnocování pravdivosti formulí jako hra

Příklad: Formule $\exists x \forall y \exists z (P(x, y) \rightarrow Q(y, z))$

universum $A = \{a, b, c\}$



Vyhodnocování pravdivosti formulí jako hra

- Formule φ platí právě tehdy, když má Hráč I v této hře **vítěznou strategii**.
- Formule φ neplatí právě tehdy, když má vítěznou strategii Hráč II.

Strategie — určuje, jak má hráč hrát v každé situaci, tj. určuje tahy pro všechny možné tahy protihráče.

Vítězná strategie — strategie, která zaručí vítězství daného hráče bez ohledu na to, jak hraje protihráč.

Vyhodnocování pravdivosti formulí jako hra

Příklad: Interpretace, kde universum je množina reálných čísel $\mathbb{R}$ a binární predikátový symbol R reprezentuje relaci *„větší nebo rovno“* (tj. $R(x, y)$ platí právě tehdy, když $x \geq y$).

Formule $\exists x \forall y R(x, y)$ — vítězná strategie Hráče II:

- Hráč I zvolí číslo x.
- Hráč II zvolí číslo $y = x + 1$ — Hráč II vyhrál, protože zjevně není pravda, že $x \geq x + 1$.

Formule $\forall y \exists x R(x, y)$ — vítězná strategie Hráče I:

- Hráč II zvolí číslo y.
- Hráč I zvolí číslo $x = y$ — Hráč I vyhrál, protože zjevně platí, že $x \geq x$.

Logicky platné formule

Formule φ je **logicky platná**, jestliže je pravdivá v každé interpretaci a valuaci, tj. jestliže pro každou interpretaci $\mathcal{A}$ a valuaci v platí

$$\mathcal{A}, v \models \varphi.$$

Příklad:

- $\exists x P(x) \rightarrow \exists y P(y)$
- $\forall x P(x) \wedge \neg \exists y Q(y) \rightarrow \forall z (P(z) \wedge \neg Q(z))$
- $\forall x P(x) \rightarrow \exists x P(x)$

Logicky platné formule

Pokud vezmeme libovolnou tautologii výrokové logiky a nahradíme v ní atomické výroky libovolnými formulemi predikátové logiky, dostaneme logicky platnou formuli.

Příklad: Tautologie $p \rightarrow (q \vee p)$

- p nahradíme $\forall z(P(x,z) \leftrightarrow \neg Q(z,y))$
- q nahradíme $R(x)$

Dostaneme logicky platnou formuli

$$\forall z(P(x,z) \leftrightarrow \neg Q(z,y)) \rightarrow (R(x) \vee \forall z(P(x,z) \leftrightarrow \neg Q(z,y)))$$

Logicky ekvivalentní formule

Formule φ a ψ jsou **logicky ekvivalentní**, jestliže mají stejné pravdivostní hodnoty v každé interpretaci a valuaci, tj. jestliže pro každou interpretaci $\mathcal{A}$ a valuaci v platí

$$\mathcal{A}, v \models \varphi \qquad \text{právě tehdy, když} \qquad \mathcal{A}, v \models \psi.$$

To, že φ a ψ jsou logicky ekvivalentní, se označuje zápisem

$$\varphi \Leftrightarrow \psi.$$

- Stejně jako ve výrokové logice, je v predikátové logice možné provádět ekvivalentní úpravy.
- Všechny ekvivalence, které platí ve výrokové logice, platí i v predikátové logice.

Logicky ekvivalentní formule

- V predikátové logice platí řada dalších ekvivalencí, které nemají analogii ve výrokové logice.

Příklady některý důležitých ekvivalencí:

$$\neg\forall x\varphi \Leftrightarrow \exists x\neg\varphi$$
$$\neg\exists x\varphi \Leftrightarrow \forall x\neg\varphi$$

$$\forall x\forall y\varphi \Leftrightarrow \forall y\forall x\varphi$$
$$\exists x\exists y\varphi \Leftrightarrow \exists y\exists x\varphi$$

Pokud $x \notin \mathit{free}(\varphi)$:

$$\forall x\varphi \Leftrightarrow \varphi$$
$$\exists x\varphi \Leftrightarrow \varphi$$

Logicky ekvivalentní formule

Některé další důležité ekvivalence:

$$(\forall x \varphi) \wedge (\forall x \psi) \Leftrightarrow \forall x (\varphi \wedge \psi)$$
$$(\exists x \varphi) \vee (\exists x \psi) \Leftrightarrow \exists x (\varphi \vee \psi)$$

Pokud $x \notin \mathit{free}(\psi)$:

$$(\forall x \varphi) \wedge \psi \Leftrightarrow \forall x (\varphi \wedge \psi)$$
$$(\forall x \varphi) \vee \psi \Leftrightarrow \forall x (\varphi \vee \psi)$$
$$(\exists x \varphi) \wedge \psi \Leftrightarrow \exists x (\varphi \wedge \psi)$$
$$(\exists x \varphi) \vee \psi \Leftrightarrow \exists x (\varphi \vee \psi)$$

Přejmenování vázaných proměnných

Pokud přejmenujeme ve formuli vázanou proměnnou, dostaneme ekvivalentní formuli.

Příklad: $\forall x P(x, y) \Leftrightarrow \forall z P(z, y)$

- Pokud přejmenováváme např. x na y ve formuli $\forall x \varphi$ nebo $\exists x \varphi$, proměnná y se **nesmí** vyskytovat ve formuli φ jako volná proměnná.

$$\exists x P(x, y) \qquad \text{není ekvivalentní} \qquad \exists y P(y, y)$$

- Při přejmenování se volné výskyty proměnných v podformuli **nesmí** stát vázanými. Např.

$$\exists x \forall y P(x, y) \qquad \text{není ekvivalentní} \qquad \exists y \forall y P(y, y)$$

Substituce

Řekněme, že ve formuli φ chceme nahradit **volné** výskyty proměnné x proměnnou y (tj. za x dosadit y).

Tato operace na formulích se nazývá **substituce** a výsledná formule se označuje

$$\varphi[y/x].$$

Poznámka: Formule φ a $\varphi[y/x]$ obecně **nejsou** ekvivalentní.

Příklad:

$$P(x,z) \qquad \text{není ekvivalentní} \qquad P(y,z)$$

Přejmenování vázaných proměnných

S použitím operace substituce je možné popsat přejmenování vázaných proměnných pomocí následujících ekvivalencí.

Pokud $y \notin \mathit{free}(\forall x \varphi)$:

$$\forall x \varphi \quad \Leftrightarrow \quad \forall y (\varphi[y/x])$$

Pokud $y \notin \mathit{free}(\exists x \varphi)$:

$$\exists x \varphi \quad \Leftrightarrow \quad \exists y (\varphi[y/x])$$

Příklad:

$$\exists x \forall y P(x, y) \quad \Leftrightarrow \quad \exists x \forall z P(x, z) \quad \Leftrightarrow \quad \exists y \forall z P(y, z) \quad \Leftrightarrow \quad \exists y \forall x P(y, x)$$

Logické vyplývání

Definice

Závěr ψ **logicky vyplývá** z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$, což zapisujeme

$$\varphi_1, \varphi_2, \ldots, \varphi_n \models \psi,$$

jestliže v každé interpretaci $\mathcal{A}$ a každé valuaci v, kde platí předpoklady $\varphi_1, \varphi_2, \ldots, \varphi_n$, platí i závěr ψ.

- Vše, co bylo řečeno o logickém vyplývání ve výrokové logice, platí analogicky i v predikátové logice.

Logické vyplývání

Pokud chceme ukázat, že daný závěr ψ z předpokladů $\varphi_1, \varphi_2, \ldots, \varphi_n$ **nevyplývá**, stačí najít příklad jedné konkrétní interpretace $\mathcal{A}$ a valuace v, kde platí tyto předpoklady a neplatí závěr ψ.

Příklad:

- *Existuje vodní živočich živící se masem.*
- *Všechny ryby jsou vodní živočichové.*

- *Existuje ryba živící se masem.*

$$\frac{\exists x(P(x) \wedge Q(x)) \qquad \forall x(R(x) \to P(x))}{\exists x(R(x) \wedge Q(x))}$$

$P(x)$ — „x *je vodní živočich*“
$Q(x)$ — „x *se živí masem*“
$R(x)$ — „x *je ryba*“

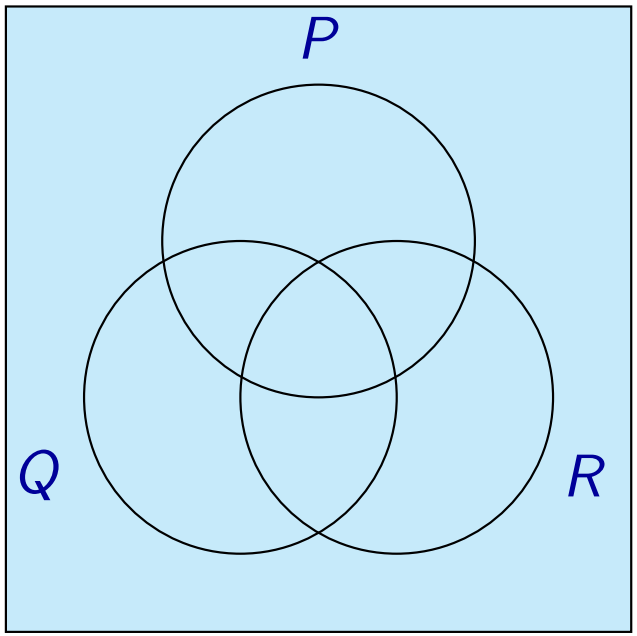
Interpretace $\mathcal{A}$, kde universum $A = \{a, b\}$

$$P^{\mathcal{A}} = \{a, b\} \qquad Q^{\mathcal{A}} = \{a\} \qquad R^{\mathcal{A}} = \{b\}$$

Vennovy diagramy

Obecně je těžké zjistit, zda závěr vyplývá nebo nevyplývá z daných předpokladů.

V případě, kdy máme pouze unární predikáty a těchto predikátů je jen malý počet (např. 3), lze při úvahách pro názornost použít tzv. **Vennovy diagramy**.



Vennovy diagramy

Příklad:

- *Ryby jsou obratlovci.*
- *Ryby žijí ve vodě.*
- *Existuje alespoň jedna ryba.*

- *Existuje obratlovec žijící ve vodě.*

$$\frac{\begin{array}{l}\forall x(P(x) \to Q(x)) \\ \forall x(P(x) \to R(x)) \\ \exists x P(x)\end{array}}{\exists x(Q(x) \wedge R(x))}$$

$P(x)$ — „x *je ryba*“
$Q(x)$ — „x *je obratlovec*“
$R(x)$ — „x *žije ve vodě*“

⟨*řešení na tabuli*⟩

Příklad důkazu

$$\frac{\begin{array}{l}\forall x(\neg R(x,x)) \\ \forall x\forall y\forall z(R(x,y) \wedge R(y,z) \rightarrow R(x,z))\end{array}}{\forall x\forall y(R(x,y) \rightarrow \neg R(y,x))}$$

1.	$\forall x(\neg R(x,x))$	- předpoklad 1
2.	$\forall x\forall y\forall z(R(x,y) \wedge R(y,z) \rightarrow R(x,z))$	- předpoklad 2
3.	Předpokládejme libovolné prvky x a y:	
4.	Předpokládejme $R(x,y)$:	
5.	Předpokládejme $R(y,x)$:	
6.	$R(x,y) \wedge R(y,x) \rightarrow R(x,x)$	- z 2.
7.	$R(x,x)$	- z 4., 5., 6.
8.	$\neg R(x,x)$	- z 1.
9.	$\neg R(y,x)$	- spor 7. a 8., takže 5. neplatí
10.	$R(x,y) \rightarrow \neg R(y,x)$	- z 4., 9.
11.	$\forall x\forall y(R(x,y) \rightarrow \neg R(y,x))$	- z 3., 10.

Rovnost

Jedním z nejdůležitějších druhů relací je **rovnost** (**identita**).

Prvky x a y jsou si rovny, což zapisujeme

$$x = y,$$

jestliže se jedná o jeden a tentýž prvek.

Rovnost lze vyjádřit jako predikát, např. můžeme zvolit, že $P(x, y)$ reprezentuje tvrzení „x *je rovno* y“.

V různých interpretacích ale může platit $P(x, y)$ i pro navzájem různé prvky x a y nebo naopak pro nějaký prvek x nemusí platit $P(x, x)$ apod.

Rovnost

Příklad: Chtěli bychom formulí popsat vztah „x *je sourozencem* y“ pomocí binárního predikátu R, kde $R(x, y)$ znamená „x *je rodičem* y“.

Pokus o řešení:

„x *je sourozencem* y“

právě tehdy, když

$$\exists z(R(z, x) \wedge R(z, y))$$

Problém: Pokud pro dané x existuje prvek z takový, že $R(z, x)$, tak bude platit

$$\exists z(R(z, x) \wedge R(z, x)),$$

takže bude platit „x *je sourozencem* x“.

Rovnost

Abeceda:

- ...
- Symbol pro rovnost: "$=$"
- ...

Atomické formule (pokrač.)

- ...
- Jestliže x a y jsou proměnné, tak $x = y$ je dobře utvořená atomická formule.
- ...

V každé interpretaci je symbol "$=$" interpretován jako rovnost, tj. v každé interpretaci $\mathcal{A}$ a valuaci v:

- $\mathcal{A}, v \models x = y$ právě tehdy, když $v(x) = v(y)$.

Rovnost

Příklad: Vztah „x *je sourozencem* y“ je možné vyjádřit formulí

$$\neg(x = y) \wedge \exists z(R(z, x) \wedge R(z, y)),$$

kde $R(x, y)$ znamená, že „x *je rodičem* y“.

Poznámka: Místo $\neg(x = y)$ se často používá zápis $x \neq y$.

Rovnost

- „*Existuje* **právě jedno** x *takové, že* $P(x)$“:

$$\exists x(P(x) \wedge \forall y(P(y) \rightarrow x = y))$$

- „*Existují* **alespoň dva** *prvky* x *takové, že* $P(x)$“:

$$\exists x \exists y(P(x) \wedge P(y) \wedge \neg(x = y))$$

- „*Existují* **právě dva** *prvky* x *takové, že* $P(x)$“:

$$\exists x \exists y(P(x) \wedge P(y) \wedge \neg(x = y) \wedge \forall z(P(z) \rightarrow (z = x \vee z = y)))$$

- „*Existuje* **právě jedno** x *takové, že pro něj platí* φ“:

$$\exists x(\varphi \wedge \forall y(\varphi[y/x] \rightarrow x = y))$$

Konstanty

Někdy chceme mluvit o nějakém konkrétním prvku universa.

Příklad: *„Existuje alespoň jedno x takové, že Jan Novák je rodičem x a x je žena.“* (Tj. *„Jan Novák má alespoň jednu dceru.“*)

Pokud je proměnné y přiřazen *„Jan Novák“*:

$$\exists x(R(y,x) \wedge S(x))$$

- $R(x,y)$ — *„x je rodičem y“*
- $S(x)$ — *„x je žena“*

Mohli bychom zavést unární predikát N reprezentující vlastnost *„být Jan Novák“*:

$$\forall y(N(y) \rightarrow \exists x(R(y,x) \wedge S(x)))$$

Konstanty

Pokud máme nějaký unární predikát N, kde nás zajímají jen ty interpretace, kde existuje **právě jeden** prvek x, pro který platí $N(x)$, může být výhodné mít možnost tento prvek pojmenovat a obejít se tak bez predikátu N.

K tomuto účelu slouží **konstantní symboly** (**konstanty**).

Abeceda:

- ...
- konstantní symboly: "a", "b", "c", "d", ...
- ...

Konstanty

V **atomických formulích** se konstanty mohou vyskytovat na stejných místech jako proměnné:

$$P(c, x) \qquad Q(d) \qquad R(a, a) \qquad x = a$$

- Konstanty se **nesmí** používat v kvantifikátorech — např. $\exists c P(x, c)$ **není** dobře utvořená formule.

Hodnoty přiřazené konstantním symbolům jsou dány příslušnou **interpretací**:

- Daná interpretace $\mathcal{A}$ (s universem A) přiřazuje každému konstantnímu symbolu c nějaký prvek universa A.

  Označme tento prvek $c^{\mathcal{A}}$. Platí tedy $c^{\mathcal{A}} \in A$.

Konstanty

Příklad: *„Existuje alespoň jedno x takové, že Jan Novák je rodičem x a x je žena.“*

$$\exists x(R(a,x) \wedge S(x))$$

- $R(x,y)$ — *„x je rodičem y“*
- $S(x)$ — *„x je žena“*
- a — konstatní symbol reprezentující *„Jana Nováka“*

Konstanty

Příklad: *„Každé prvočíslo je větší než jedna.“*

$$\forall x(P(x) \rightarrow R(x, e))$$

- $P(x)$ — *„x je prvočíslo“*
- $R(x, y)$ — *„x je větší než y“*
- e — konstatní symbol reprezentující hodnotu 1

Funkce

Binární relace R je (unární) **funkce**, jestliže pro každé x existuje nejvýše jedno y takové, že

$$(x, y) \in R.$$

Tato funkce je **totální**, jestliže pro každé x existuje právě jedno takové y.

Příklad: Binární relace R na množině přirozených čísel $\mathbb{N}$, kde

$$(x, y) \in R \qquad \text{právě tehdy, když} \qquad y = x + 1$$

Platí tedy

$$R = \{(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), \ldots\}$$

Funkce

Podobně ternární relace T je (binární) funkce, jestliže pro každé dva prvky x_1 a x_2 existuje nejvýše jedno (resp. v případě totální funkce právě jedno) y takové, že

$$(x_1, x_2, y) \in T.$$

Příklad: Sčítání na množině reálných čísel $\mathbb{R}$ je možné chápat jako ternární relaci S (tj. jako množinu trojic reálných čísel), kde

$$(x_1, x_2, y) \in S \qquad \text{právě tehdy, když} \qquad x_1 + x_2 = y$$

Funkce

V predikátové logice můžeme funkce reprezentovat pomocí predikátů zastupujících příslušné relace — není to ale příliš přímočaré ani pohodlné.

Příklad: „*Pro každé* x *a* y *platí, že* $x + y \geq y + x$.“

$$\forall x \forall y \exists z \exists w (S(x, y, z) \wedge S(y, x, w) \wedge P(z, w))$$

- $S(x, y, z)$ — „z *je součtem hodnot* x *a* y“
- $P(x, y)$ — „x *je větší nebo rovno* y“

Poznámka: Navíc musíme předpokládat, že pro každé dva prvky x a y existuje právě jeden prvek z takový, že $S(x, y, z)$.

Funkce

V predikátové logice je možné reprezentovat funkce pomocí **funkčních symbolů**.

Abeceda:

- ...
- funkční symboly: "f", "g", "h", ...
- ...

Každý funkční symbol musí mít stanovenu **aritu** odpovídající aritě funkce, kterou tento symbol zastupuje (tj. počet argumentů příslušné funkce).

Termy

Termy — výrazy složené z proměnných a konstantních a funkčních symbolů reprezentující prvky universa

Příklad:

- Řekněme, že máme binární predikát F, kde předpokládáme, že pro každé x existuje právě jedno y takové, že

  $$F(x, y).$$

  Místo binárního predikátu F můžeme použít unární funkční symbol f.

  Term

  $$f(x)$$

  reprezentuje onen jediný prvek y, pro který platí $F(x, y)$.

  Místo $\exists y(F(x, y) \wedge P(y))$ pak můžeme psát $P(f(x))$.

Termy

Příklad:

- Řekněme, že máme ternární predikát G, kde předpokládáme, že pro každé dva prvky x_1 a x_2 existuje právě jedno y takové, že

$$G(x_1, x_2, y).$$

Místo ternárního predikátu G můžeme použít binární funkční symbol g.

Term

$$g(x_1, x_2)$$

reprezentuje onen jediný prvek y, pro který platí $G(x_1, x_2, y)$.

Termy

Příklad: „*Pro každé* x *a* y *platí, že* $x + y \geq y + x$.“

$$\forall x \forall y P(f(x, y), f(y, x))$$

- f — binární funkční symbol, kde $f(x, y)$ reprezentuje součet hodnot x a y
- P — binární predikátový symbol, kde $P(x, y)$ reprezentuje vztah „x *je větší nebo rovno* y“

Termy

Proměnné, konstantní symboly a funkční symboly je možné v termech libovolně skládat — je pouze třeba dodržet aritu všech funkčních symbolů (aplikovat každý funkční symbol na správný počet argumentů).

Příklad:

- c — konstantní symbol
- f — unární funkční symbol
- g — binární funkční symbol
- h — binární funkční symbol

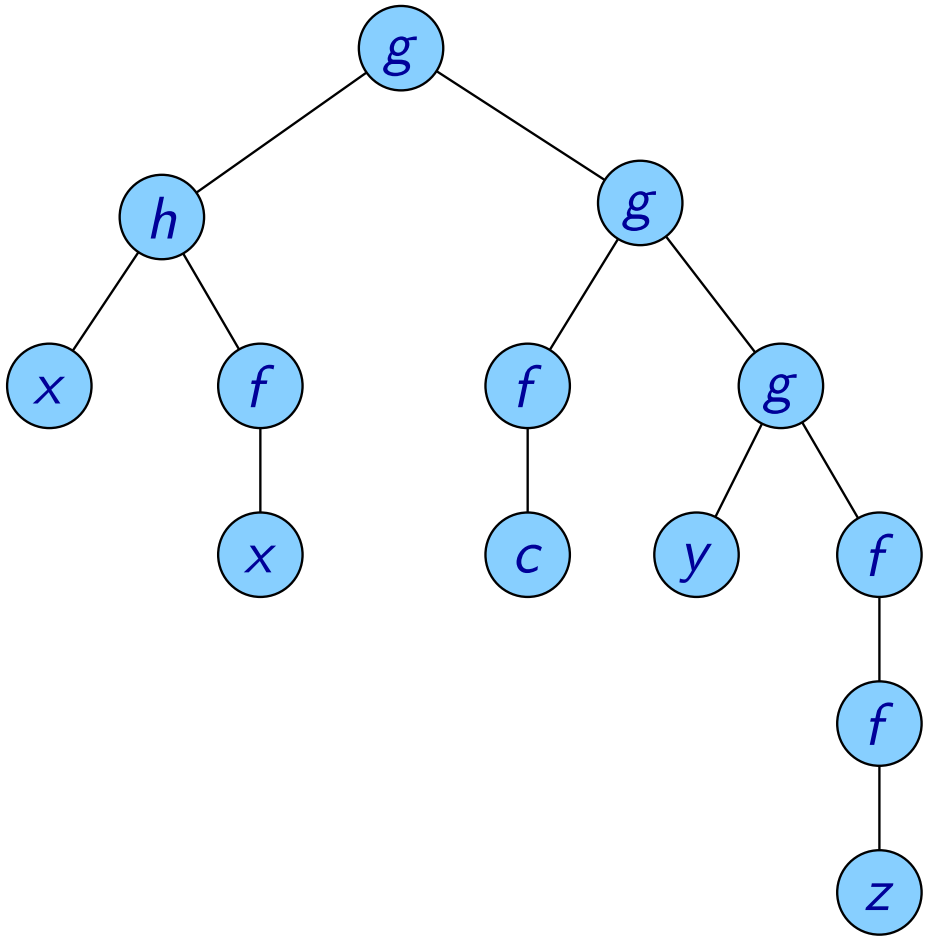
Příklady termů:

$$x \qquad f(y) \qquad g(c,x) \qquad g(h(x,x),f(c))$$

$$g(h(x,f(x)),g(f(c),g(y,f(f(z)))))$$

Termy

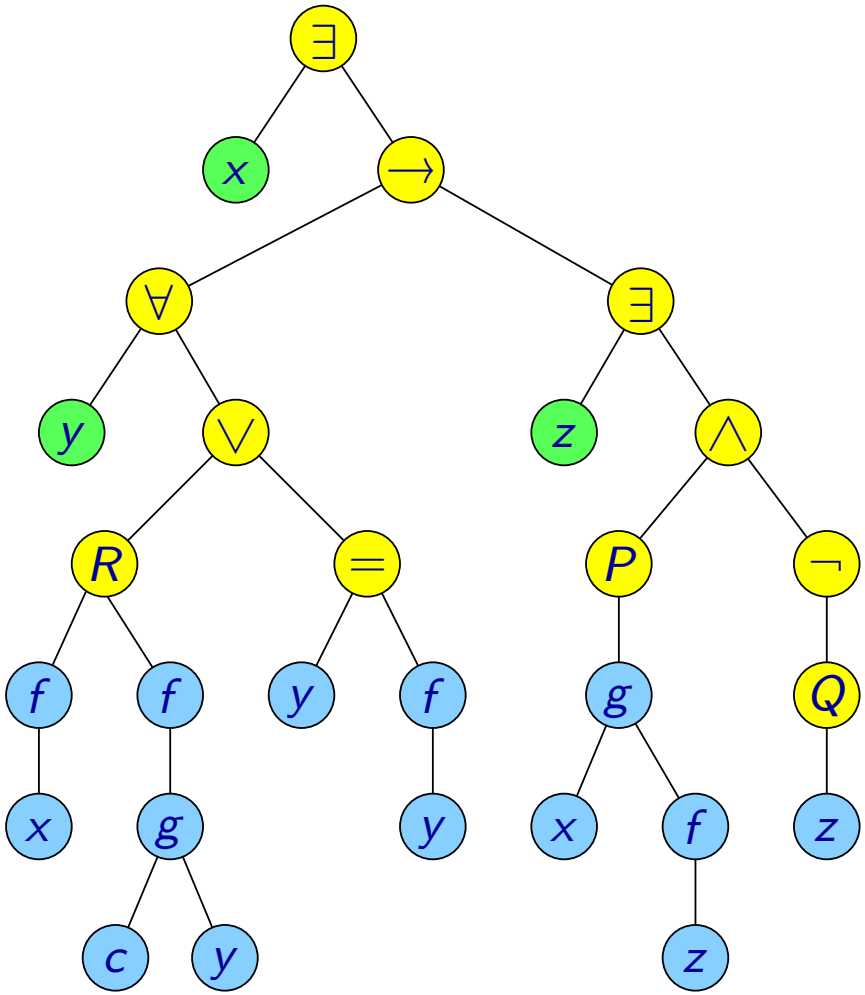
Syntaktický strom termu $g(h(x, f(x)), g(f(c), g(y, f(f(z)))))$



Termy ve formulích

Syntaktický strom formule

$\exists x(\forall y(R(f(x), f(g(c, y))) \vee y = f(y)) \rightarrow \exists z(P(g(x, f(z))) \wedge \neg Q(z)))$



Termy ve formulích

Příklad:

- Pro každé x, y a z platí $(x + y) + z = x + (y + z)$:
$$\forall x \forall y \forall z (f(f(x, y), z) = f(x, f(y, z)))$$
- Pro každé x platí $x + 0 = x$ a $0 + x = x$:
$$\forall x (f(x, e) = x \wedge f(e, x) = x)$$
- Pro každé x existuje y takové, že $x + y = 0$:
$$\forall x \exists y (f(x, y) = e)$$

Konstantní a funkční symboly:

- f — binární funkční symbol reprezentující „*sčítání*“ (operace “$+$”)
- e — konstatní symbol reprezentující prvek “0”

Termy ve formulích

Příklad:

- Pro každé x, y a z platí $x \cdot (y + z) = x \cdot y + x \cdot z$:

$$\forall x \forall y \forall z (g(x, f(y, z)) = f(g(x, y), g(x, z)))$$

- Pro každé x a y takové, že $x \leq y$, platí pro všechna z, že $x + z \leq y + z$:

$$\forall x \forall y (R(x, y) \rightarrow \forall z R(f(x, y), f(y, z)))$$

Konstantní a funkční symboly:

- f — binární funkční symbol reprezentující „*sčítání*“ (operace “$+$”)
- g — binární funkční symbol reprezentující „*násobení*“ (operace “$\cdot$”)
- R — binární predikátový symbol reprezentující relaci „*menší nebo rovno*“ (relace “$\leq$”)

Syntaxe formulí predikátové logiky

Abeceda:

- **logické spojky** — "$\neg$", "$\wedge$", "$\vee$", "$\rightarrow$" a "$\leftrightarrow$"
- **kvantifikátory** — "$\forall$" a "$\exists$"
- **rovnost** — "$=$"
- **pomocné symboly** — "$($", "$)$" a "$,$"
- **proměnné** — "x", "y", "z", ..., "x_0", "x_1", "x_2", ...

- **predikátové symboly** — například symboly "P", "Q", "R", apod.
  (u každého symbolu musí být navíc stanovena příslušná arita)
- **funkční symboly** — například symboly "f", "g", "h", apod.
  (u každého symbolu musí být navíc stanovena příslušná arita)
- **konstantní symboly** — například symboly "a", "b", "c", apod.

Poznámka: Na konstantní symboly se lze dívat jako na funkční symboly s aritou 0.

Syntaxe formulí predikátové logiky

Definice

Dobře utvořené **termy** jsou definovány následujícím způsobem:

1. Jestliže x je proměnná, tak x je dobře utvořený term.
2. Jestliže c je konstatní symbol, tak c je dobře utvořený term.
3. Jestliže f je funkční symbol s aritou n a t_1, t_2, ..., t_n jsou dobře utvořené termy, tak

   $$f(t_1, t_2, \ldots, t_n)$$

   je dobře utvořený term.
4. Neexistují žádné další dobře utvořené termy než ty, které jsou vytvořeny pomocí předchozích pravidel.

Syntaxe formulí predikátové logiky

Definice

Dobře utvořené **atomické formule** jsou definovány následujícím způsobem:

1. Jestliže P je predikátový symbol s aritou n a t_1, t_2, ..., t_n jsou dobře utvořené termy, tak

   $$P(t_1, t_2, \ldots, t_n)$$

   je dobře utvořená atomická formule.

2. Jestliže t_1 a t_2 jsou dobře utvořené termy, tak

   $$t_1 = t_2$$

   je dobře utvořená atomická formule.

3. Neexistují žádné další dobře atomické formule než ty, které jsou vytvořeny pomocí předchozích dvou pravidel.

Syntaxe formulí predikátové logiky

Definice (zopakování dříve uvedené definice)

Dobře utvořené **formule predikátové logiky** jsou posloupnosti symbolů vytvořené podle následujících pravidel:

1. Dobře utvořené atomické formule jsou dobře utvořené formule.
2. Jestliže φ a ψ jsou dobře utvořené formule, pak i $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$ a $(\varphi \leftrightarrow \psi)$ jsou dobře utvořené formule.
3. Jestliže φ je dobře utvořená formule a x je proměnná, tak $\forall x \varphi$ a $\exists x \varphi$ jsou dobře utvořené formule.
4. Neexistují žádné další dobře utvořené formule než ty, které jsou vytvořené pomocí předchozích pravidel.

Sémantika predikátové logiky

Interpretace $\mathcal{A}$:

- universum A
- každému predikátovému symbolu P s aritou n je přiřazena n-ární relace $P^{\mathcal{A}}$, kde $P^{\mathcal{A}} \subseteq A \times A \times \cdots \times A$
- každému funkčnímu symbolu f s aritou n je přiřazena n-ární funkce $f^{\mathcal{A}}$, kde $f^{\mathcal{A}} : A \times A \times \cdots \times A \to A$
- každému konstantnímu symbolu c je přiřazen prvek universa $c^{\mathcal{A}}$, tj. $c^{\mathcal{A}} \in A$

Poznámka: V interpretacích se funkčním symbolům přiřazují pouze **totální** funkce, tj. funkce, jejichž hodnota je definovaná pro všechny možné hodnoty argumentů.

Sémantika predikátové logiky

Hodnota termu v interpretaci $\mathcal{A}$ a valuaci v:

- Term x, kde x je proměnná — hodnotou tohoto termu je prvek $a \in A$ takový, že $v(x) = a$.
- Term c, kde c je konstantní symbol — hodnotou tohoto termu je prvek $c^{\mathcal{A}} \in A$.
- Term $f(t_1, t_2, \ldots, t_n)$, kde $t_1, t_2, \ldots, t_n$ jsou termy — hodnotou tohoto termu je prvek $b \in A$ takový, že

$$b = f^{\mathcal{A}}(a_1, a_2, \ldots, a_n),$$

kde $a_1, a_2, \ldots, a_n$ jsou hodnoty termů $t_1, t_2, \ldots, t_n$ v interpretaci $\mathcal{A}$ a valuaci v.

Sémantika predikátové logiky

Příklad: Intrepretace $\mathcal{A}$, kde universem je množina přirozených čísel $\mathbb{N} = \{0, 1, 2, \ldots\}$.

- $a^{\mathcal{A}} = 0$
- $f^{\mathcal{A}}$ je funkce „*následník*“, tj. $f^{\mathcal{A}}(x) = x + 1$
- $g^{\mathcal{A}}$ je funkce „*součet*“, tj. $g^{\mathcal{A}}(x, y) = x + y$

Valuace v, kde $v(x) = 5$, $v(y) = 13$, $v(z) = 2$, ...

Hodnoty termů v interpretaci $\mathcal{A}$ a valuaci v:

- Term x — hodnota 5
- Term a — hodnota 0
- Term $f(a)$ — hodnota 1 $(0 + 1 = 1)$
- Term $f(f(a))$ — hodnota 2 $(1 + 1 = 2)$
- Term $g(x, f(f(a)))$ — hodnota 7 $(5 + 2 = 7)$
- Term $g(z, y)$ — hodnota 15 $(2 + 13 = 15)$
- Term $f(g(z, y))$ — hodnota 16 $(15 + 1 = 16)$

Sémantika predikátové logiky

Pravdivost atomických formulí v interpretaci $\mathcal{A}$ a valuaci v:

- $\mathcal{A}, v \models P(t_1, t_2, \ldots, t_n)$, kde P je predikátový symbol s aritou n a $t_1, t_2, \ldots, t_n$ jsou termy, platí právě tehdy, když

  $$(a_1, a_2, \ldots, a_n) \in P^{\mathcal{A}},$$

  kde $a_1, a_2, \ldots, a_n$ jsou hodnoty termů $t_1, t_2, \ldots, t_n$ v interpretaci $\mathcal{A}$ a valuaci v.

- $\mathcal{A}, v \models t_1 = t_2$, kde t_1 a t_2 jsou termy, platí právě tehdy, když

  $$a_1 = a_2,$$

  kde a_1 a a_2 jsou hodnoty termů t_1 a t_2 v interpretaci $\mathcal{A}$ a valuaci v.

Sémantika predikátové logiky

Příklad: Intrepretace $\mathcal{A}$, kde universem je množina přirozených čísel $\mathbb{N} = \{0, 1, 2, \ldots\}$.

- $f^{\mathcal{A}}$ je funkce „*následník*“, tj. $f^{\mathcal{A}}(x) = x + 1$
- $g^{\mathcal{A}}$ je funkce „*součet*“, tj. $g^{\mathcal{A}}(x, y) = x + y$
- $P^{\mathcal{A}}$ je množina všech prvočísel
- $Q^{\mathcal{A}}$ je binární relace “$<$” tj. $(x, y) \in Q^{\mathcal{A}}$ právě tehdy, když $x < y$

Valuace v, kde $v(x) = 5$, $v(y) = 13$, $v(z) = 2$, ...

- $\mathcal{A}, v \models P(x)$ (5 je prvočíslo)
- $\mathcal{A}, v \not\models Q(y, z)$ (není pravda, že $13 < 2$)
- $\mathcal{A}, v \models Q(f(f(z)), g(x, y))$ ($(2 + 1) + 1 < 5 + 13$)
- $\mathcal{A}, v \not\models P(f(g(z, x)))$ ($(2 + 5) + 1 = 8$ a 8 není prvočíslo)

Predikátová logika — další poznámky

- Jedná se o predikátovou logiku **1. řádu** — kvantifikovat lze pouze přes prvky universa (v predikátové logice 2. řádu je možné kvantifikovat přes relace) — it is possible to quantify only over the elements of the universe (in the second order predicate logic, it is possible to quantify over relations).

Predikátová logika — další poznámky

V běžné matematické praxi se většinou nepoužívá syntaxe formulí predikátové logiky přesně podle definice, ale při zápisu se používá celá řada konvencí a zkratek.

- Pro **binární** funkční a predikátové symboly se často používá **infixový** způsob zápisu:

  Například místo $f(x, y)$ a $R(x, y)$ se může psát

  $$x\ f\ y \qquad\qquad x\ R\ y$$

- Pro označení predikátových, funkčních a konstantních symbolů a proměnných se používají všechny možné druhy symbolů:

  Například místo $R(f(x, y), g(z))$ se může psát třeba

  $$x + \delta \leq |\varepsilon| \qquad \text{nebo například} \qquad x \circ y \sqsupset G(z)$$

Predikátová logika — další poznámky

Příklady formulí reprezentujících tvrzení z teorie množin:

- x je prvkem množiny A:

$$x \in A$$

"$\in$" — binární predikátový symbol reprezentující relaci *„náležení“*
"x", "A" — proměnné

Pokud bychom provedli následující změny:

- místo symbolu "$\in$" bychom použili binární predikátový symbol E,
- místo proměnné A bychom použili proměnnou y,

tak by formule vypadala následovně:

$$E(x, y)$$

Predikátová logika — další poznámky

- Dvě množiny se rovnají, pokud obsahují stejné prvky:

$$A = B \leftrightarrow \forall x(x \in A \leftrightarrow x \in B)$$

Pokud bychom místo "$\in$" použili predikát E, a místo A a B použili y a z, formule bude vypadat takto:

$$y = z \leftrightarrow \forall x(E(x, y) \leftrightarrow E(x, z))$$

- Definice relace *„být podmnožinou"* (označena symbolem "$\subseteq$"):

$$A \subseteq B \leftrightarrow \forall x(x \in A \rightarrow x \in B)$$

Pokud místo "$\subseteq$" použijeme binární predikátový symbol S:

$$S(y, z) \leftrightarrow \forall x(E(x, y) \rightarrow E(x, z))$$

Predikátová logika — další poznámky

- Definice operace „*sjednocení*“ (označena symbolem “$\cup$”):

$$\forall x(x \in A \cup B \leftrightarrow (x \in A \vee x \in B))$$

Pokud místo “$\cup$” použijeme binární funkční symbol f:

$$\forall x(E(x, f(y, z)) \leftrightarrow (E(x, y) \vee E(x, z)))$$

Predikátová logika — další poznámky

- Místo

$$\exists x(x \in A \wedge \ldots)$$

se někdy používá zkrácený zápis

$$(\exists x \in A)(\ldots)$$

Tj. místo

„*existuje* x *takové, že* $x \in A$ *a* $\ldots$“

se řekne

„*existuje* $x \in A$ *takové, že* $\ldots$“

- Podobně třeba místo $\exists x(x \geq 1 \wedge \ldots)$ se někdy zkráceně píše

$$(\exists x \geq 1)(\ldots)$$

Predikátová logika — další poznámky

- Místo

$$\forall x(x \in A \rightarrow \ldots)$$

se někdy používá zkrácený zápis

$$(\forall x \in A)(\ldots)$$

Tj. místo

„*pro každé* x, *pro které platí* $x \in A$, *platí* $\ldots$“

se řekne

„*pro každé* $x \in A$ *platí* $\ldots$“

- Podobně třeba místo $\forall x(x \geq 1 \rightarrow \ldots)$ se někdy zkráceně píše

$$(\forall x \geq 1)(\ldots)$$

Formální jazyky

Abeceda a slovo

Definice

Abeceda je libovolná neprázdná konečná množina **symbolů** (**znaků**).

Poznámka: Abeceda se často označuje řeckým písmenem Σ (velké sigma).

Definice

Slovo v dané abecedě je libovolná konečná posloupnost symbolů z této abecedy.

Příklad 1:

$\Sigma = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$

Slova v abecedě Σ: $\mathtt{AHOJ}$ $\mathtt{ABRACADABRA}$ $\mathtt{ERROR}$

Abeceda a slovo

Příklad 2:

$\Sigma_2 = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{J}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{O}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{U}, \mathtt{V}, \mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \sqcup\}$

Slovo v abecedě Σ_2: $\mathtt{HELLO}\sqcup\mathtt{WORLD}$

Příklad 3:

$\Sigma_3 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Slova v abecedě Σ_3: 0, 31415926536, 65536

Příklad 4:

Slova v abecedě $\Sigma_4 = \{0, 1\}$: 011010001, 111, 1010101010101010

Příklad 5:

Slova v abecedě $\Sigma_5 = \{a, b\}$: *aababb*, *abbabbba*, *aaab*

Abeceda a slovo

Příklad 6:

Abeceda Σ_6 je množina všech ASCII znaků.

Příklad slova:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

```
class␣HelloWorld␣{ ↩ ␣␣␣␣public␣static␣void␣main(Str···
```

Teorie formálních jazyků – motivace

Jazyk — množina (některých) slov tvořených symboly z dané abecedy

Příklady typů problémů, při jejichž řešení se využívá poznatků z teorie formálních jazyků:

- Tvorba překladačů:
  - lexikální analýza
  - syntaktická analýza
- Vyhledávání v textu:
  - hledání zadaného vzorku
  - hledání textu zadaného regulárním výrazem

Práce s formálními jazyky

Když chceme nějaký jazyk popsat, máme několik možností:

- Můžeme vyjmenovat všechna jeho slova (což je ale použitelné jen pro malé konečné jazyky).

  Příklad: $L = \{aab, babba, aaaaaa\}$

- Můžeme specifikovat nějakou vlastnost, kterou mají právě ta slova, která do tohoto jazyka patří:

  Příklad: Jazyk nad abecedou $\{0, 1\}$, obsahující všechna slova, ve kterých je počet výskytů symbolu 1 sudý.

Práce s formálními jazyky

V teorii formálních jazyků se používají především následující dva přístupy:

- Popsat (idealizovaný) stroj, zařízení, algoritmus, který rozpozná slova patřící do daného jazyka – vede k použití tzv. **automatů**.
- Popsat nějaký mechanismus umožňující generovat všechna možná slova patřící do daného jazyka – vede k tzv. **gramatikám** a **regulárním výrazům**.

Některé základní pojmy

Délka slova je počet znaků ve slově.

Například délka slova $abaab$ je 5.

Délku slova w označujeme $|w|$.

Pokud tedy např. $w = abaab$, pak $|w| = 5$.

Počet výskytů znaku a ve slově w označujeme $|w|_a$.

Pro slovo $w = ababb$ tedy platí $|w|_a = 2$ a $|w|_b = 3$.

Prázdné slovo je slovo délky 0, tj. neobsahující žádné znaky.

Prázdné slovo se označuje řeckým písmenem ε (epsilon).

(Pozn.: V literatuře se pro označení prázdného slova někdy používá místo symbolu ε řecké písmeno λ (lambda).)

$$|\varepsilon| = 0$$

Zřetězení slov

Se slovy je možné provádět operaci **zřetězení**:

Například zřetězením slov OST a RAVA vznikne slovo OSTRAVA.

Operace zřetězení se označuje symbolem $\cdot$ (podobně jako násobení). Tento symbol je možné vypouštět.

$$\mathtt{OST} \cdot \mathtt{RAVA} = \mathtt{OSTRAVA}$$

Zřetězení je **asociativní**, tj. pro libovolná tři slova u, v a w platí

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

což znamená, že při zápisu více zřetězení můžeme vypouštět závorky a psát například $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$ místo $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$.

Zřetězení slov

Zřetězení není **komutativní**, tj. obecně pro dvojici slov u a v neplatí rovnost

$$u \cdot v = v \cdot u$$

Příklad:

$$\texttt{OST} \cdot \texttt{RAVA} \neq \texttt{RAVA} \cdot \texttt{OST}$$

Zjevně pro libovolná slova v a w platí:

$$|v \cdot w| = |v| + |w|$$

Pro libovolné slovo w také platí:

$$\varepsilon \cdot w = w \cdot \varepsilon = w$$

Prefixy, sufixy a podslova

Definice

Slovo x je **prefixem** slova y, jestliže existuje slovo v takové, že $y = xv$.

Slovo x je **sufixem** slova y, jestliže existuje slovo u takové, že $y = ux$.

Slovo x je **podslovem** slova y, jestliže existují slova u a v taková, že $y = uxv$.

Příklad:

- Prefixy slova abaab jsou ε, a, ab, aba, abaa, abaab.
- Sufixy slova abaab jsou ε, b, ab, aab, baab, abaab.
- Podslova slova abaab jsou ε, a, b, ab, ba, aa, aba, baa, aab, abaa, baab, abaab.

Jazyk

Množinu všech slov tvořených symboly z abecedy Σ označujeme Σ^*.

Definice

(Formální) jazyk L v abecedě Σ je nějaká libovolná podmnožina množiny Σ^*, tj. $L \subseteq \Sigma^*$.

Příklad 1: Množina $\{00, 01001, 1101\}$ je jazyk v abecedě $\{0, 1\}$.

Příklad 2: Množina všech syntakticky správných programů v jazyce C je jazyk v abecedě tvořené množinou všech ASCII znaků.

Příklad 3: Množina všech textů obsahujících sekvenci znaků `ahoj` je jazyk v abecedě tvořené množinou všech ASCII znaků.

Množinové operace na jazycích

Vzhledem k tomu, že jazyky jsou množiny, můžeme s nimi provádět množinové operace:

Sjednocení – $L_1 \cup L_2$ je jazyk tvořený slovy, která patří buď do jazyka L_1 nebo do jazyka L_2 (nebo do obou).

Průnik – $L_1 \cap L_2$ je jazyk tvořený slovy, která patří současně do jazyka L_1 i do jazyka L_2.

Doplněk – $\overline{L_1}$ je jazyk tvořený těmi slovy ze Σ^*, která nepatří do L_1.

Rozdíl – $L_1 - L_2$ je jazyk tvořený slovy, která patří do L_1, ale nepatří do L_2.

Poznámka: Při operacích nad jazyky předpokládáme, že jazyky, se kterými operaci provádíme, používají tutéž abecedu Σ.

Množinové operace na jazycích

Formálně:

Sjednocení: $L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$

Průnik: $L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$

Doplněk: $\overline{L_1} = \{w \in \Sigma^* \mid w \notin L_1\}$

Rozdíl: $L_1 - L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \notin L_2\}$

Poznámka: Předpokládáme, že $L_1, L_2 \subseteq \Sigma^*$ pro nějakou danou abecedu Σ.

Množinové operace na jazycích

Příklad:

Uvažujme jazyky nad abecedou $\{\mathtt{a}, \mathtt{b}\}$.

- L_1 — množina všech slov obsahujících podslovo $\mathtt{baa}$
- L_2 — množina všech slov se sudým počtem výskytů symbolu $\mathtt{b}$

Pak

- $L_1 \cup L_2$ — množina všech slov obsahujících podslovo $\mathtt{baa}$ nebo sudý počet symbolů $\mathtt{b}$
- $L_1 \cap L_2$ — množina všech slov obsahujících podslovo $\mathtt{baa}$ a sudý počet symbolů $\mathtt{b}$
- $\overline{L_1}$ — množina všech slov, která neobsahují podslovo $\mathtt{baa}$
- $L_1 - L_2$ — množina všech slov, ve kterých se vyskytuje podslovo $\mathtt{baa}$, ale kde počet symbolů $\mathtt{b}$ není sudý

Zřetězení jazyků

Definice

Zřetězení jazyků L_1 a L_2, kde $L_1, L_2 \subseteq \Sigma^*$, je jazyk $L \subseteq \Sigma^*$ takový, že pro každé $w \in \Sigma^*$ platí

$$w \in L \leftrightarrow (\exists u \in L_1)(\exists v \in L_2)(w = u \cdot v)$$

Zřetězení jazyků L_1 a L_2 označujeme $L_1 \cdot L_2$.

Příklad:

$$\begin{aligned} L_1 &= \{abb, ba\} \\ L_2 &= \{a, ab, bbb\} \end{aligned}$$

Jazyk $L_1 \cdot L_2$ obsahuje slova:

$$abba \qquad abbab \qquad abbbbb \qquad baa \qquad baab \qquad babbb$$

Iterace jazyka

Definice

Iterace jazyka L, označovaná zápisem L^*, je jazyk tvořený slovy vzniklými zřetězením libovolného počtu slov z jazyka L.
Tj. $w \in L^*$ právě tehdy, když

$$\exists n \in \mathbb{N} : \exists w_1, w_2, \ldots, w_n \in L : w = w_1 w_2 \cdots w_n$$

Příklad: $L = \{aa, b\}$

$$L^* = \{\varepsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \ldots\}$$

Poznámka: Počet slov, která zřetězujeme, může být i 0, což znamená, že vždy platí $\varepsilon \in L^*$ (bez ohledu na to, zda $\varepsilon \in L$ nebo ne).

Iterace jazyka — alternativní definice

Nejprve definujeme pro jazyk L a číslo $k \in \mathbb{N}$ jazyk L^k:

$$L^0 = \{\varepsilon\}, \qquad L^k = L^{k-1} \cdot L \quad \text{pro } k \geq 1$$

To znamená

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= L \\ L^2 &= L \cdot L \\ L^3 &= L \cdot L \cdot L \\ L^4 &= L \cdot L \cdot L \cdot L \\ L^5 &= L \cdot L \cdot L \cdot L \cdot L \\ &\dots \end{aligned}$$

Příklad: Pro $L = \{aa, b\}$ jazyk L^3 obsahuje následující slova:

$$aaaaaa \quad aaaab \quad aabaa \quad aabb \quad baaaa \quad baab \quad bbaa \quad bbb$$

Iterace jazyka — alternativní definice

Alternativní definice

Iterace jazyka L je jazyk

$$L^* = \bigcup_{k \geq 0} L^k$$

Poznámka:

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

Iterace jazyka

Poznámka: Používá se také zápis L^+ jako zkratka za $L \cdot L^*$, tj.

$$L^+ = \bigcup_{k \geq 1} L^k$$

Zrcadlový obraz

Zrcadlový obraz slova w je slovo w zapsané „pozpátku“.

Zrcadlový obraz slova w značíme w^R.

Příklad: $w = \texttt{AHOJ}$ $w^R = \texttt{JOHA}$

Formálně můžeme definovat, že pro $w = a_1 a_2 \cdots a_n$ (kde $a_i \in \Sigma$) je $w^R = a_n a_{n-1} \cdots a_1$.

Zrcadlový obraz

Zrcadlový obraz jazyka L je jazyk tvořený zrcadlovými obrazy všech slov z jazyka L.

Zrcadlový obraz jazyka L značíme L^R.

$$L^R = \{w^R \mid w \in L\}$$

Příklad: $L = \{ab, baaba, aaab\}$

$L^R = \{ba, abaab, baaa\}$

Uspořádání na slovech

Předpokládejme určité (lineární) uspořádání $<$ symbolů abecedy Σ, tj. pokud $\Sigma = \{a_1, a_2, \ldots, a_n\}$, tak platí

$$a_1 < a_2 < \ldots < a_n \,.$$

Příklad: $\Sigma = \{a, b, c\}$, přičemž $a < b < c$.

Na množině Σ^* můžeme definovat následující (lineární) uspořádání $<_L$: $x <_L y$ právě tehdy, když:

- $|x| < |y|$, nebo
- $|x| = |y|$ a existují slova $u, v, w \in \Sigma^*$ a symboly $a, b \in \Sigma$ takové, že platí

$$x = uav \qquad y = ubw \qquad a < b$$

Neformálně můžeme říct v uspořádání $<_L$ řadíme slova podle délky a v rámci stejné délky lexikograficky (podle abecedy).

Uspořádání na slovech

Všechna slova nad abecedou Σ můžeme pomocí uspořádání $<_L$ seřadit do posloupnosti

$$w_0, w_1, w_2, \ldots$$

ve které se každé slovo $w \in \Sigma^*$ vyskytuje právě jednou a kde pro libovolná $i, j \in \mathbb{N}$ platí, že $w_i <_L w_j$ právě tehdy, když $i < j$.

Příklad: Pro abecedu $\Sigma = \{a, b, c\}$ (kde $a < b < c$) bude začátek posloupnosti vypadat následovně:

$$\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, \ldots$$

Pokud budeme mluvit například o prvních deseti slovech jazyka $L \subseteq \Sigma^*$, máme tím na mysli deset slov, která patří do jazyka L a jsou mezi všemi slovy z jazyka L nejmenší vzhledem k uspořádání $<_L$.

Regulární výrazy

Regulární výrazy

Regulární výrazy popisující jazyky nad abecedou Σ:

- $\emptyset$, ε, a (kde $a \in \Sigma$) jsou regulární výrazy:

  $\emptyset$ ... označuje prázdný jazyk
  ε ... označuje jazyk $\{\varepsilon\}$
  a ... označuje jazyk $\{a\}$

- Jestliže α, β jsou regulární výrazy, pak i $(\alpha + \beta)$, $(\alpha \cdot \beta)$, (α^*) jsou regulární výrazy:

  $(\alpha + \beta)$ ... označuje sjednocení jazyků označených α a β
  $(\alpha \cdot \beta)$ ... označuje zřetězení jazyků označených α a β
  (α^*) ... označuje iteraci jazyka označeného α

- Neexistují žádné další regulární výrazy než ty definované podle předchozích dvou bodů.

Regulární výrazy

Příklad: abeceda $\Sigma = \{0, 1\}$

- Podle definice jsou 0 i 1 regulární výrazy.

Regulární výrazy

Příklad: abeceda $\Sigma = \{0, 1\}$

- Podle definice jsou 0 i 1 regulární výrazy.
- Protože 0 i 1 jsou regulární výrazy, je i $(0 + 1)$ regulární výraz.

Regulární výrazy

Příklad: abeceda $\Sigma = \{0, 1\}$

- Podle definice jsou 0 i 1 regulární výrazy.
- Protože 0 i 1 jsou regulární výrazy, je i $(0 + 1)$ regulární výraz.
- Protože 0 je regulární výraz, je i (0^*) regulární výraz.

Regulární výrazy

Příklad: abeceda $\Sigma = \{0, 1\}$

- Podle definice jsou 0 i 1 regulární výrazy.
- Protože 0 i 1 jsou regulární výrazy, je i $(0 + 1)$ regulární výraz.
- Protože 0 je regulární výraz, je i (0^*) regulární výraz.
- Protože $(0 + 1)$ i (0^*) jsou regulární výrazy, je i $((0 + 1) \cdot (0^*))$ regulární výraz.

Regulární výrazy

Příklad: abeceda $\Sigma = \{0, 1\}$

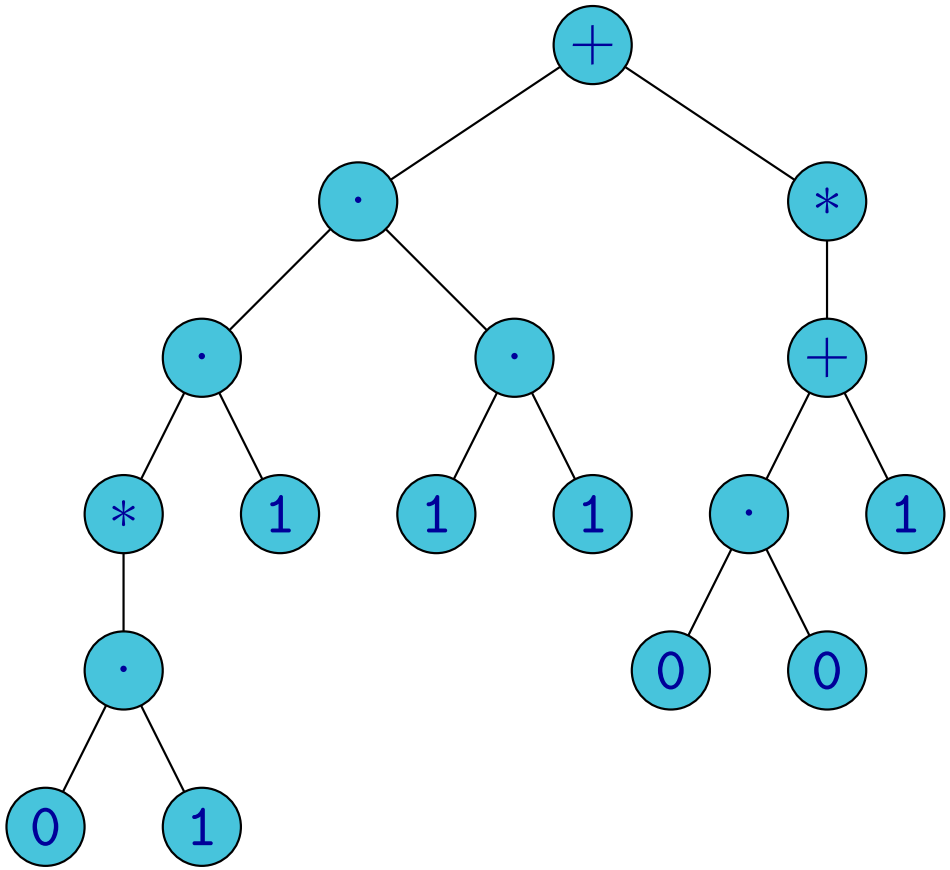
- Podle definice jsou 0 i 1 regulární výrazy.
- Protože 0 i 1 jsou regulární výrazy, je i $(0 + 1)$ regulární výraz.
- Protože 0 je regulární výraz, je i (0^*) regulární výraz.
- Protože $(0 + 1)$ i (0^*) jsou regulární výrazy, je i $((0 + 1) \cdot (0^*))$ regulární výraz.

Poznámka: Jestliže α je regulární výraz, zápisem $[\alpha]$ označujeme jazyk definovaný regulárním výrazem α.

$$[((0 + 1) \cdot (0^*))] = \{0,\ 1,\ 00,\ 10,\ 000,\ 100,\ 0000,\ 1000,\ 00000,\ \ldots\}$$

Regulární výrazy

Strukturu regulárního výrazu si můžeme znázornit abstraktním syntaktickým stromem:



$$((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$$

Regulární výrazy

Formální definice sémantiky regulárních výrazů:

- $[\emptyset] = \emptyset$
- $[\varepsilon] = \{\varepsilon\}$
- $[a] = \{a\}$
- $[\alpha^*] = [\alpha]^*$
- $[\alpha \cdot \beta] = [\alpha] \cdot [\beta]$
- $[\alpha + \beta] = [\alpha] \cup [\beta]$

Regulární výrazy

Aby byl zápis regulárních výrazů přehlednější a stručnější, používáme následují pravidla:

- Vynecháváme vnější pár závorek.
- Vynecháváme závorky, které jsou zbytečné vzhledem k asociativitě operací sjednocení ($+$) a zřetězení ($\cdot$).
- Vynecháváme závorky, které jsou zbytečné vzhledem k prioritě operací (nejvyšší prioritu má iterace (*), menší zřetězení ($\cdot$) a nejmenší sjednocení ($+$)).
- Nepíšeme tečku pro zřetězení.

Příklad: Místo

$$(((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$$

obvykle píšeme

$$(01)^*111 + (00 + 1)^*$$

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 . . . jazyk tvořený jediným slovem 0

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 . . . jazyk tvořený jediným slovem 0

01 . . . jazyk tvořený jediným slovem 01

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

0^* ... jazyk tvořený slovy ε, 0, 00, 000, ...

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

0^* ... jazyk tvořený slovy ε, 0, 00, 000, ...

$(01)^*$ ... jazyk tvořený slovy ε, 01, 0101, 010101, ...

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

0^* ... jazyk tvořený slovy ε, 0, 00, 000, ...

$(01)^*$ ... jazyk tvořený slovy ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

0^* ... jazyk tvořený slovy ε, 0, 00, 000, ...

$(01)^*$ ... jazyk tvořený slovy ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$

$(0 + 1)^*00$ ... jazyk tvořený všemi slovy končícími 00

Regulární výrazy

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

$0 + 1$ ... jazyk tvořený dvěma slovy 0 a 1

0^* ... jazyk tvořený slovy ε, 0, 00, 000, ...

$(01)^*$ ... jazyk tvořený slovy ε, 01, 0101, 010101, ...

$(0 + 1)^*$ ... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$

$(0 + 1)^*00$ ... jazyk tvořený všemi slovy končícími 00

$(01)^*111(01)^*$ ... jazyk tvořený všemi slovy obsahujícími podslovo 111 předcházené i následované libovolným počtem slov 01

Regulární výrazy

$(0+1)^*00 + (01)^*111(01)^*$ ... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podslovo 111 předcházené i následované libovolným počtem slov 01

Regulární výrazy

$(0+1)^*00+(01)^*111(01)^*$ ... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podslovo 111 předcházené i následované libovolným počtem slov 01

$(0+1)^*1(0+1)^*$ ... jazyk tvořený všemi slovy obsahujícími alespoň jeden symbol 1

Regulární výrazy

$(0+1)^*00+(01)^*111(01)^*$ ... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podslovo 111 předcházené i následované libovolným počtem slov 01

$(0+1)^*1(0+1)^*$ ... jazyk tvořený všemi slovy obsahujícími alespoň jeden symbol 1

$0^*(10^*10^*)^*$ ... jazyk tvořený všemi slovy obsahujícími sudý počet symbolů 1

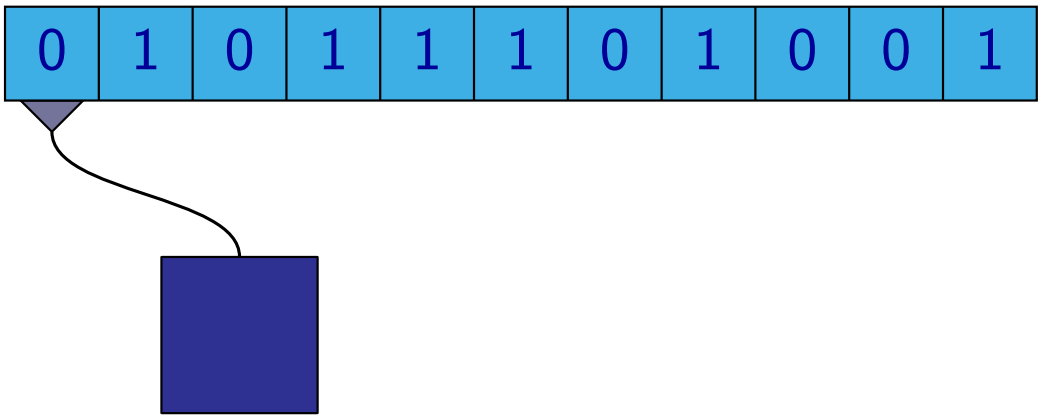
Konečné automaty

Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

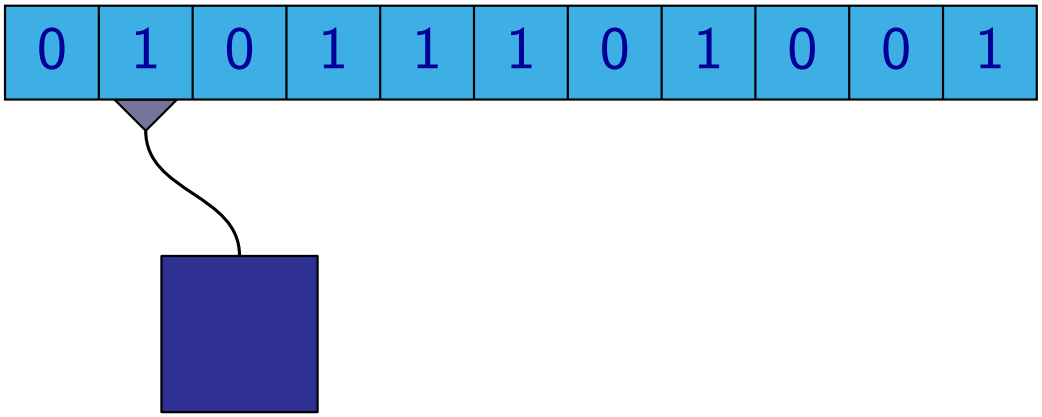


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

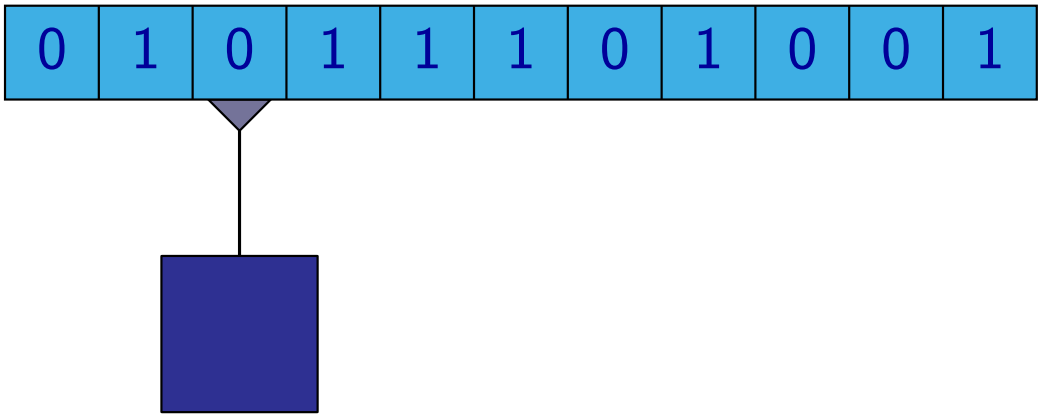


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

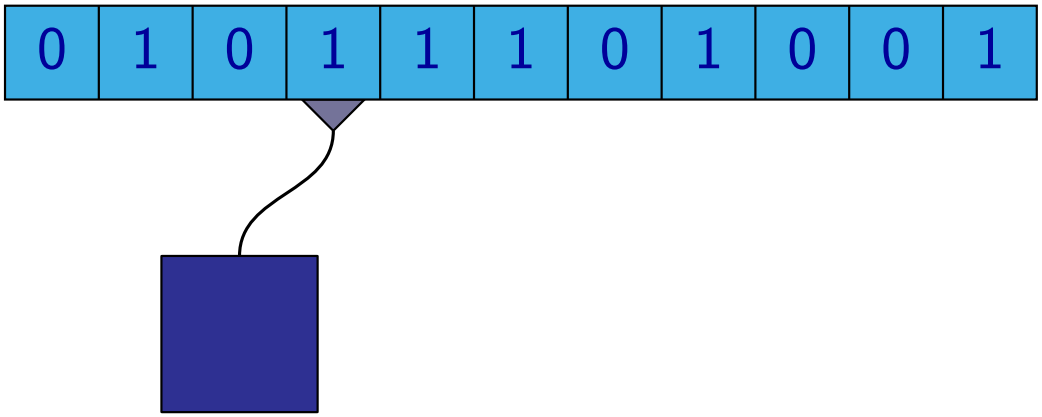


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

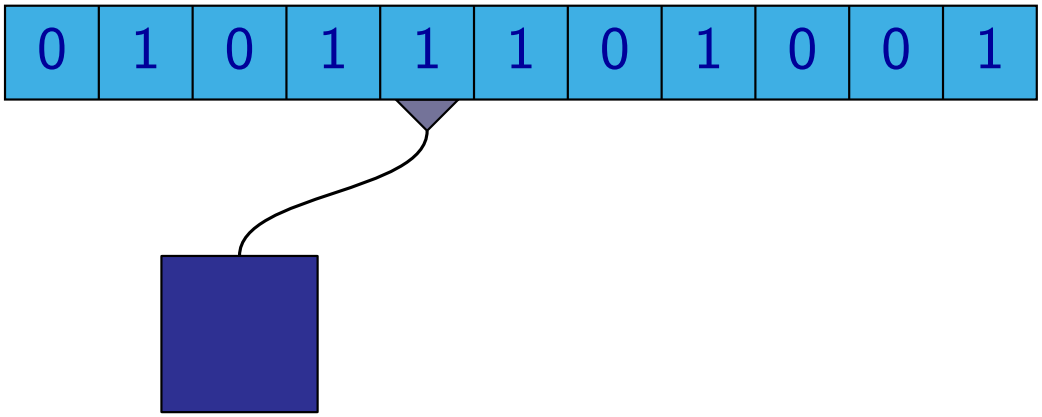


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

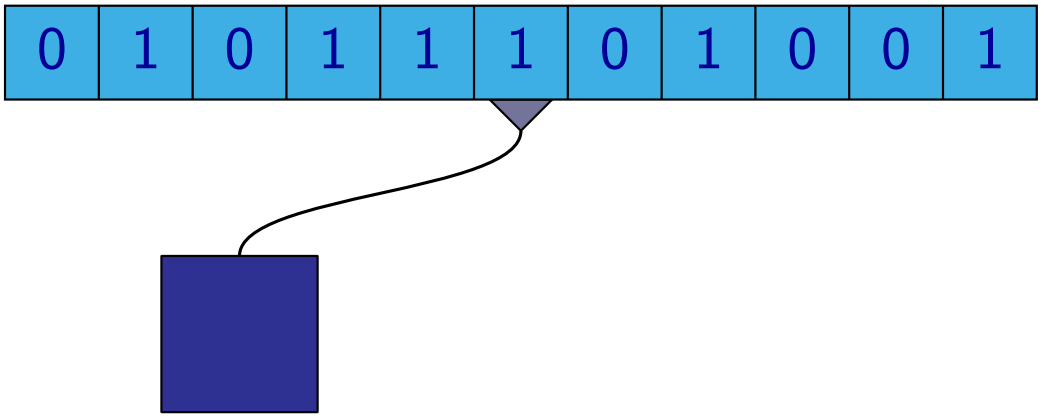


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

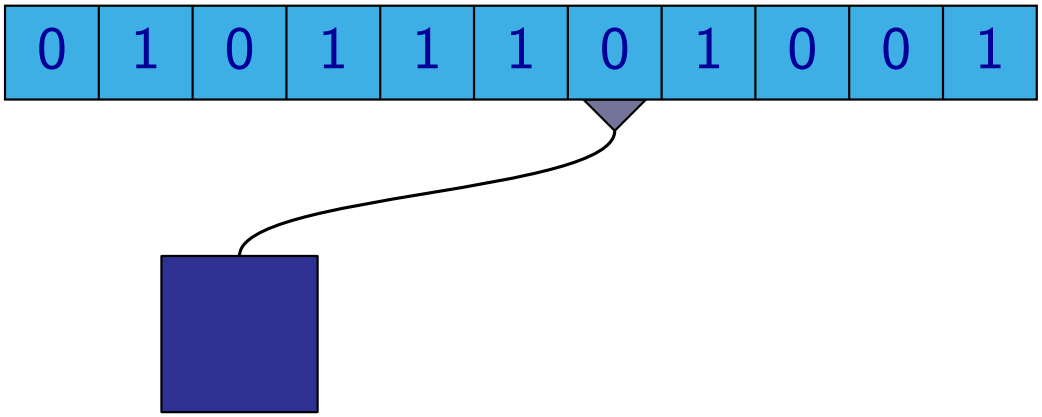


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

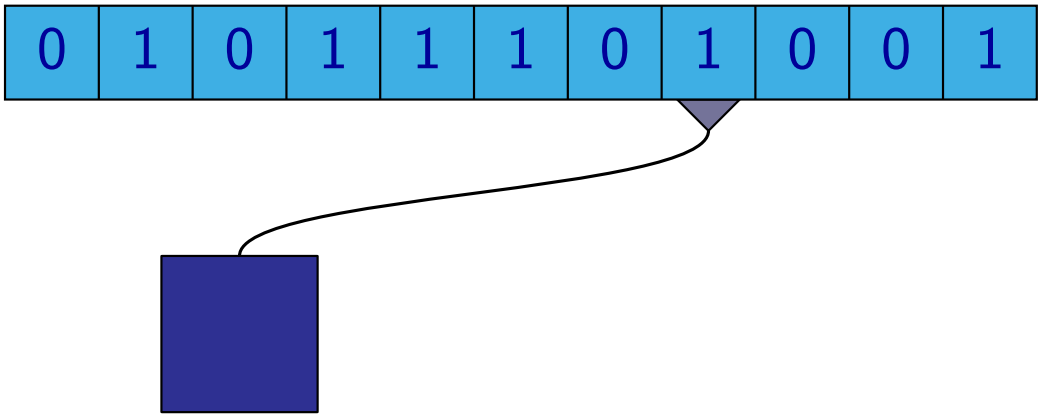


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

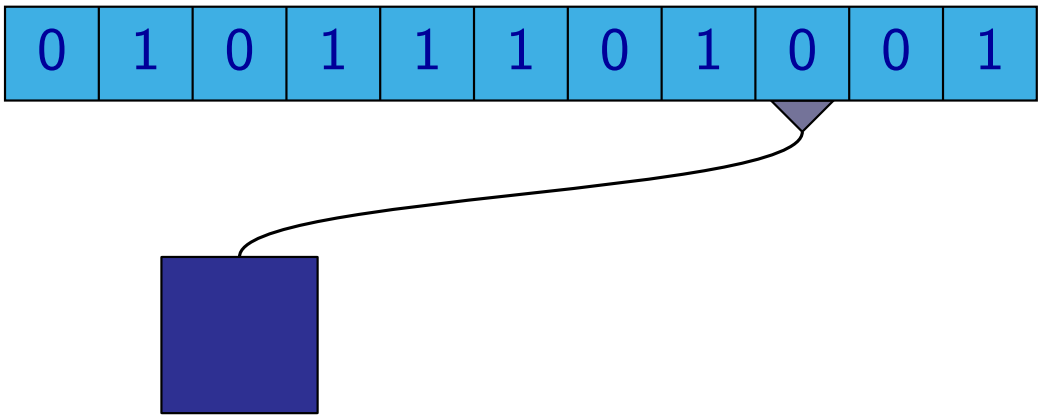


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

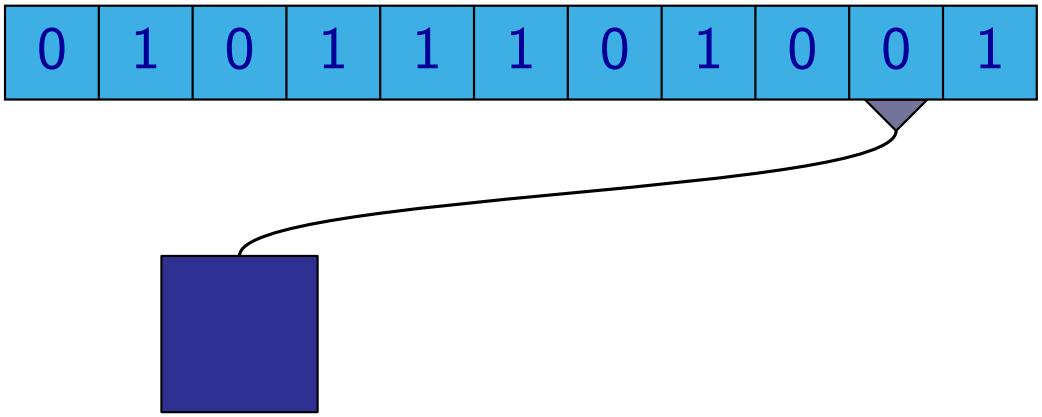


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

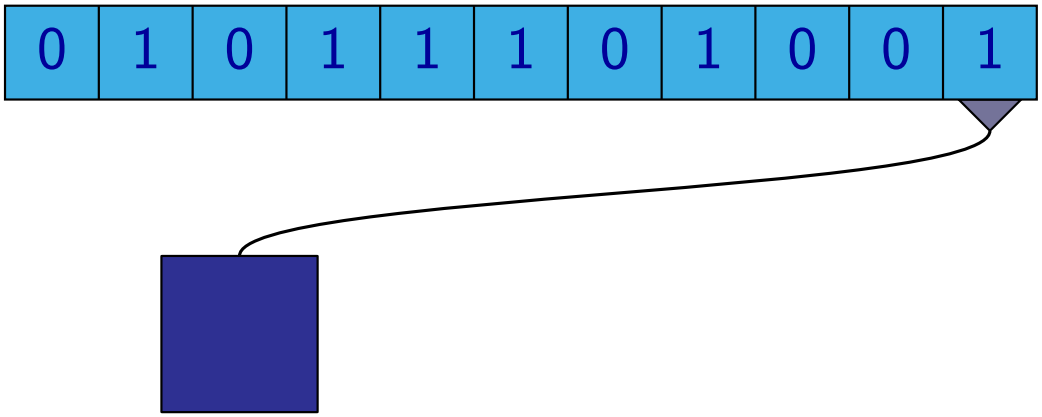


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.

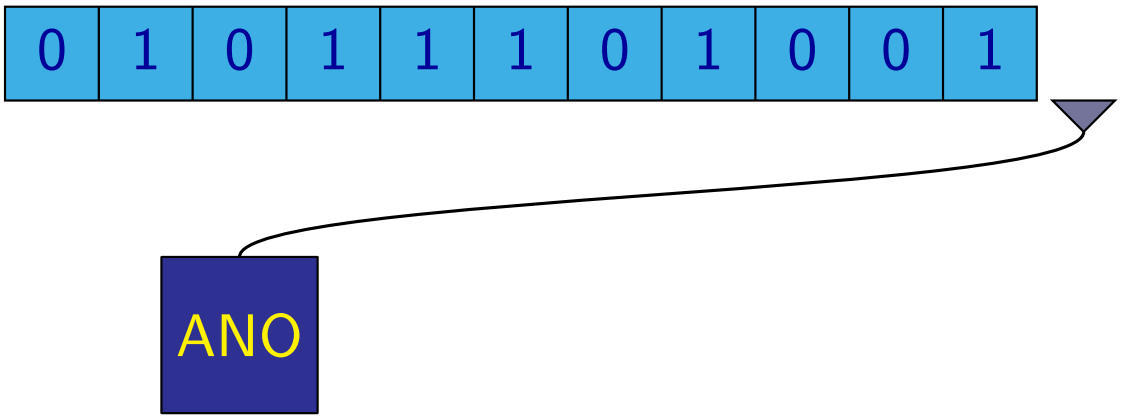


Rozpoznávání jazyka

Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

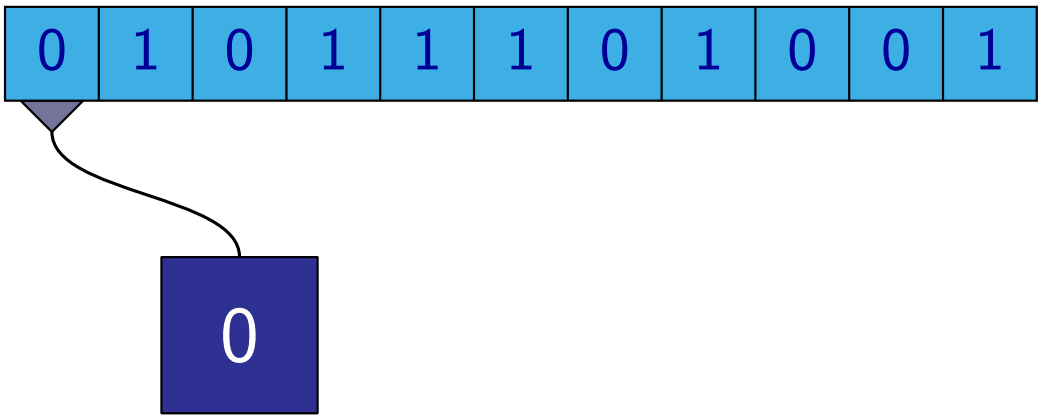
Chtěli bychom rozpoznávat jazyk L, který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1.

Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.



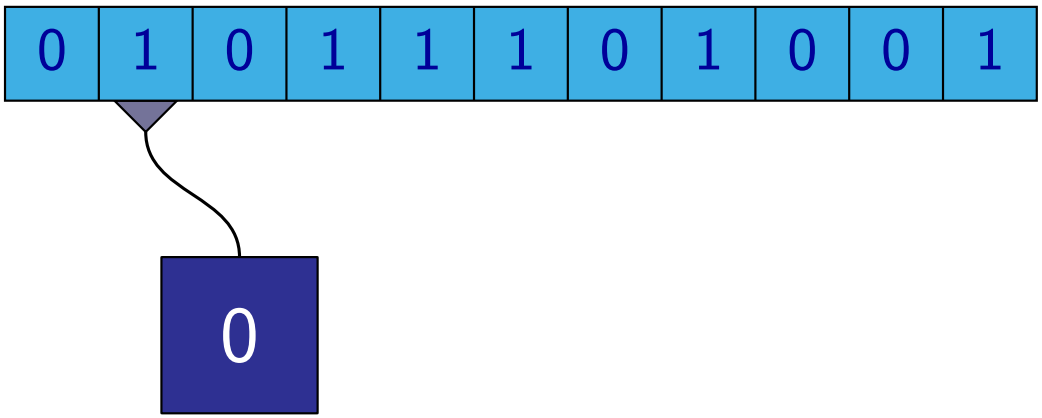
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



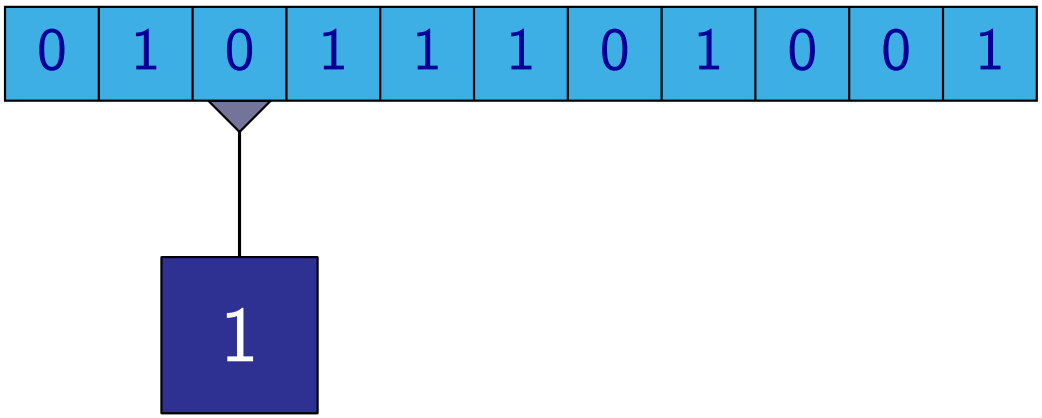
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



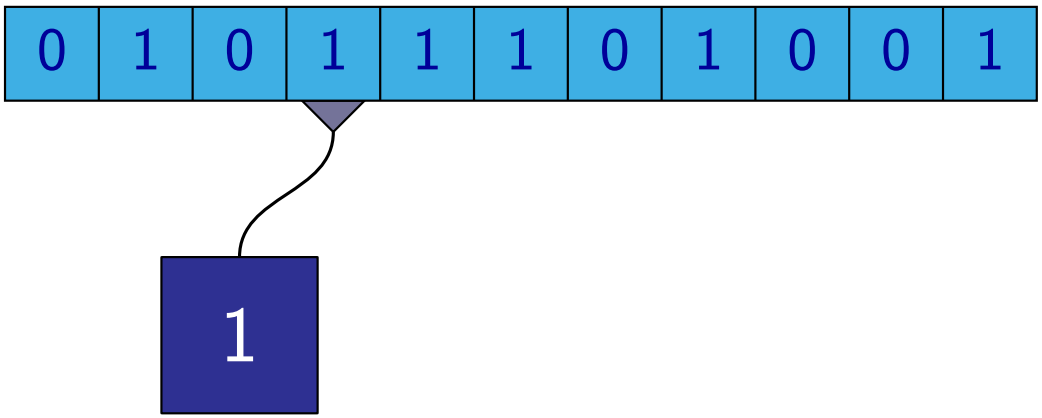
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



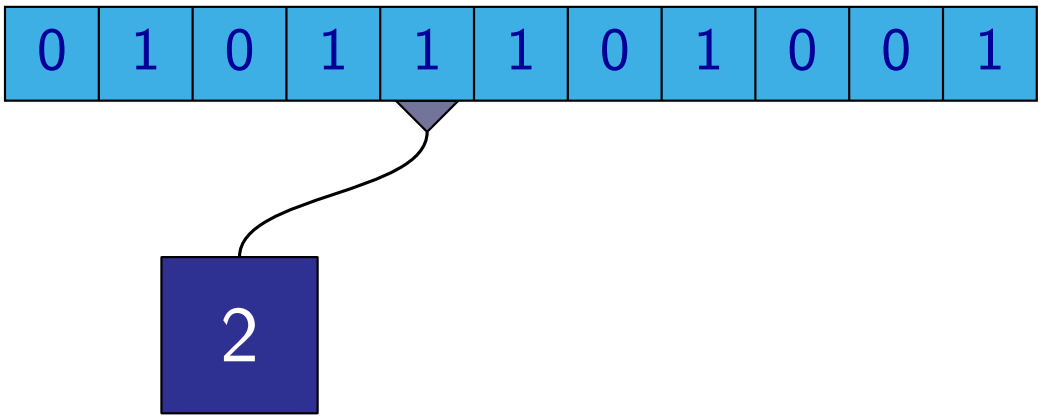
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



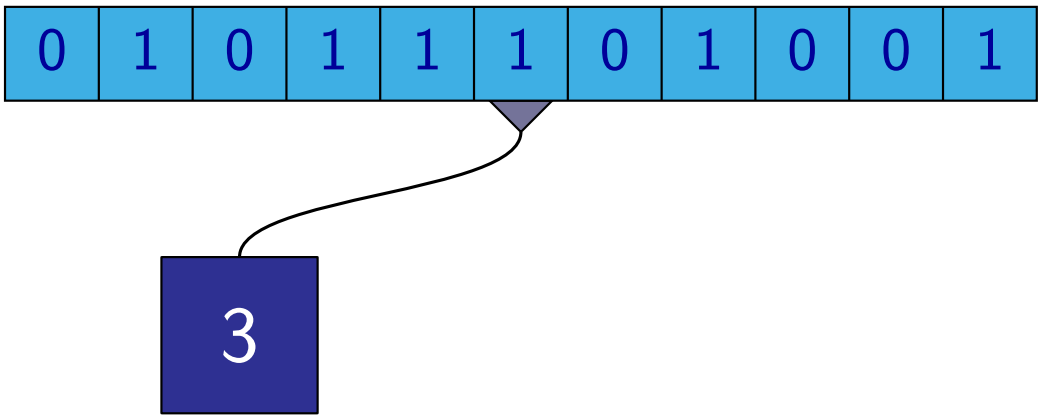
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



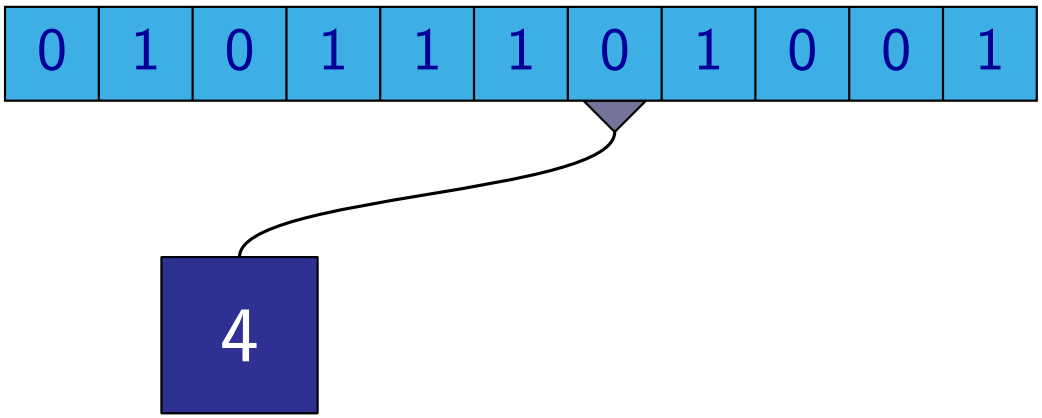
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



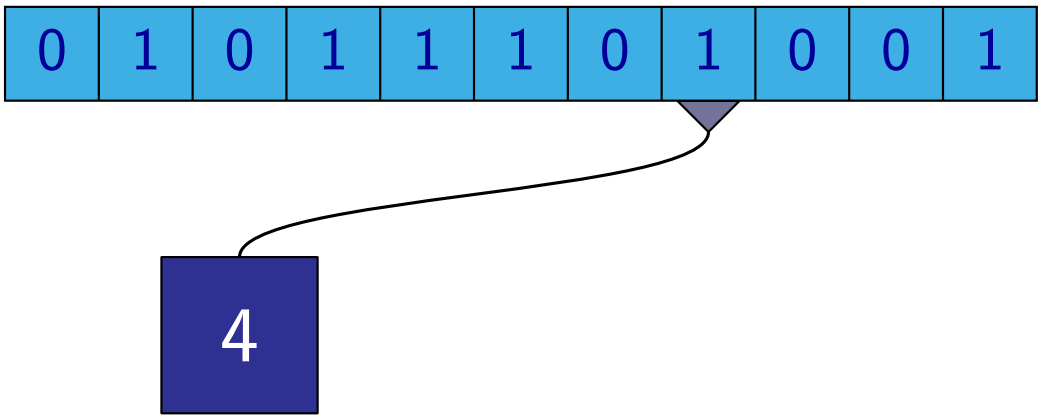
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



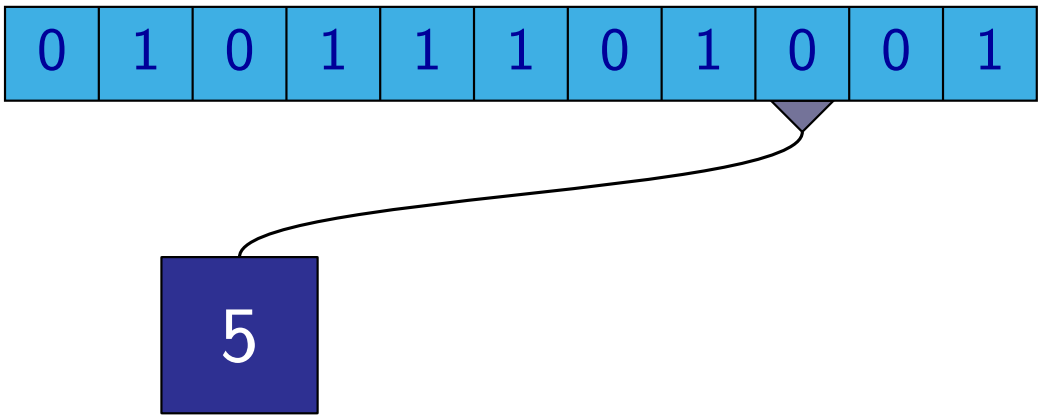
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



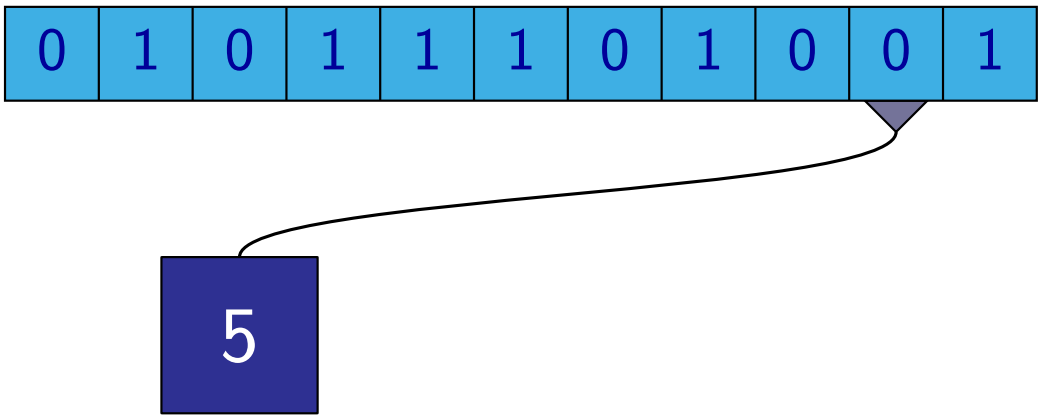
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



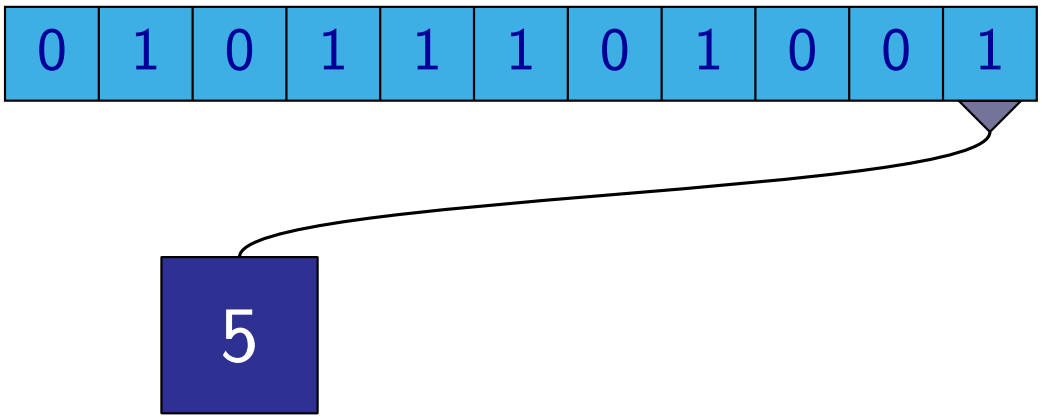
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



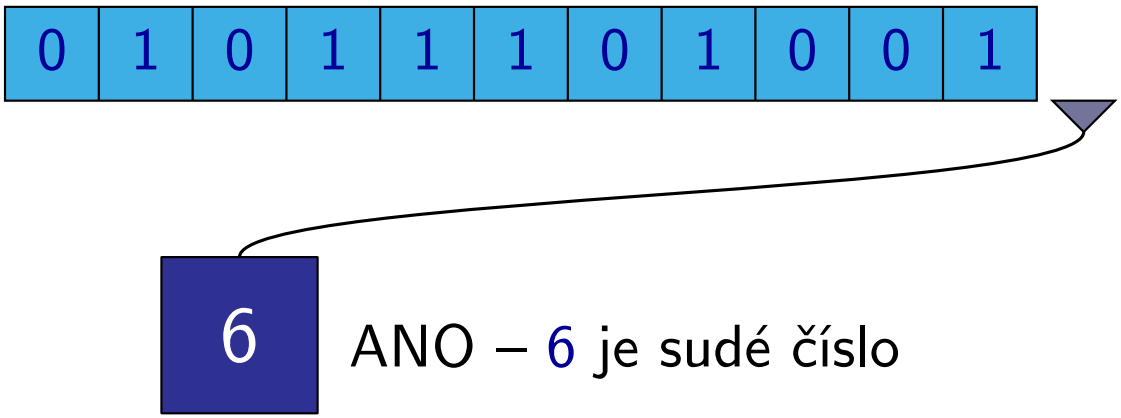
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



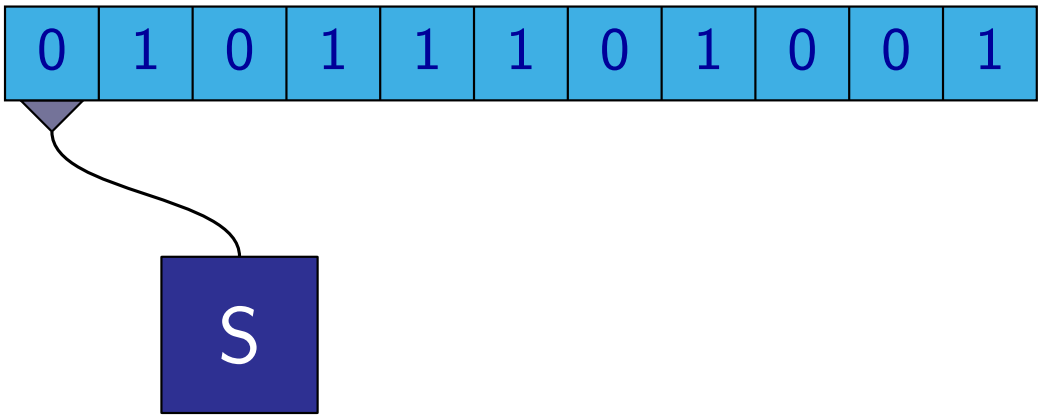
Rozpoznávání jazyka

První nápad: Počítat počet výskytů symbolů 1.



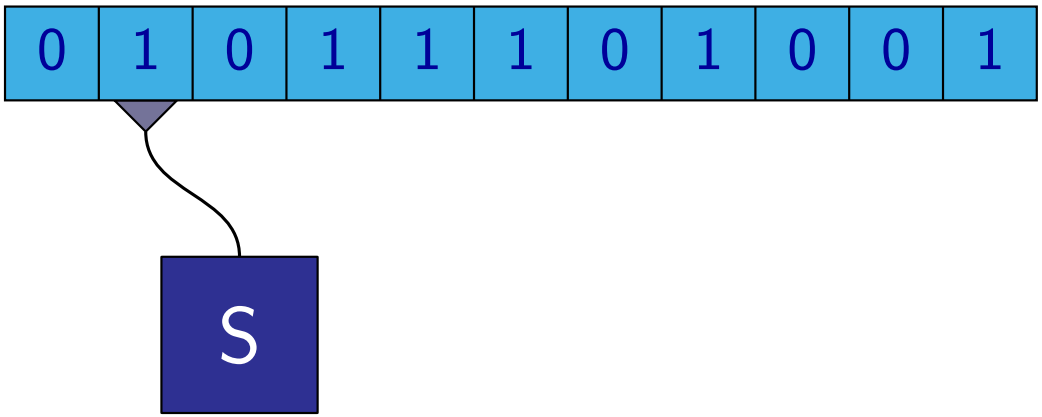
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



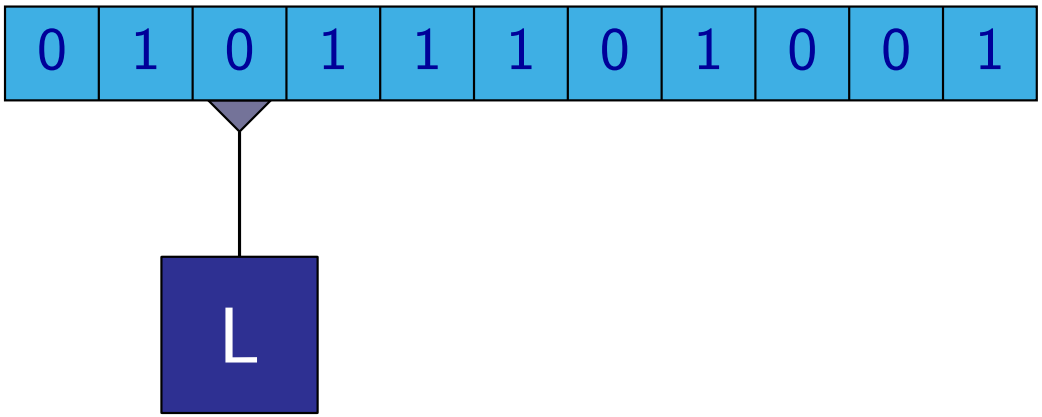
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



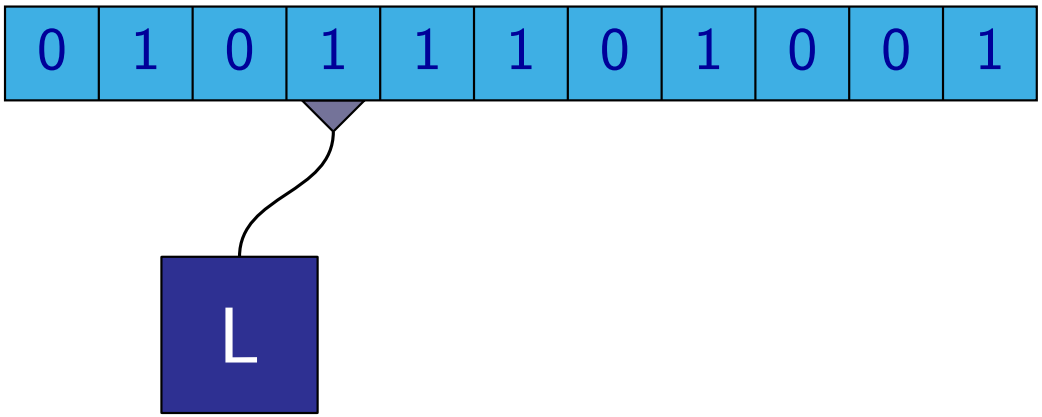
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



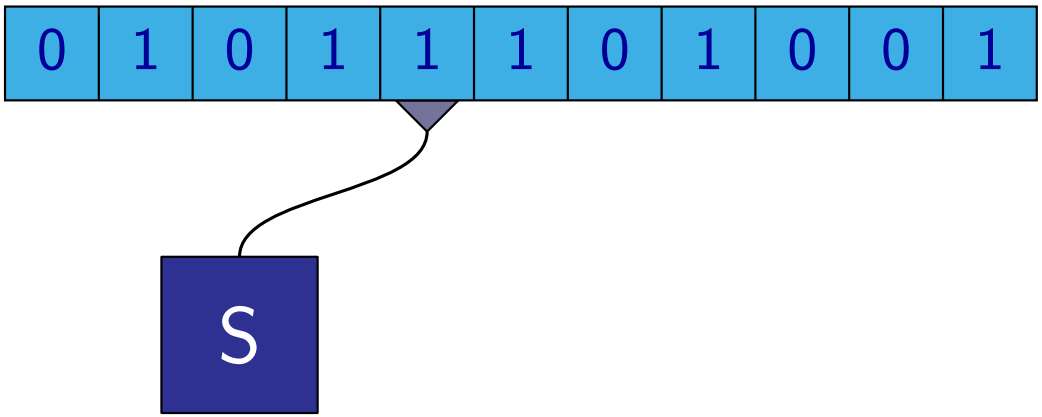
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



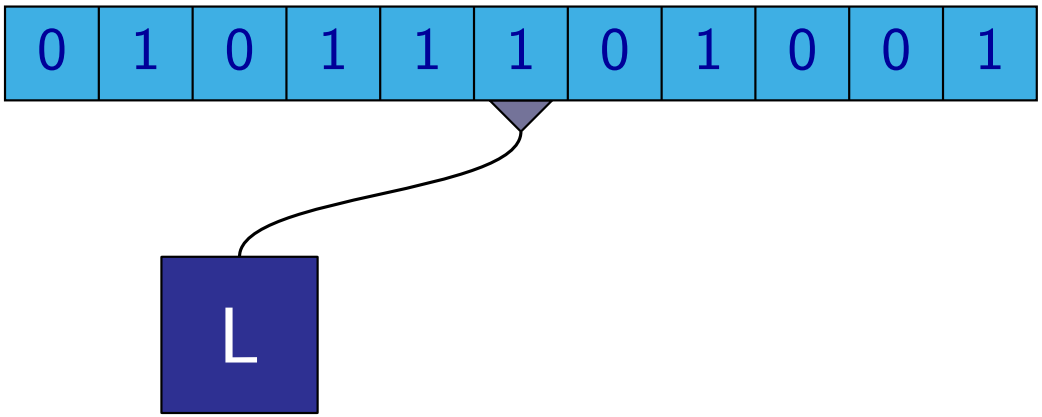
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



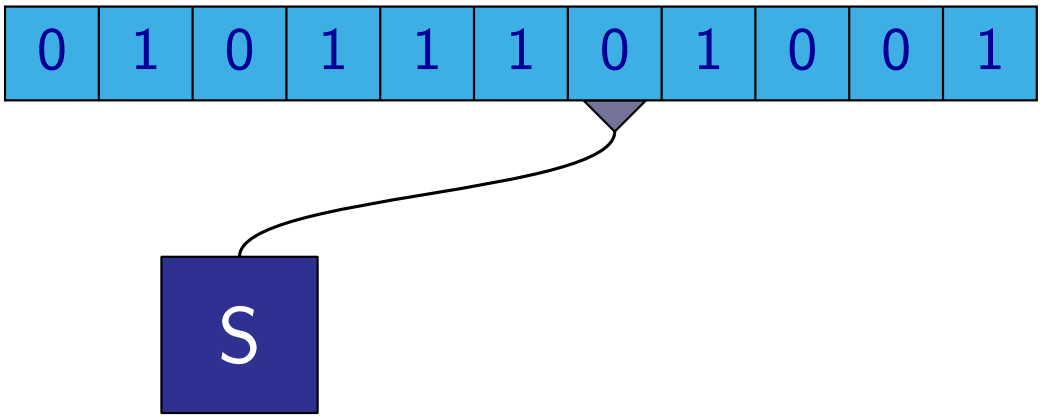
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



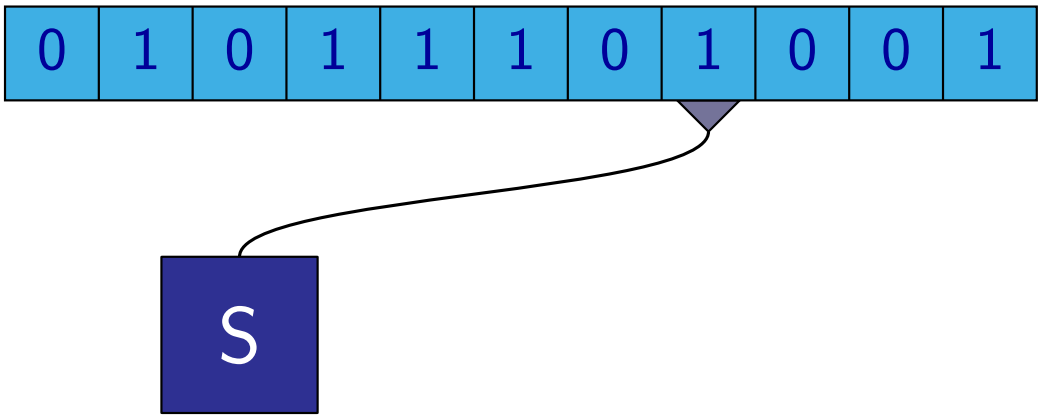
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



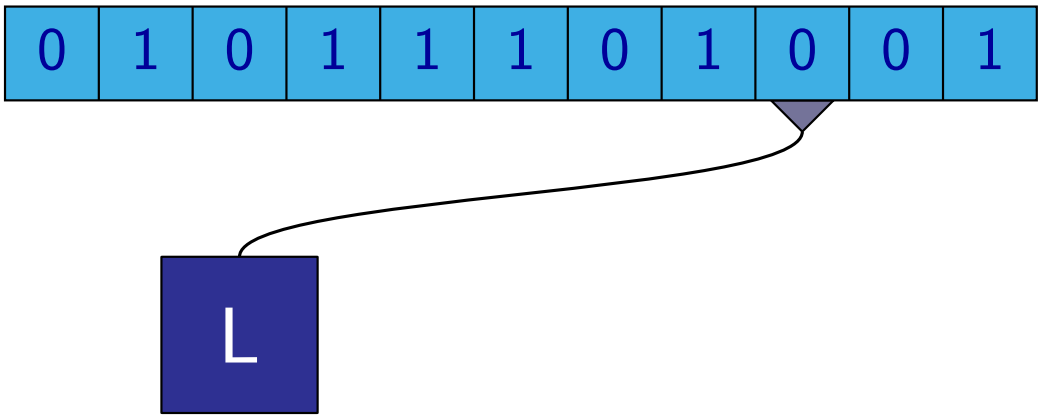
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



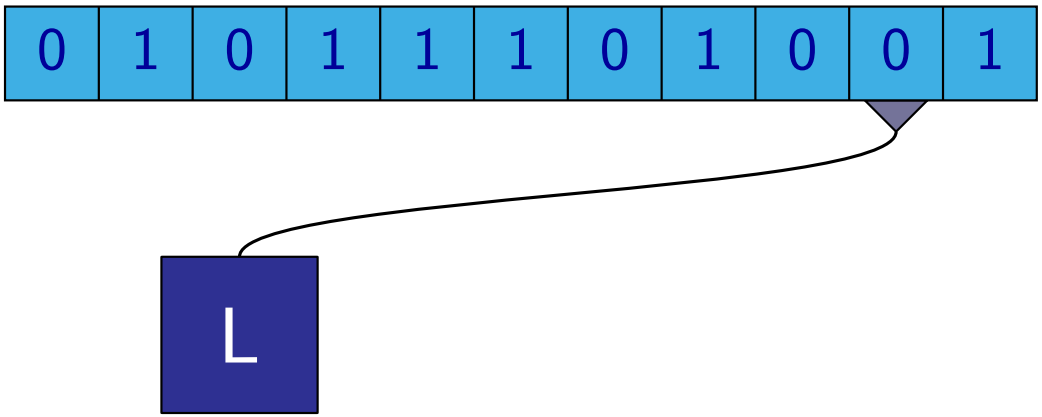
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



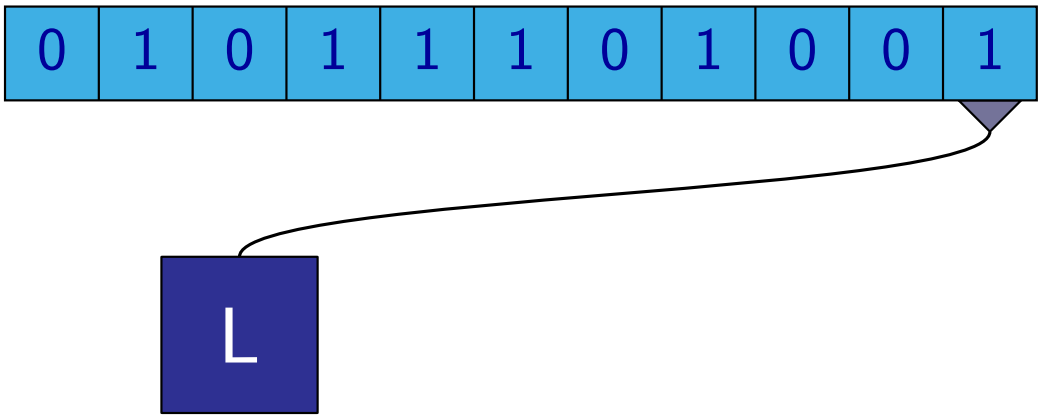
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



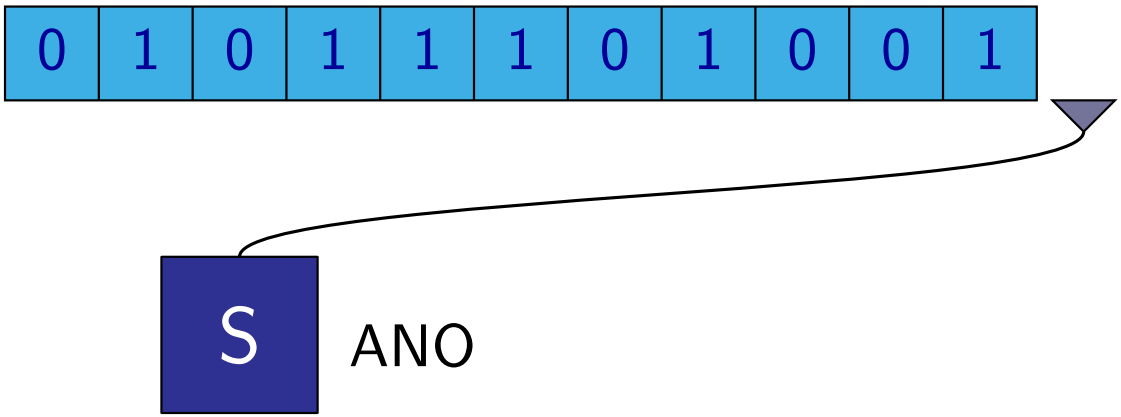
Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



Rozpoznávání jazyka

Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů 1 je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



Rozpoznávání jazyka

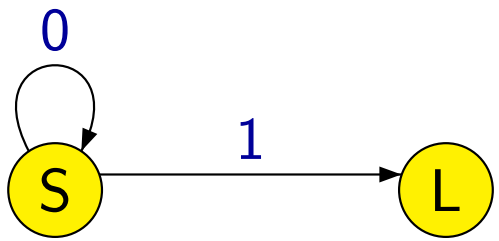
Chovaní tohoto zařízení můžeme popsat grafem:

Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:

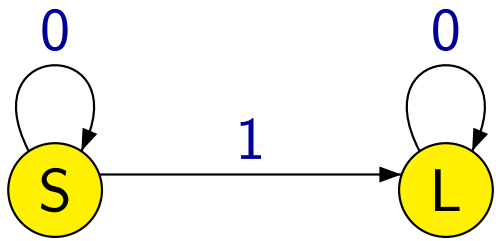
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



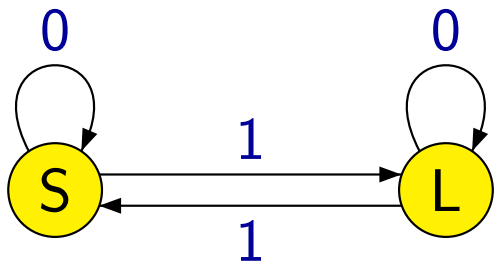
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



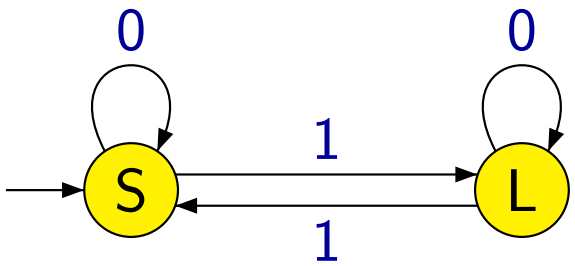
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



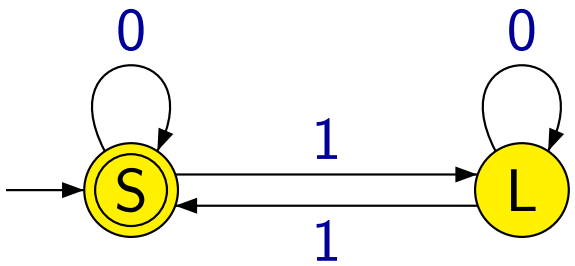
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:



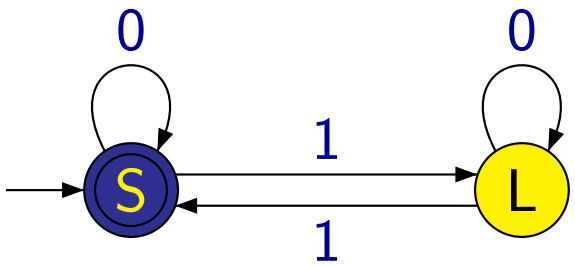
Rozpoznávání jazyka

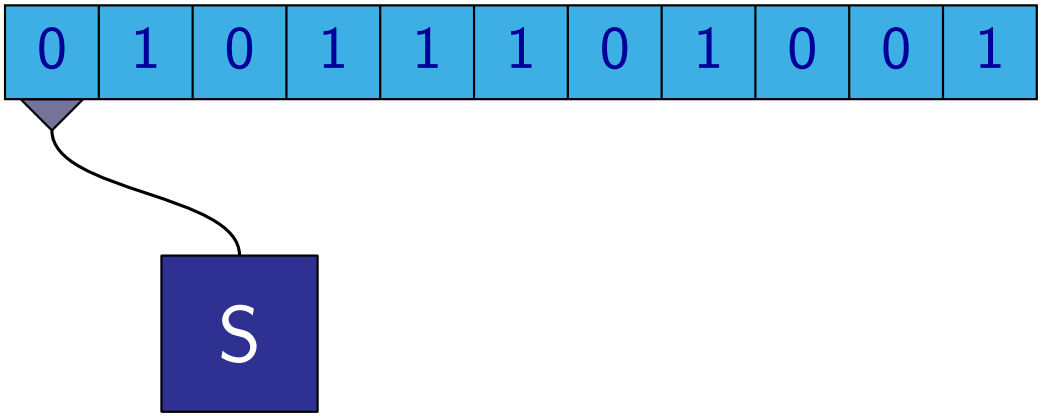
Chovaní tohoto zařízení můžeme popsat grafem:



Rozpoznávání jazyka

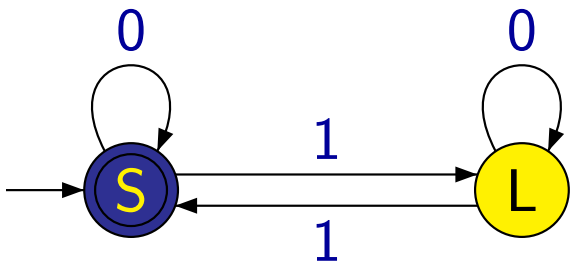
Chovaní tohoto zařízení můžeme popsat grafem:

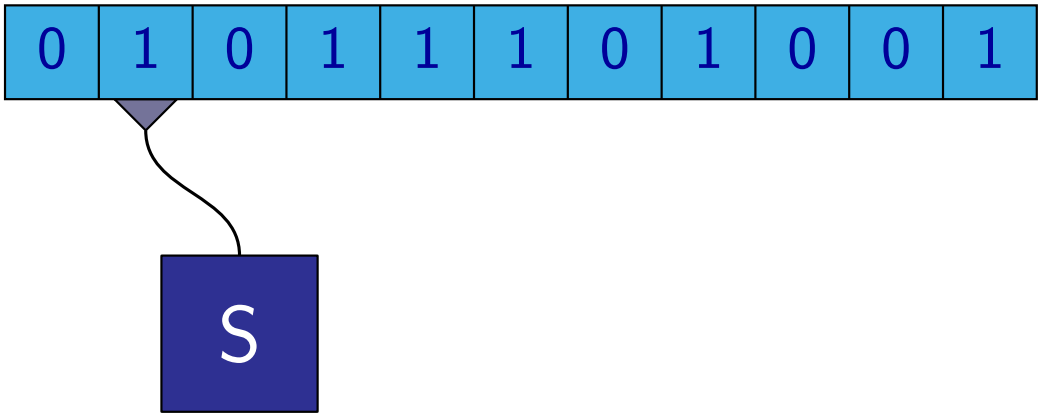




Rozpoznávání jazyka

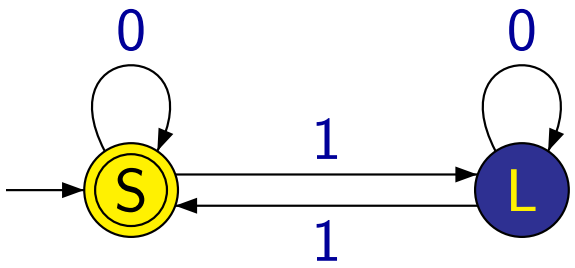
Chovaní tohoto zařízení můžeme popsat grafem:

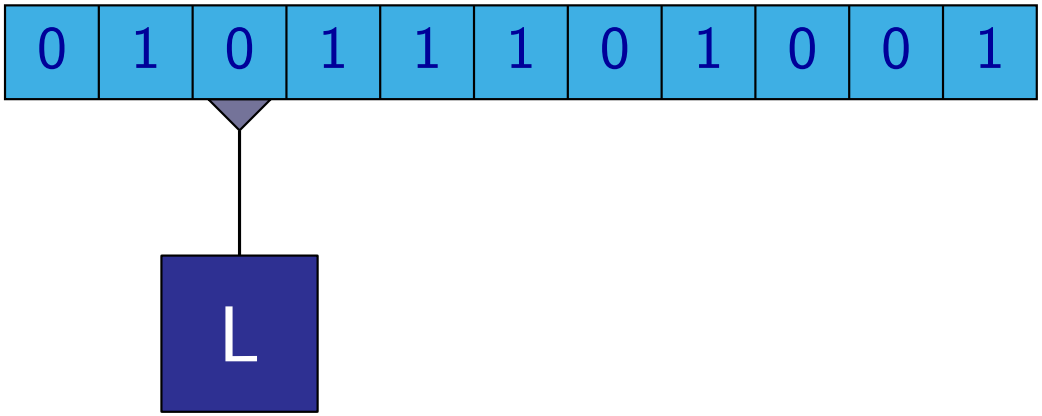




Rozpoznávání jazyka

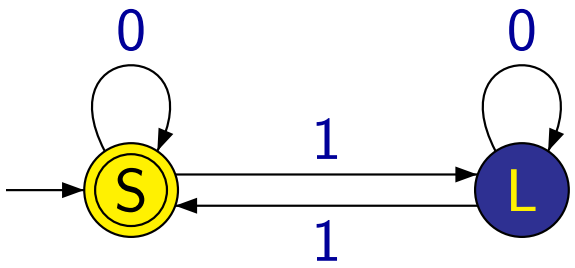
Chovaní tohoto zařízení můžeme popsat grafem:

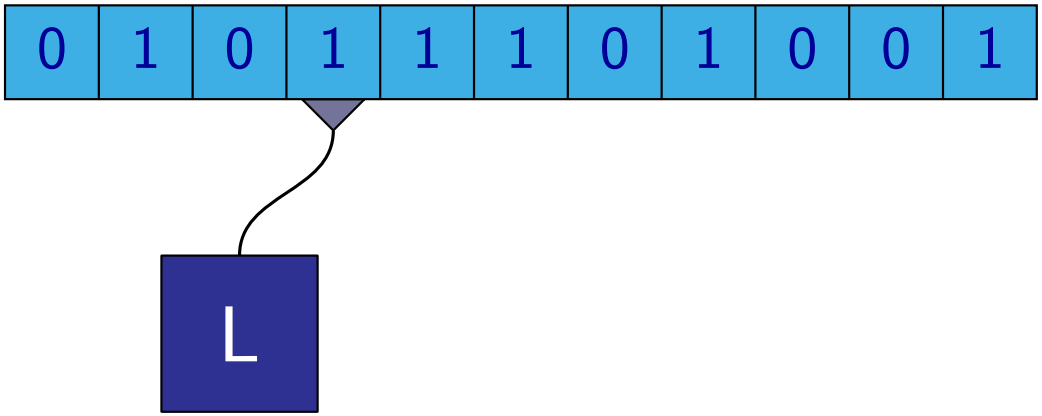




Rozpoznávání jazyka

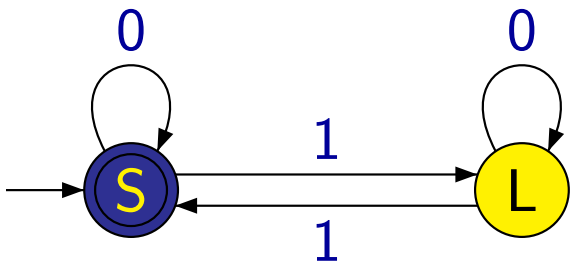
Chovaní tohoto zařízení můžeme popsat grafem:

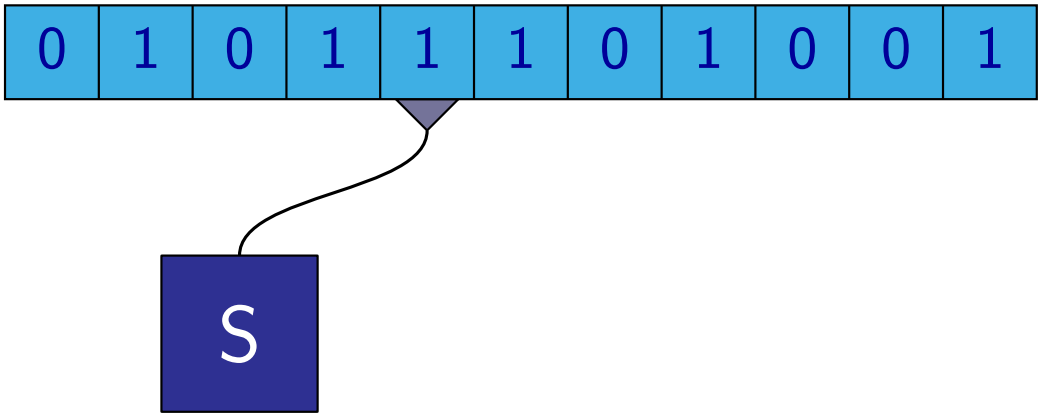




Rozpoznávání jazyka

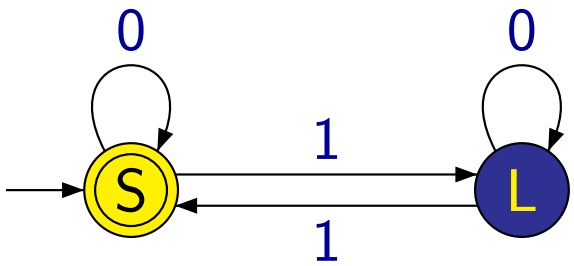
Chovaní tohoto zařízení můžeme popsat grafem:

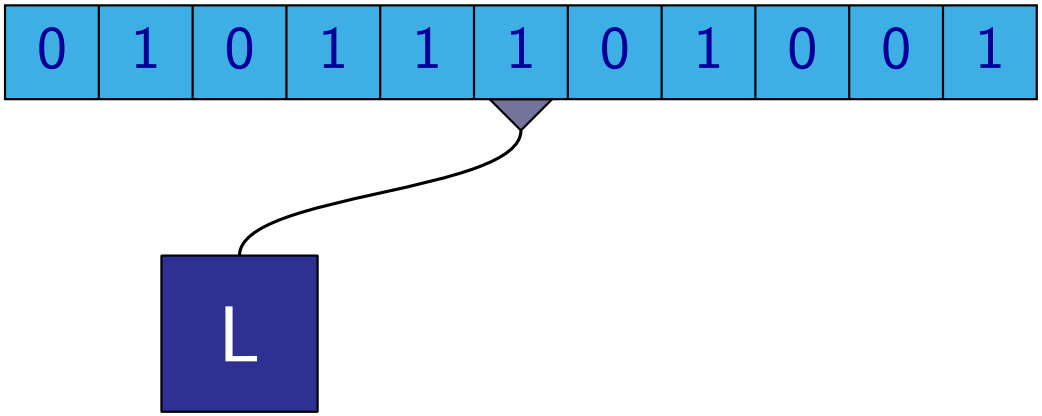




Rozpoznávání jazyka

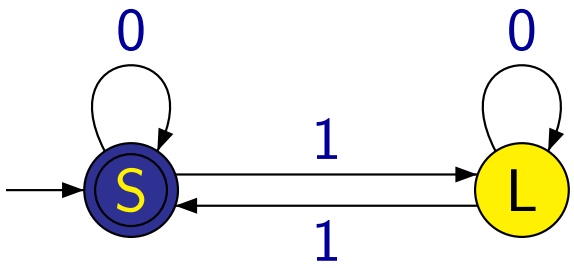
Chovaní tohoto zařízení můžeme popsat grafem:

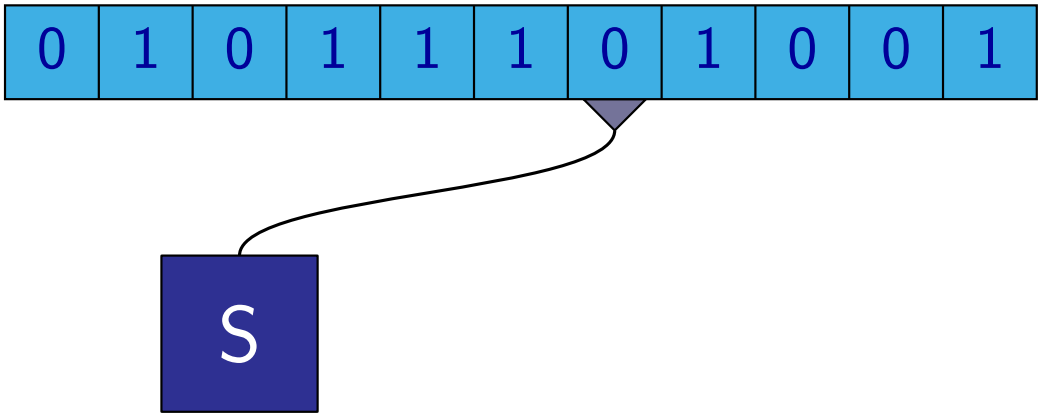




Rozpoznávání jazyka

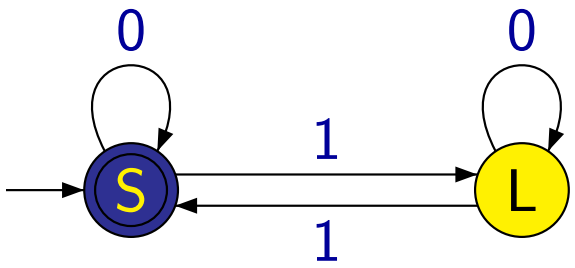
Chovaní tohoto zařízení můžeme popsat grafem:

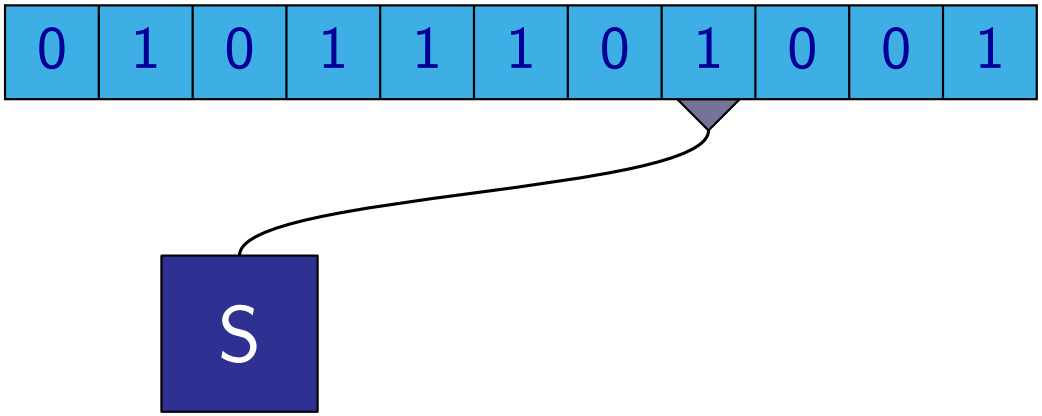




Rozpoznávání jazyka

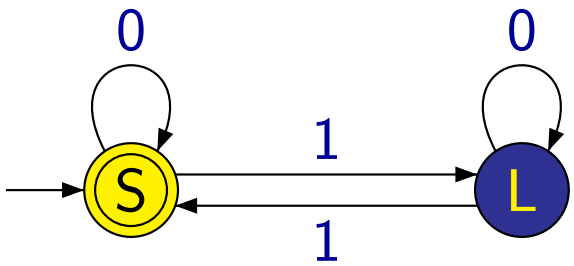
Chovaní tohoto zařízení můžeme popsat grafem:

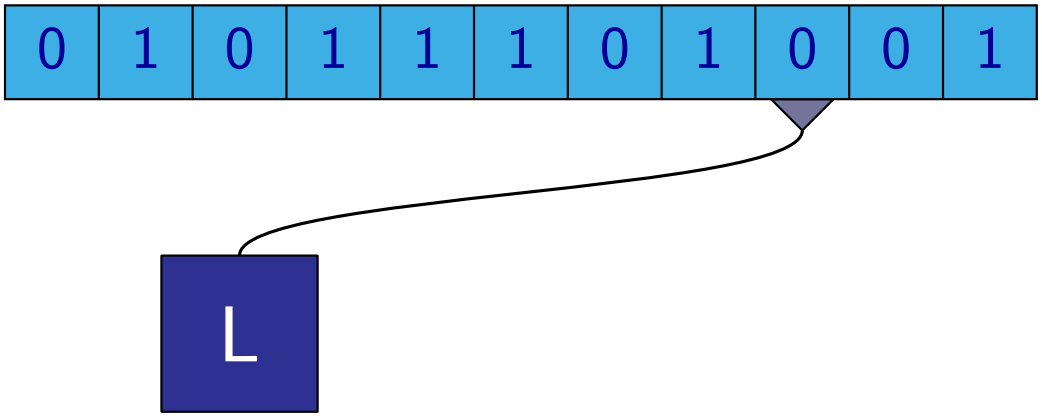




Rozpoznávání jazyka

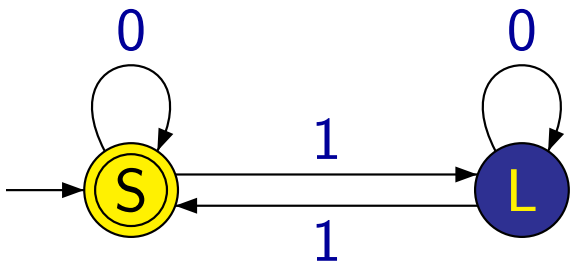
Chovaní tohoto zařízení můžeme popsat grafem:

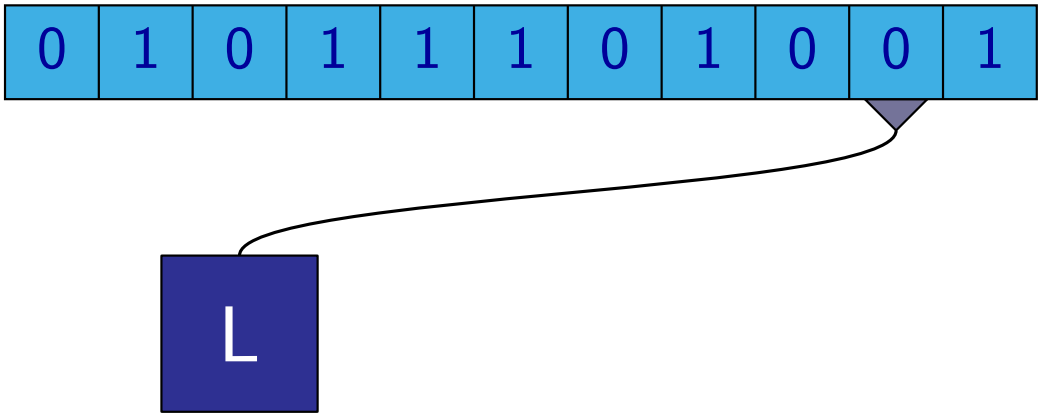




Rozpoznávání jazyka

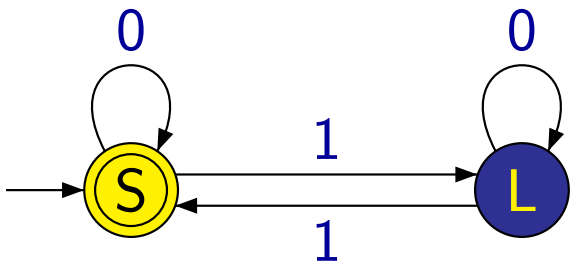
Chovaní tohoto zařízení můžeme popsat grafem:

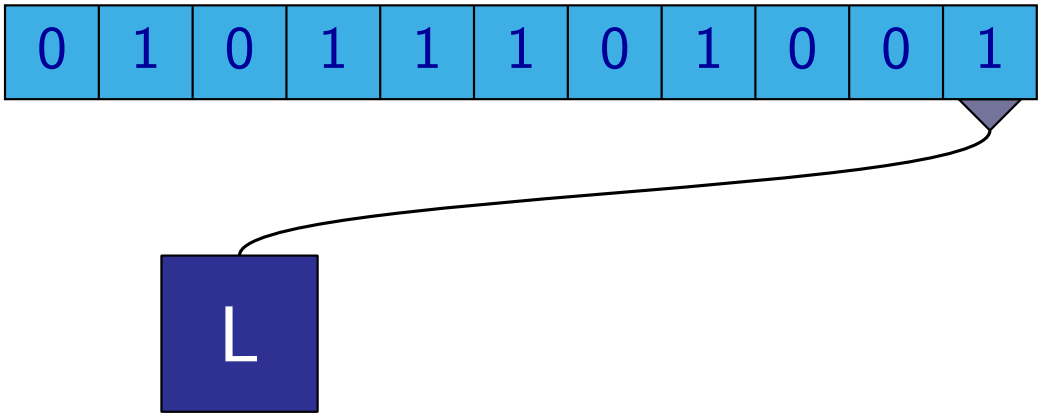




Rozpoznávání jazyka

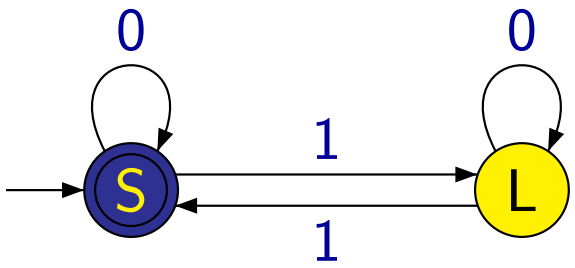
Chovaní tohoto zařízení můžeme popsat grafem:

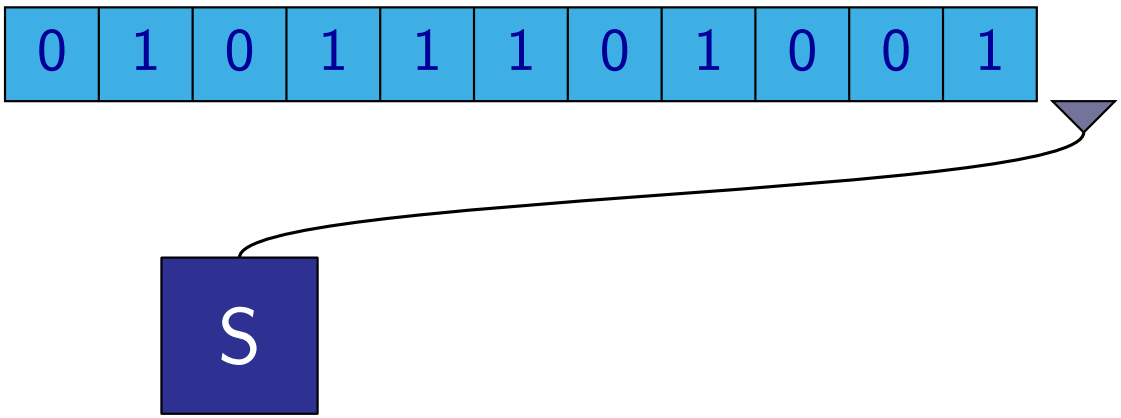




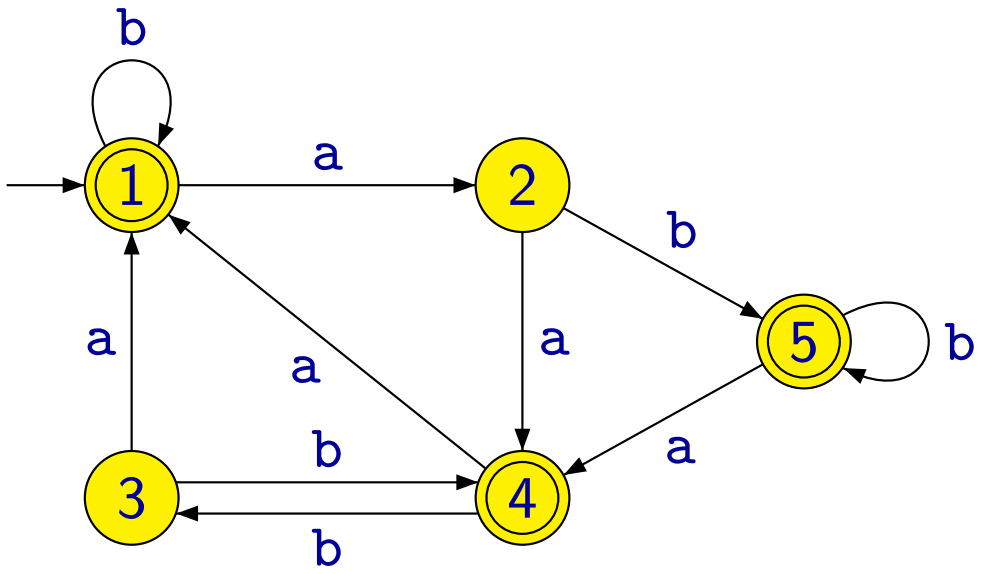
Rozpoznávání jazyka

Chovaní tohoto zařízení můžeme popsat grafem:





Deterministický konečný automat



Deterministický konečný automat se skládá ze **stavů** a **přechodů**. Jeden ze stavů je označen jako **počáteční stav** a některé ze stavů jsou označeny jako **přijímající**.

Deterministický konečný automat

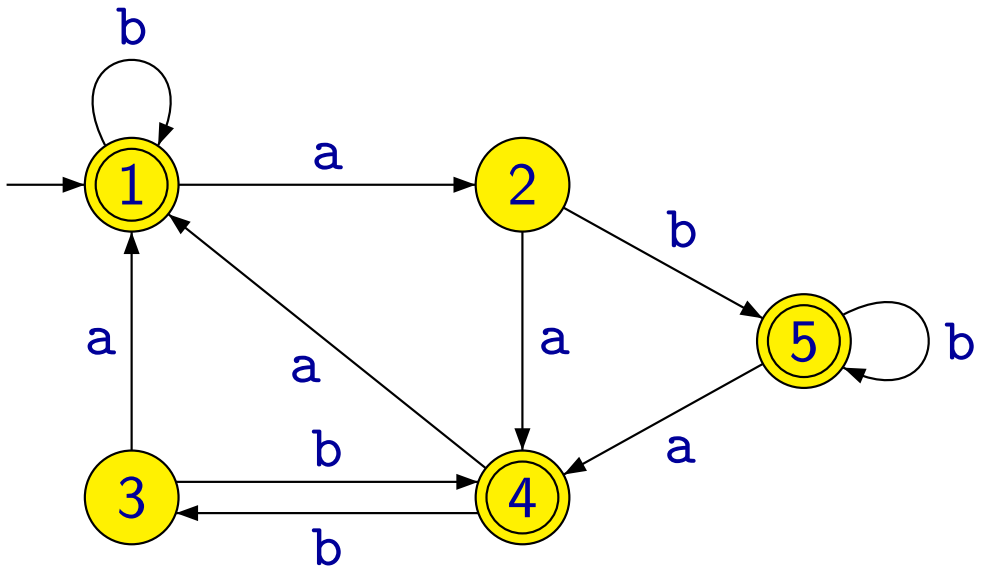
Formálně je **deterministický konečný automat** (**DKA**) definován jako pětice

$$(Q, \Sigma, \delta, q_0, F)$$

kde:

- Q je neprázdná konečná množina **stavů**
- Σ je **abeceda** (neprázdná konečná množina symbolů)
- $\delta : Q \times \Sigma \to Q$ je **přechodová funkce**
- $q_0 \in Q$ je **počáteční stav**
- $F \subseteq Q$ je množina **přijímajících stavů**

Deterministický konečný automat



- $Q = \{1, 2, 3, 4, 5\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
- $q_0 = 1$
- $F = \{1, 4, 5\}$

$$\delta(1, \mathtt{a}) = 2 \qquad \delta(1, \mathtt{b}) = 1$$
$$\delta(2, \mathtt{a}) = 4 \qquad \delta(2, \mathtt{b}) = 5$$
$$\delta(3, \mathtt{a}) = 1 \qquad \delta(3, \mathtt{b}) = 4$$
$$\delta(4, \mathtt{a}) = 1 \qquad \delta(4, \mathtt{b}) = 3$$
$$\delta(5, \mathtt{a}) = 4 \qquad \delta(5, \mathtt{b}) = 5$$

Deterministický konečný automat

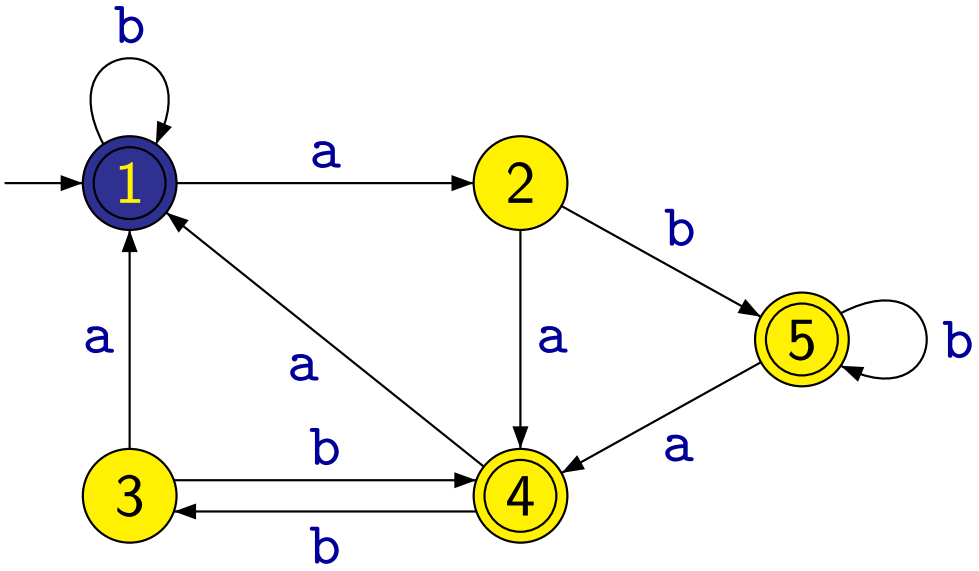
Místo zápisu

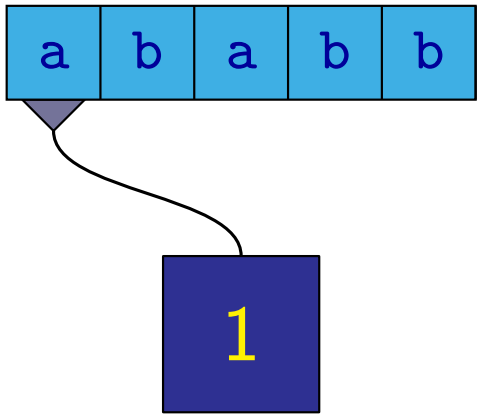
$$\begin{array}{ll} \delta(1,\mathtt{a}) = 2 & \delta(1,\mathtt{b}) = 1 \\ \delta(2,\mathtt{a}) = 4 & \delta(2,\mathtt{b}) = 5 \\ \delta(3,\mathtt{a}) = 1 & \delta(3,\mathtt{b}) = 4 \\ \delta(4,\mathtt{a}) = 1 & \delta(4,\mathtt{b}) = 3 \\ \delta(5,\mathtt{a}) = 4 & \delta(5,\mathtt{b}) = 5 \end{array}$$

budeme raději používat stručnější tabulku nebo grafické znázornění:

δ	a	b
$\leftrightarrow 1$	2	1
2	4	5
3	1	4
$\leftarrow 4$	1	3
$\leftarrow 5$	4	5

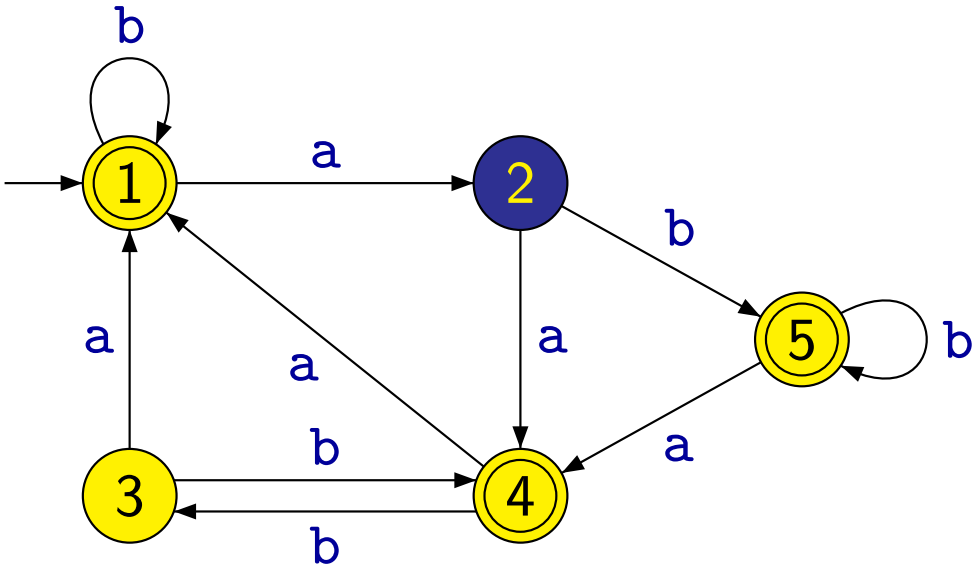
Deterministický konečný automat

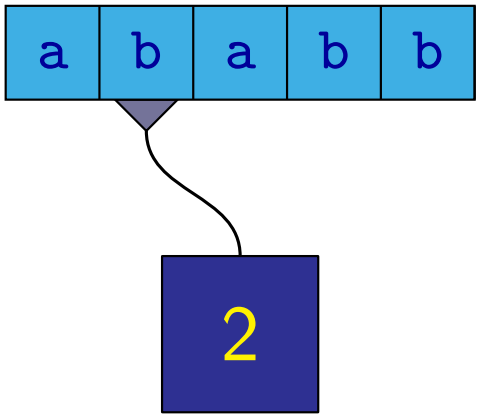




1

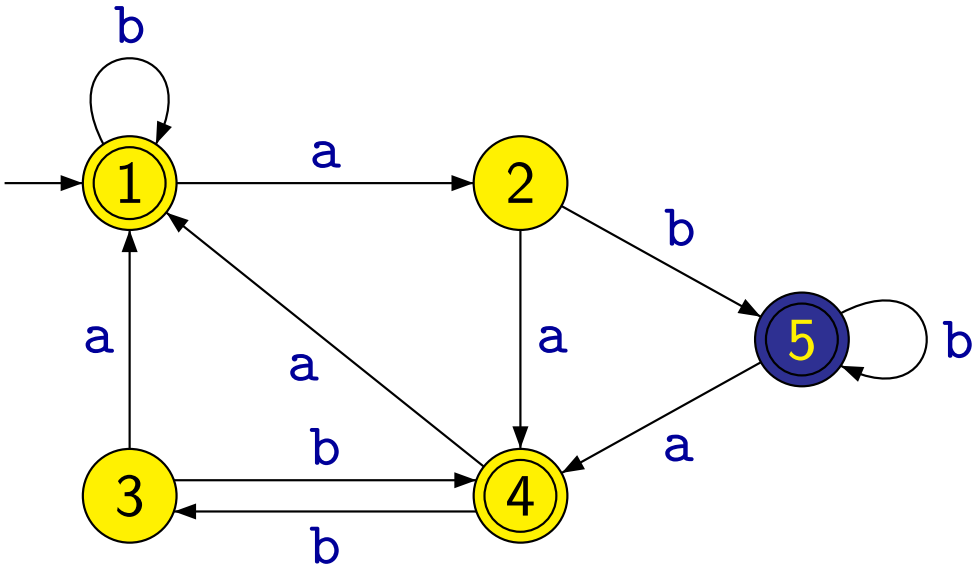
Deterministický konečný automat

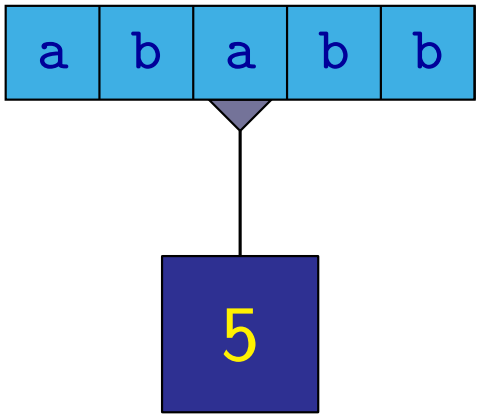




$1 \xrightarrow{a} 2$

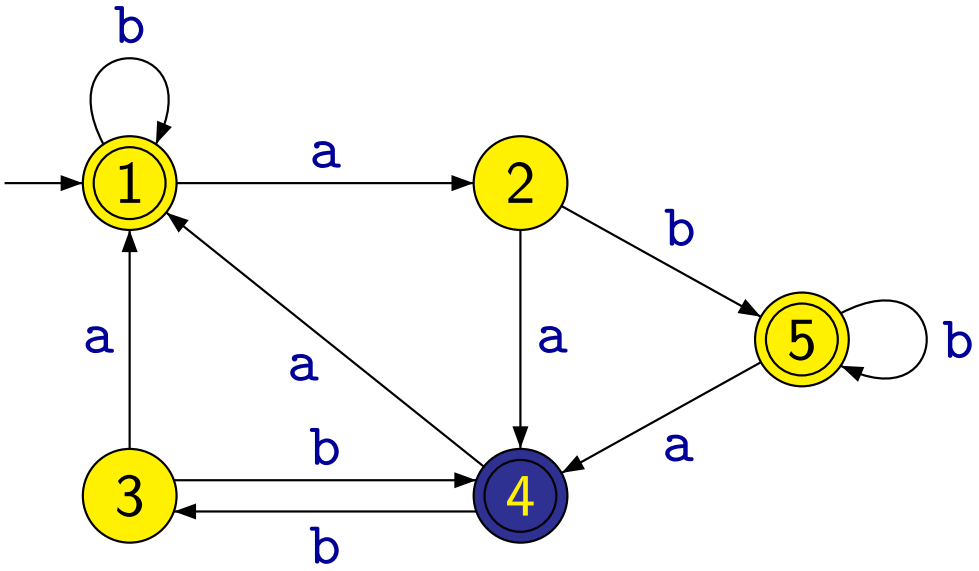
Deterministický konečný automat

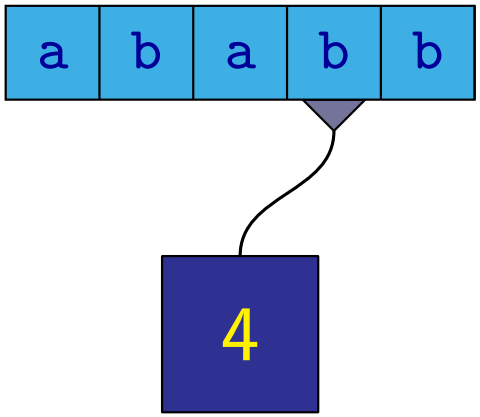




$1 \xrightarrow{a} 2 \xrightarrow{b} 5$

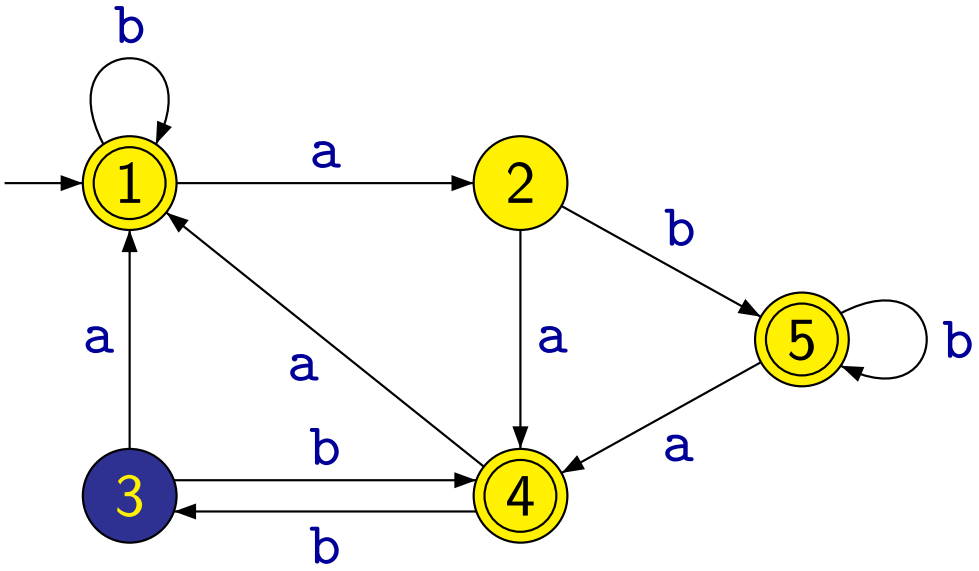
Deterministický konečný automat

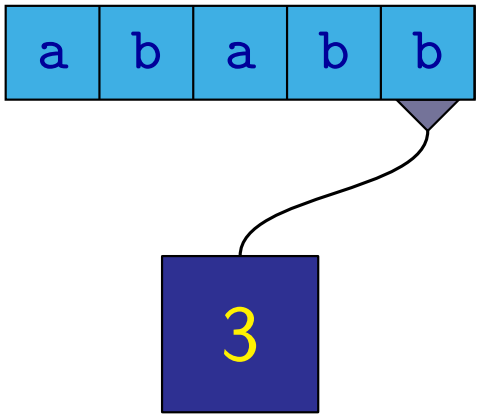




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4$$

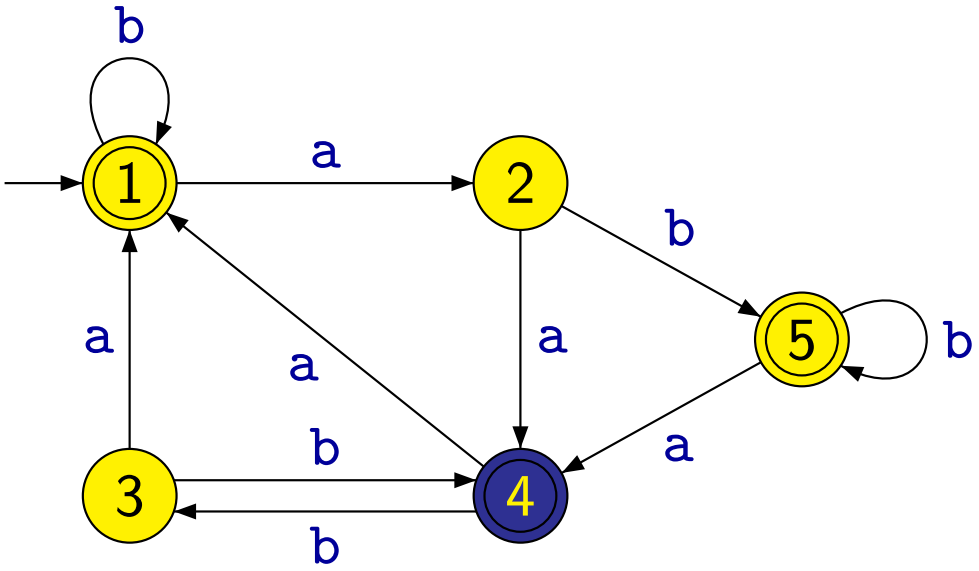
Deterministický konečný automat

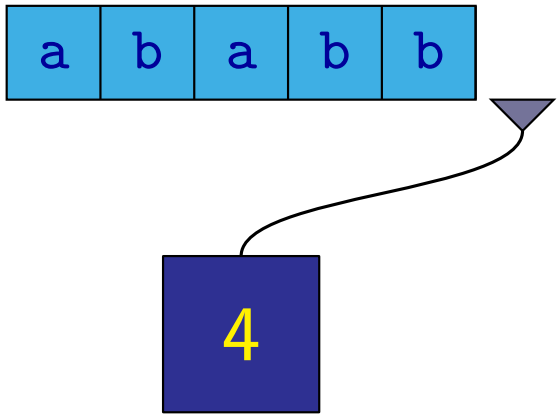




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3$$

Deterministický konečný automat





$$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3 \xrightarrow{b} 4$$

Deterministický konečný automat

Definice

Mějme DKA $A = (Q, \Sigma, \delta, q_0, F)$.

Zápisem $q \xrightarrow{w} q'$, kde $q, q' \in Q$ a $w \in \Sigma^*$, budeme označovat to, že pokud je automat ve stavu q, tak přečtením slova w přejde do stavu q'.

Poznámka: $\longrightarrow \subseteq Q \times \Sigma^* \times Q$ je ternární relace.

Místo $(q, w, q') \in \longrightarrow$ píšeme $q \xrightarrow{w} q'$.

Pro DKA platí, že pro libovolný stav q a libovolné slovo w existuje právě jeden stav q' takový, že $q \xrightarrow{w} q'$.

Deterministický konečný automat

Relaci $\longrightarrow$ můžeme formálně definovat následující induktivní definicí:

- $q \xrightarrow{\varepsilon} q$ pro libovolné $q \in Q$
- Pro $a \in \Sigma$ a $w \in \Sigma^*$:
  $q \xrightarrow{aw} q'$ právě tehdy, když existuje $q'' \in Q$ takové, že $\delta(q, a) = q''$ a $q'' \xrightarrow{w} q'$.

Deterministický konečný automat

Slovo $w \in \Sigma^*$ je **přijímáno** deterministickým konečným automatem $A = (Q, \Sigma, \delta, q_0, F)$ právě tehdy, když existuje stav $q \in F$ takový, že $q_0 \xrightarrow{w} q$.

Definice

Jazyk rozpoznávaný (přijímaný) daným deterministickým konečným automatem $A = (Q, \Sigma, \delta, q_0, F)$, označovaný $L(A)$, je množina všech slov přijímaných tímto automatem, tj.

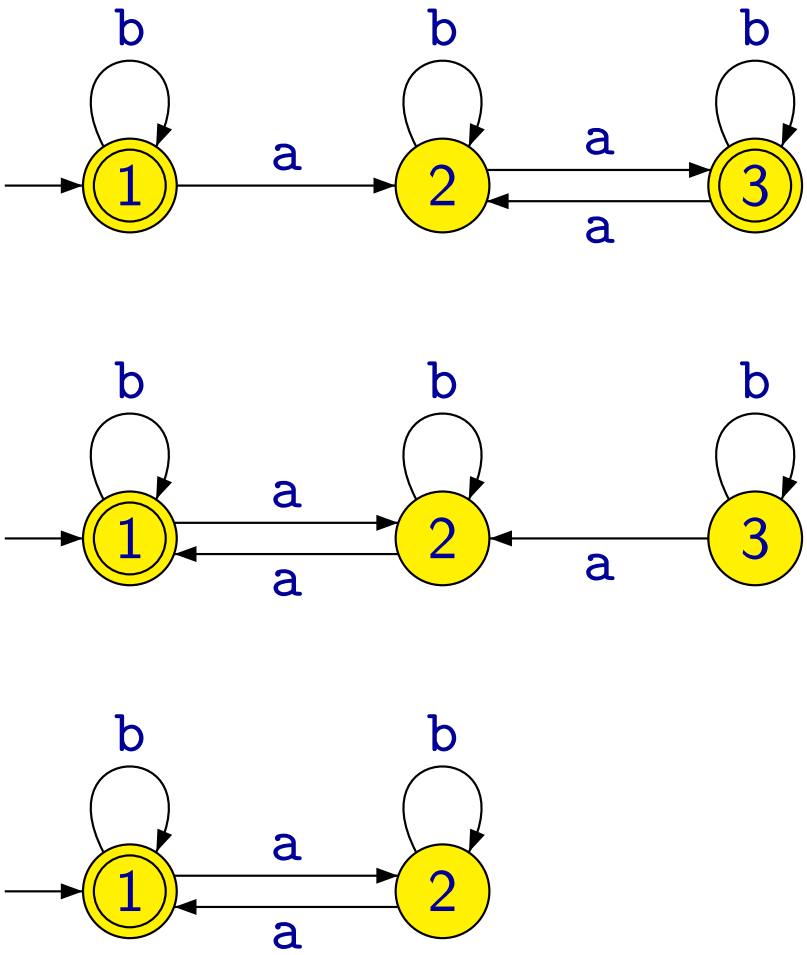
$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w} q\}$$

Regulární jazyky

Definice

Jazyk L je **regulární** právě tehdy, když existuje nějaký deterministicý konečný automat A, který jej přijímá, tj. DKA A, takový, že $L(A) = L$.

Ekvivalence automatů



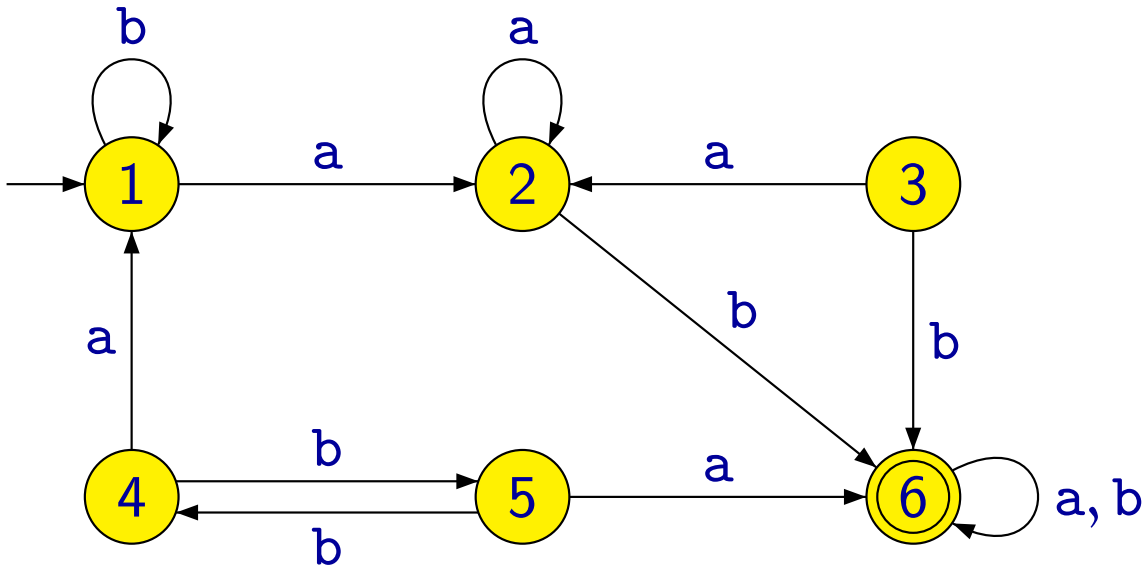
Všechny 3 automaty přijímají jazyk všech slov se sudým počtem *a*.

Ekvivalence automatů

Definice

O konečných automatech A_1, A_2 řekneme, že jsou **ekvivalentní**, jestliže $L(A_1) = L(A_2)$.

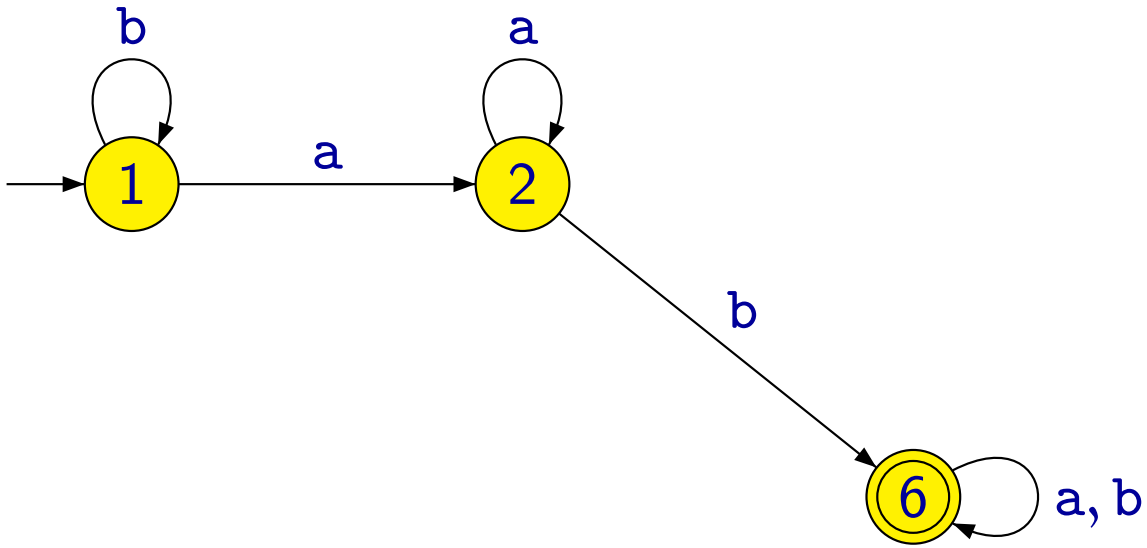
Nedosažitelné stavy automatu



- Automat přijímá jazyk $L = \{w \in \{a, b\}^* \mid w \text{ obsahuje podslovo } ab\}$
- Pro žádnou posloupnost vstupních symbolů se automat nedostane do stavů 3, 4 nebo 5.

.

Nedosažitelné stavy automatu



- Automat přijímá jazyk $L = \{w \in \{a, b\}^* \mid w \text{ obsahuje podslovo } ab\}$
- Pro žádnou posloupnost vstupních symbolů se automat nedostane do stavů 3, 4 nebo 5.
- Pokud tyto stavy odstraníme, pořád automat přijímá stejný jazyk L.

Nedosažitelné stavy automatu

Definice

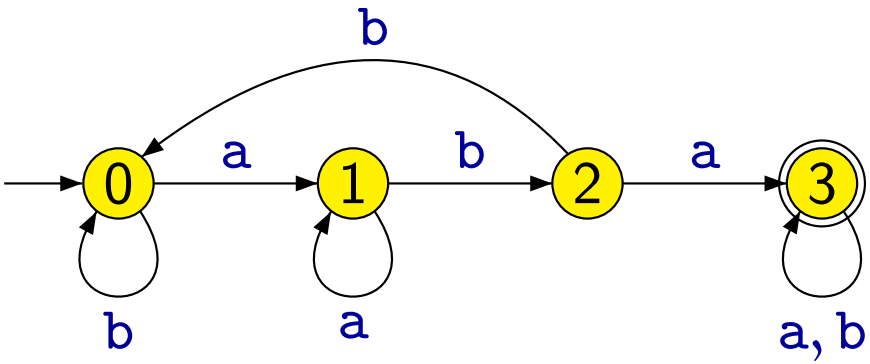
Stav q konečného automatu $A = (Q, \Sigma, \delta, q_0, F)$ je **dosažitelný** pokud existuje nějaké slovo w takové, že $q_0 \xrightarrow{w} q$.

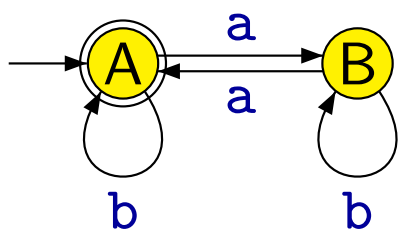
V opačném případě stav nazýváme **nedosažitelný**.

- Do nedosažitelných stavů nevede v grafu automatu žádná orientovaná cesta z počátečního stavu.
- Nedosažitelné stavy můžeme z automatu odstranit (spolu se všemi přechody vedoucími do nich a z nich). Jazyk přijímaný automatem se nezmění.

Automat pro průnik jazyků

Máme následující dva automaty:

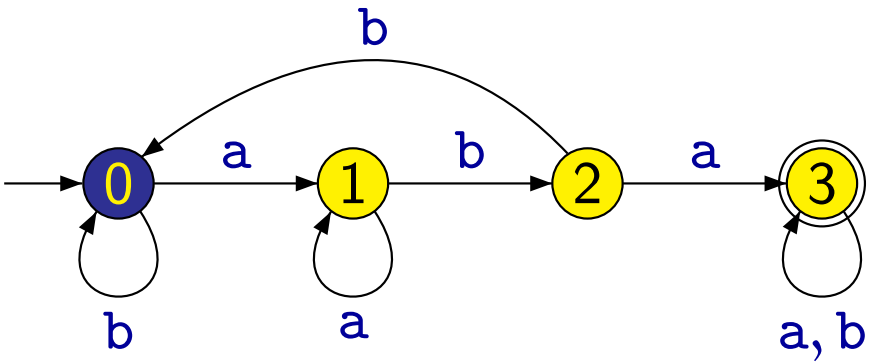


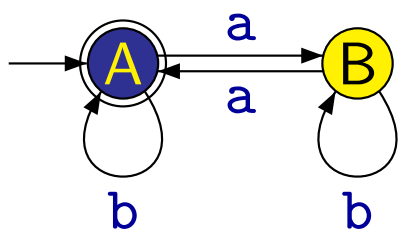


Příjmou oba slovo ababb?

Automat pro průnik jazyků

Máme následující dva automaty:

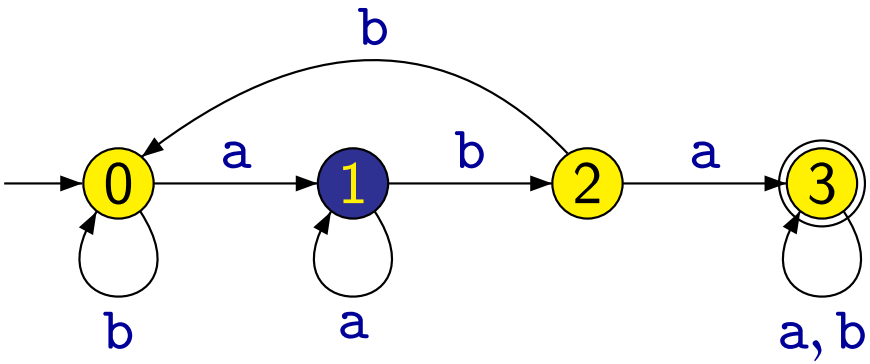


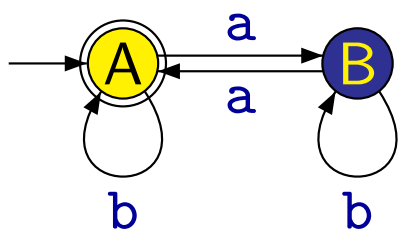


Příjmou oba slovo ababb?

Automat pro průnik jazyků

Máme následující dva automaty:

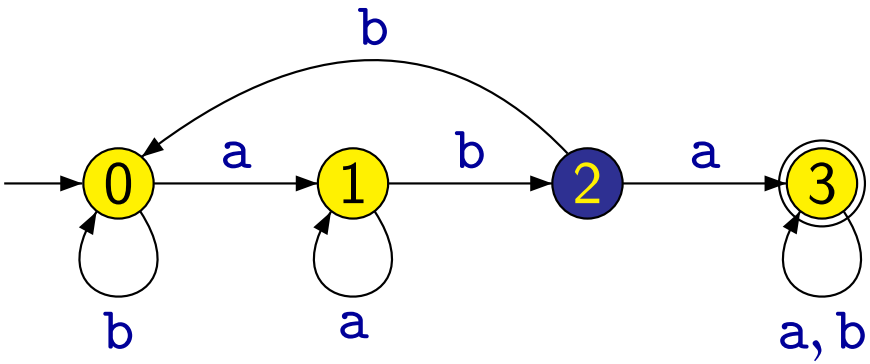


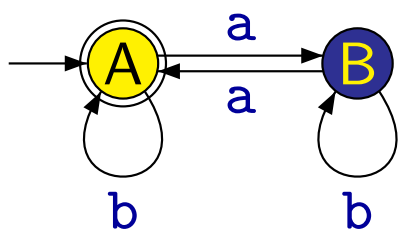


Příjmou oba slovo ababb?

Automat pro průnik jazyků

Máme následující dva automaty:

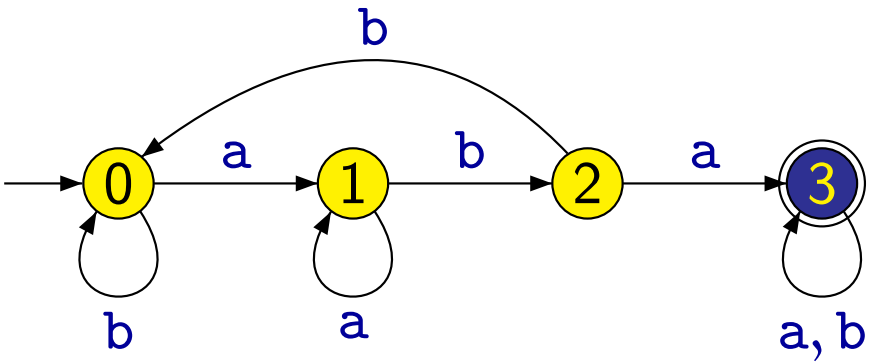


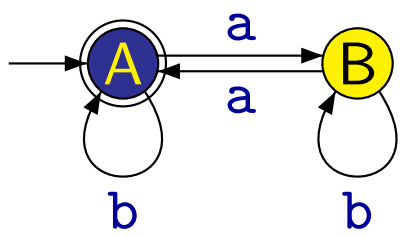


Příjmou oba slovo ababb?

Automat pro průnik jazyků

Máme následující dva automaty:

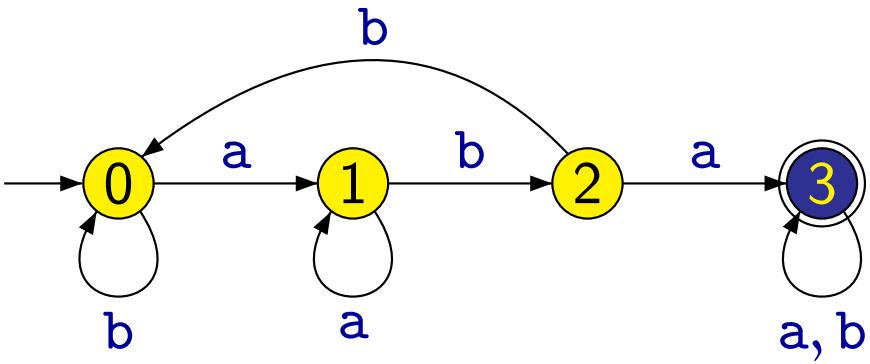


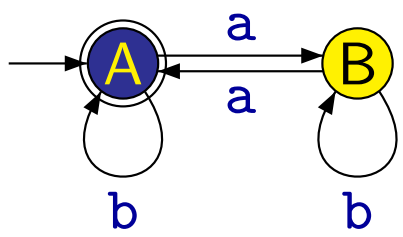


Příjmou oba slovo ababb?

Automat pro průnik jazyků

Máme následující dva automaty:

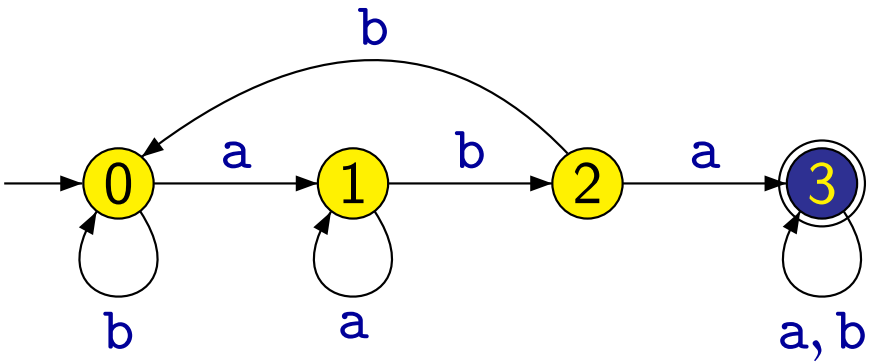


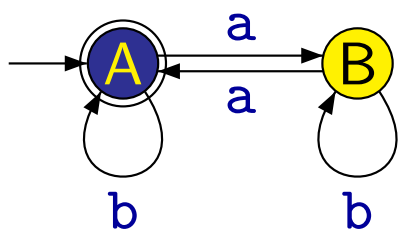


Příjmou oba slovo ababb?

Automat pro průnik jazyků

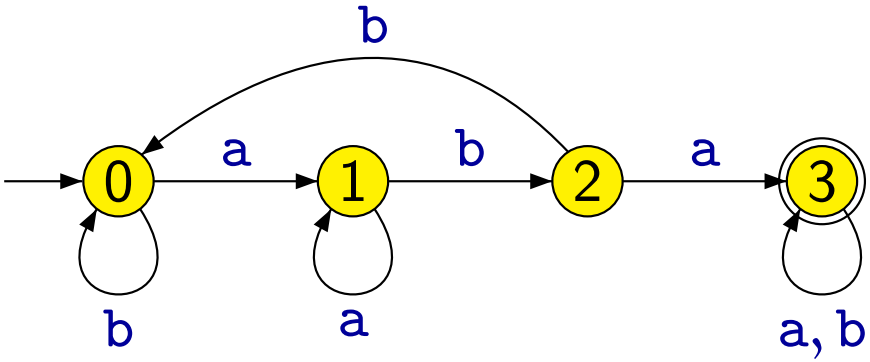
Máme následující dva automaty:

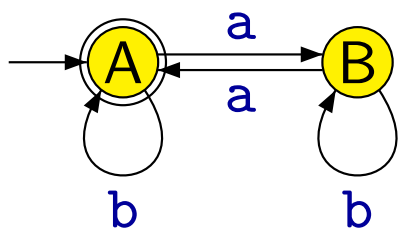




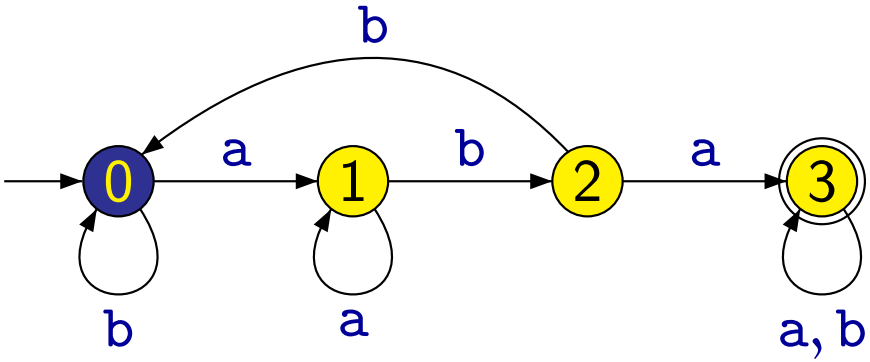
Příjmou oba slovo ababb?

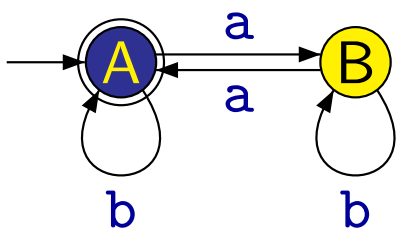
Automat pro průnik jazyků

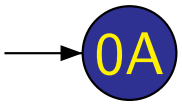




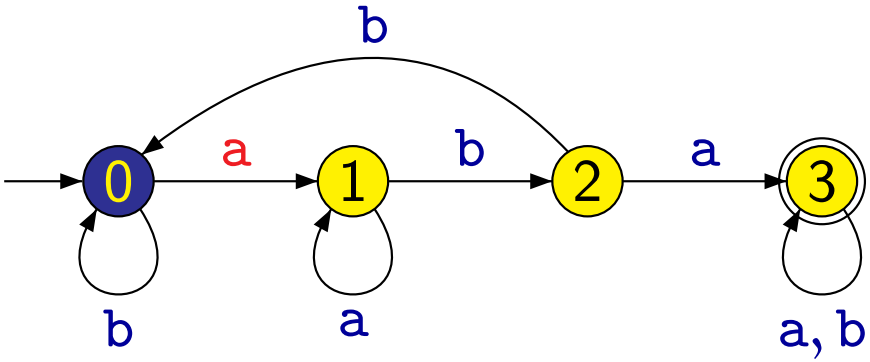
Automat pro průnik jazyků

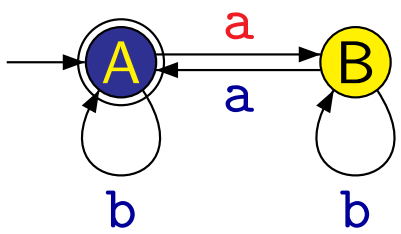




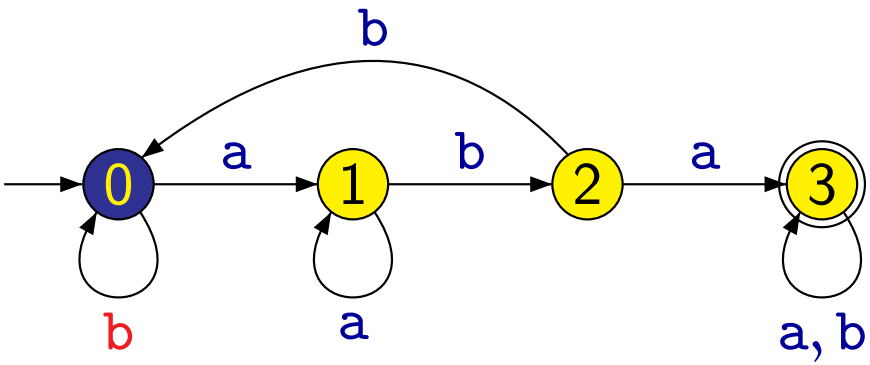


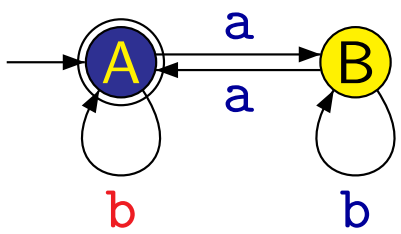
Automat pro průnik jazyků

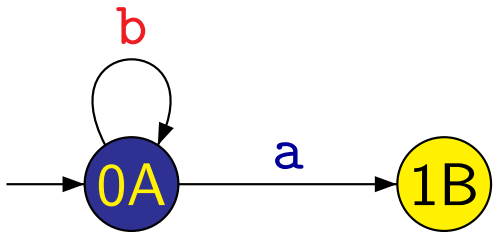




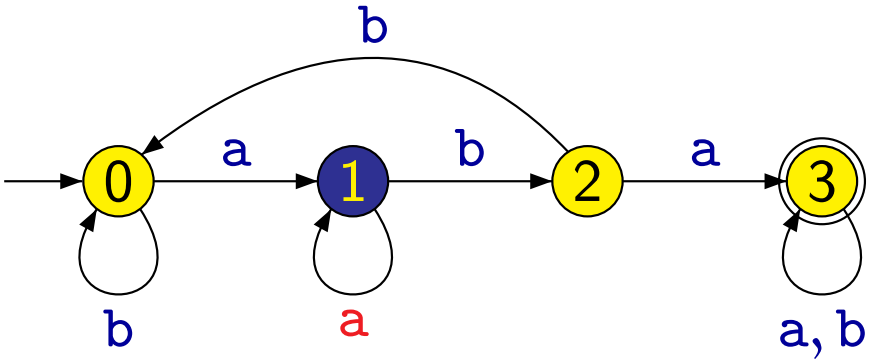
Automat pro průnik jazyků

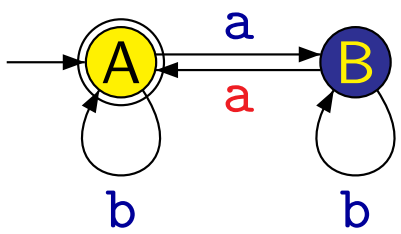


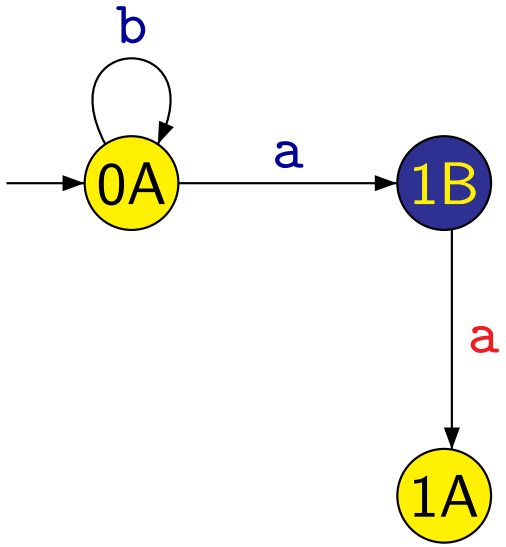




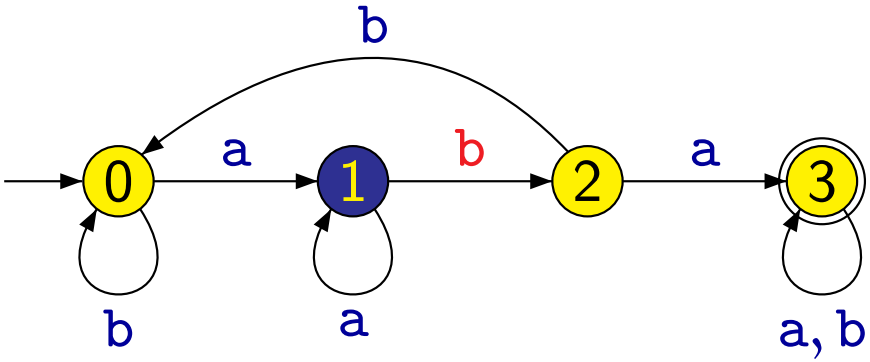
Automat pro průnik jazyků

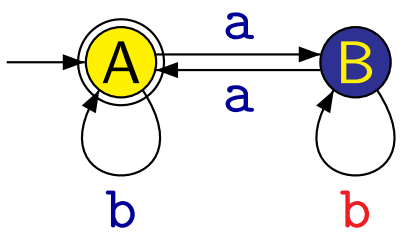


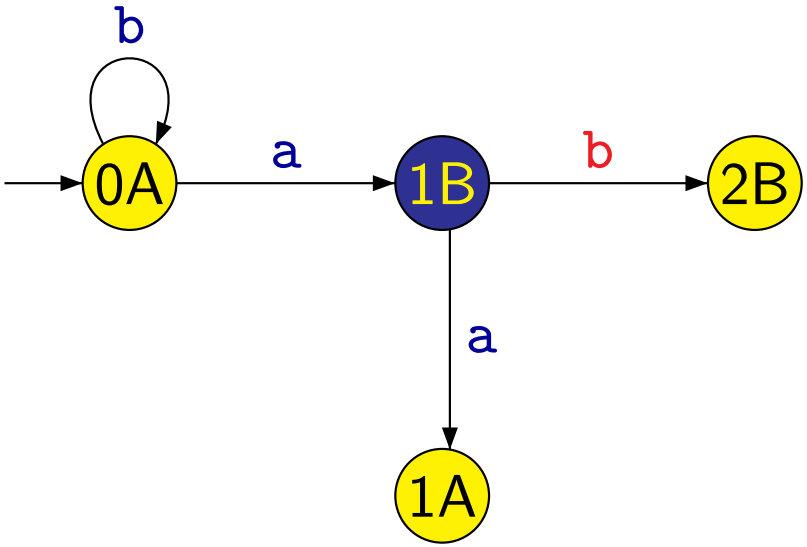




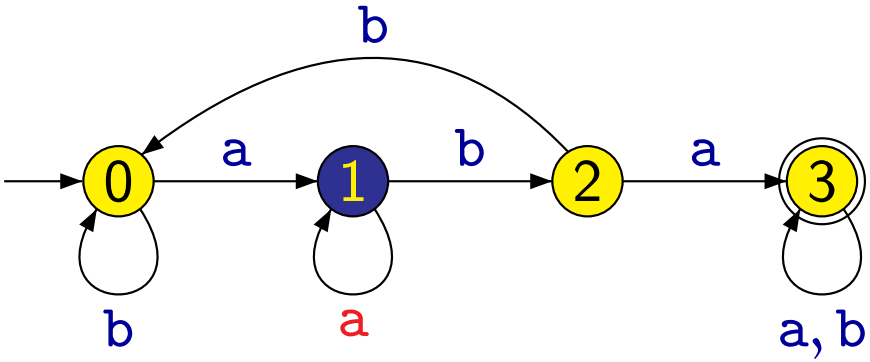
Automat pro průnik jazyků

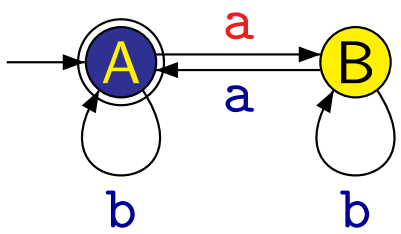


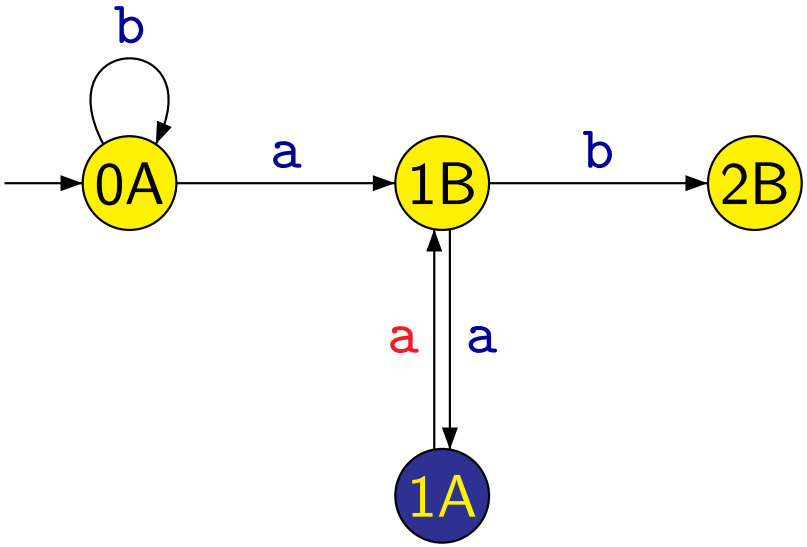




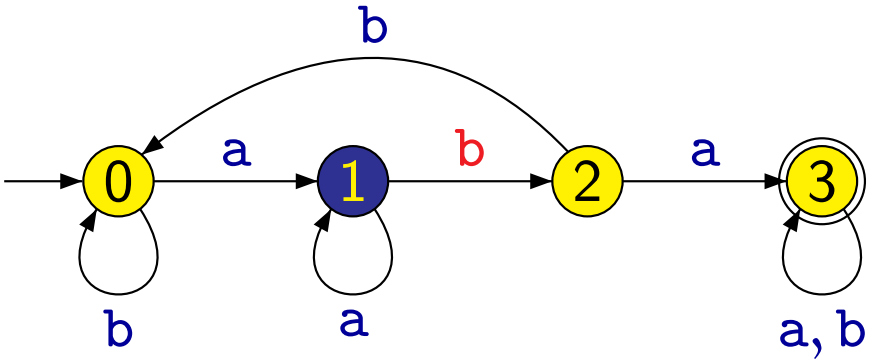
Automat pro průnik jazyků

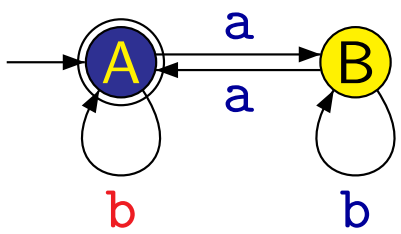


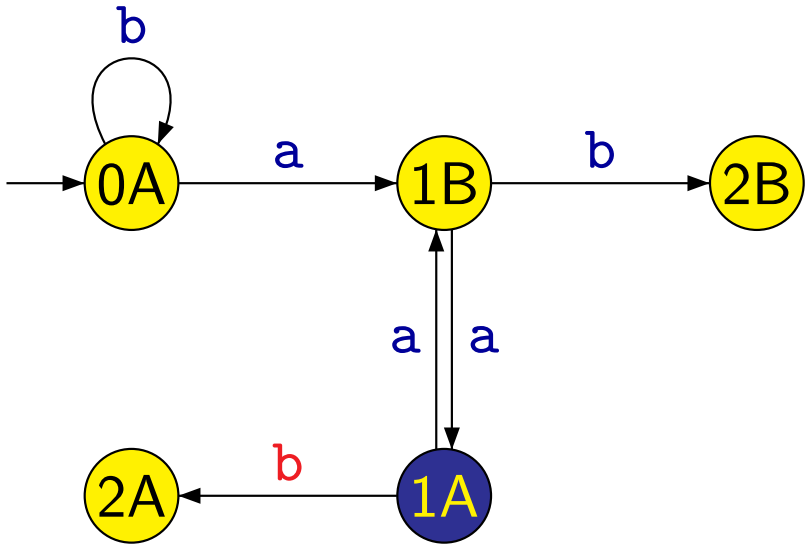




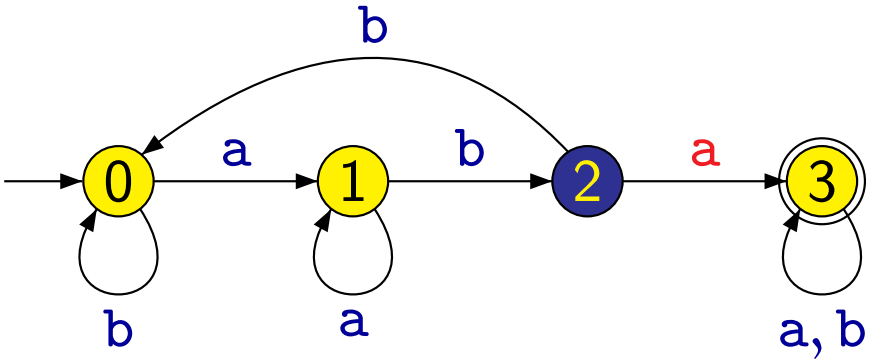
Automat pro průnik jazyků

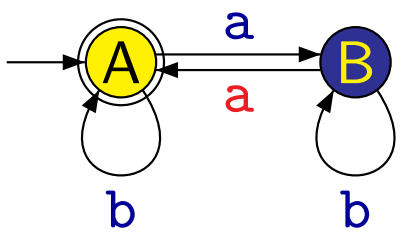


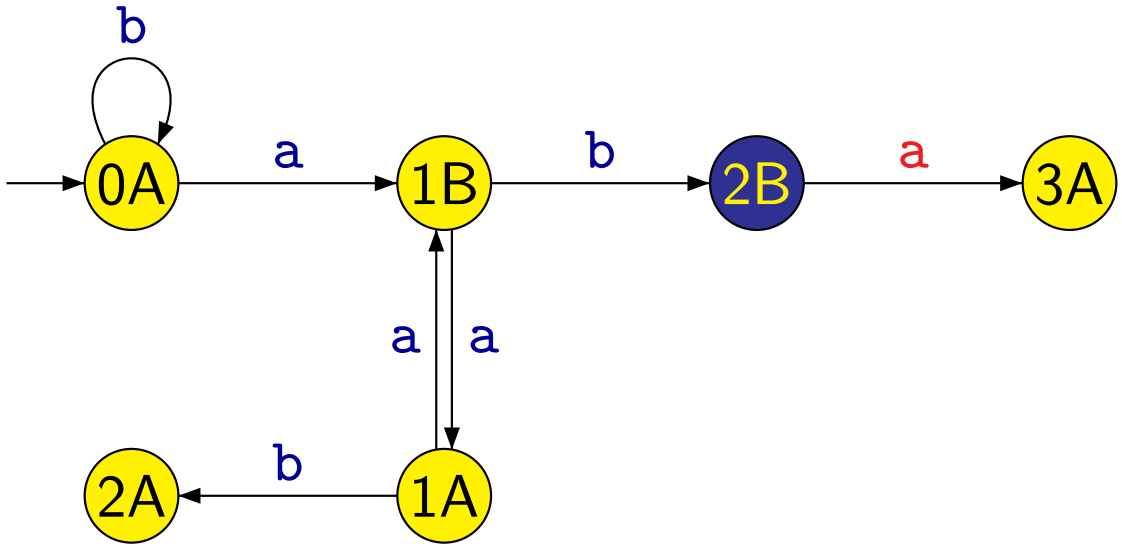




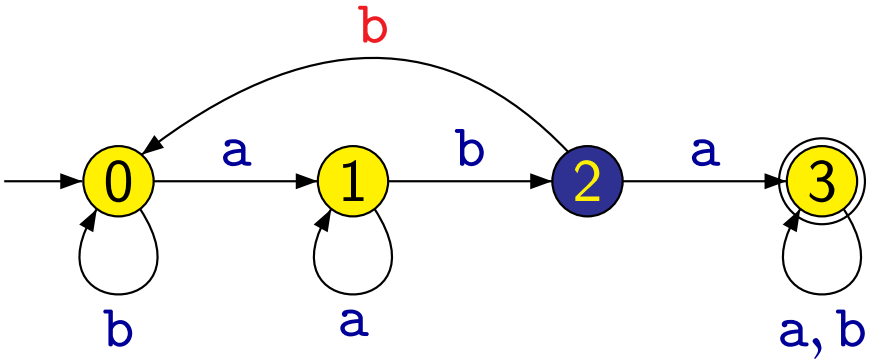
Automat pro průnik jazyků

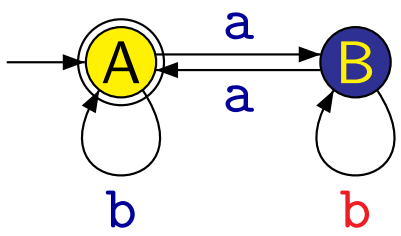


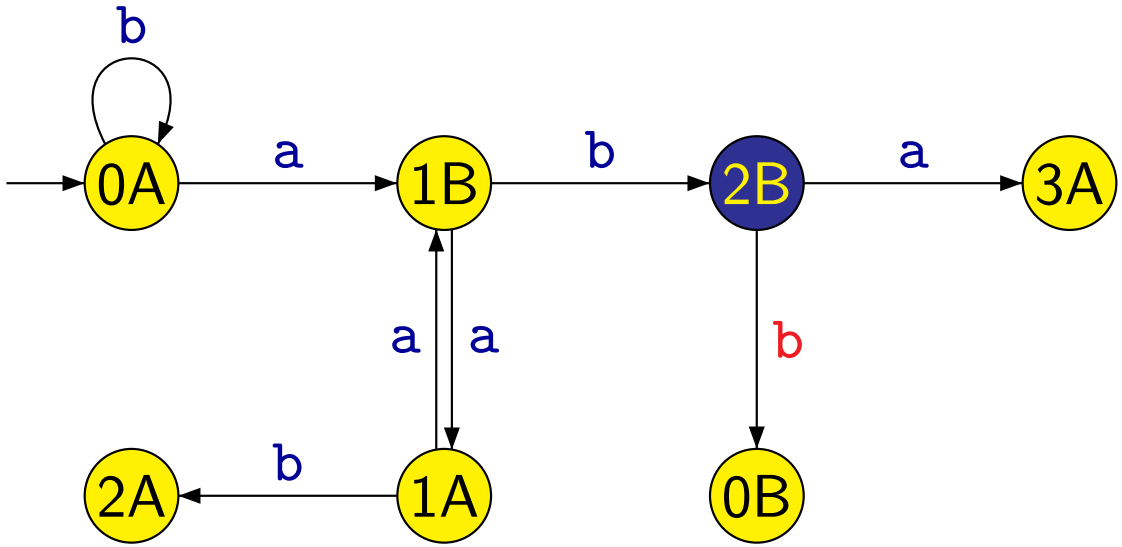




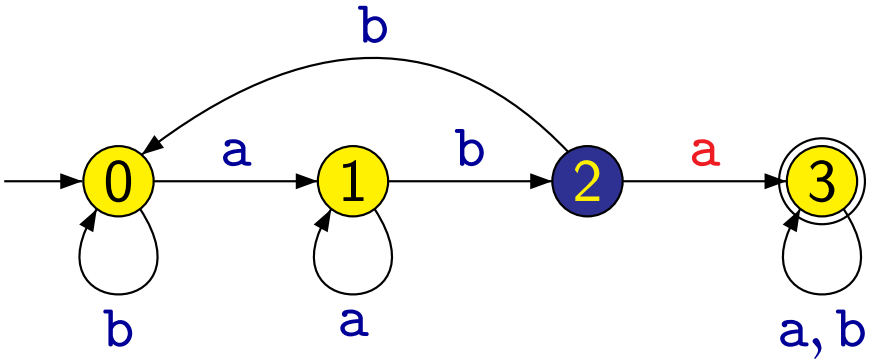
Automat pro průnik jazyků

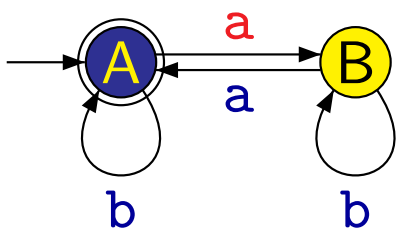


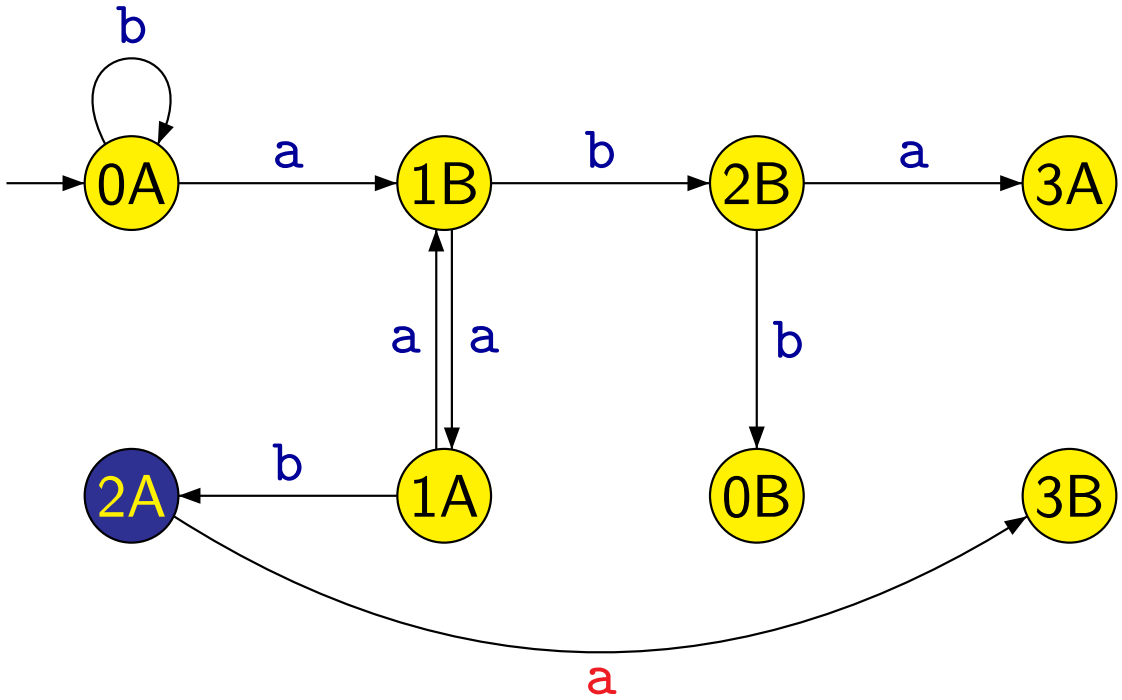




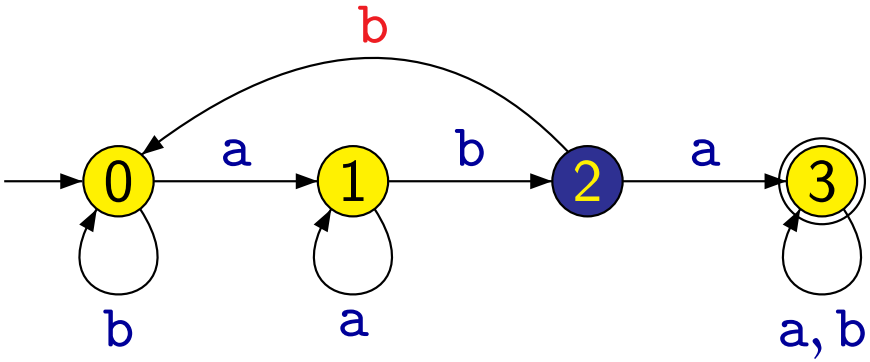
Automat pro průnik jazyků

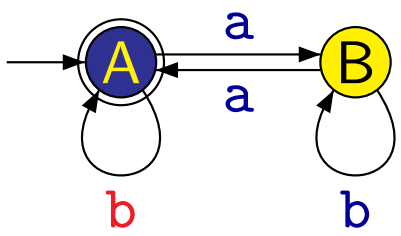


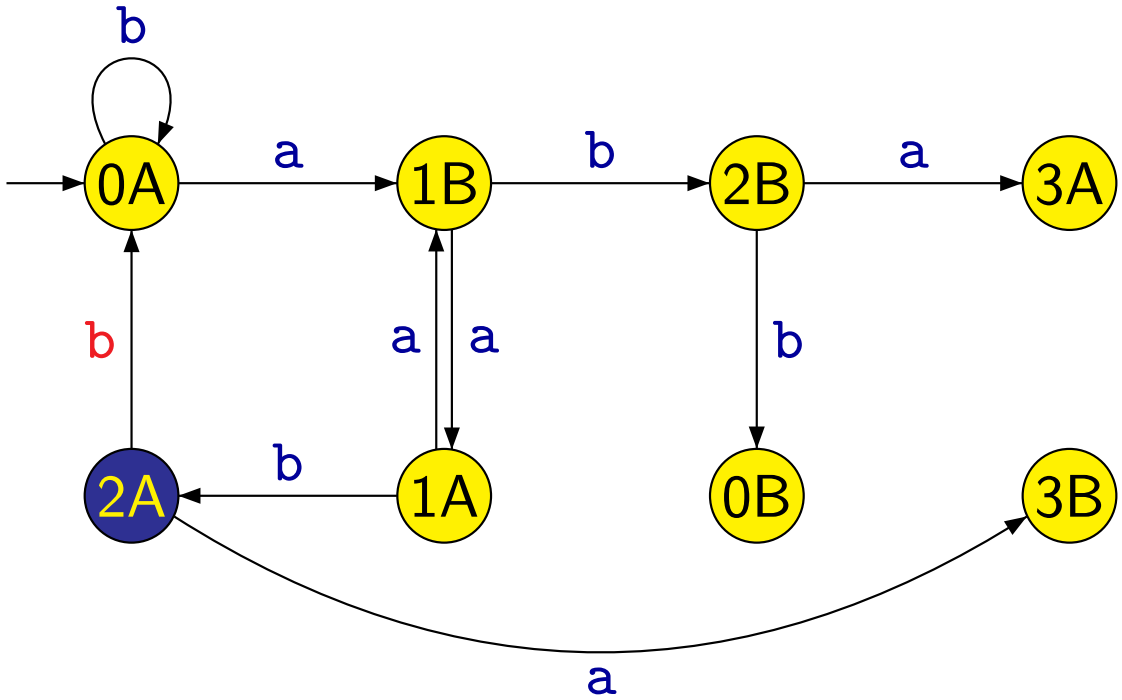




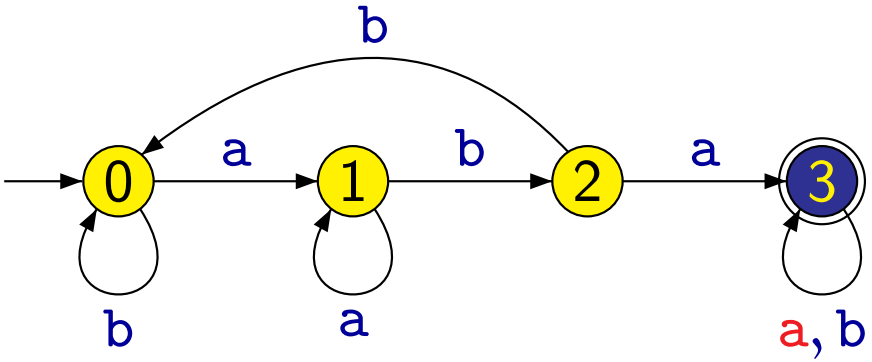
Automat pro průnik jazyků

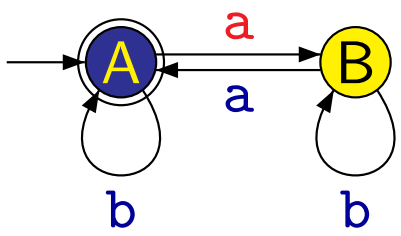


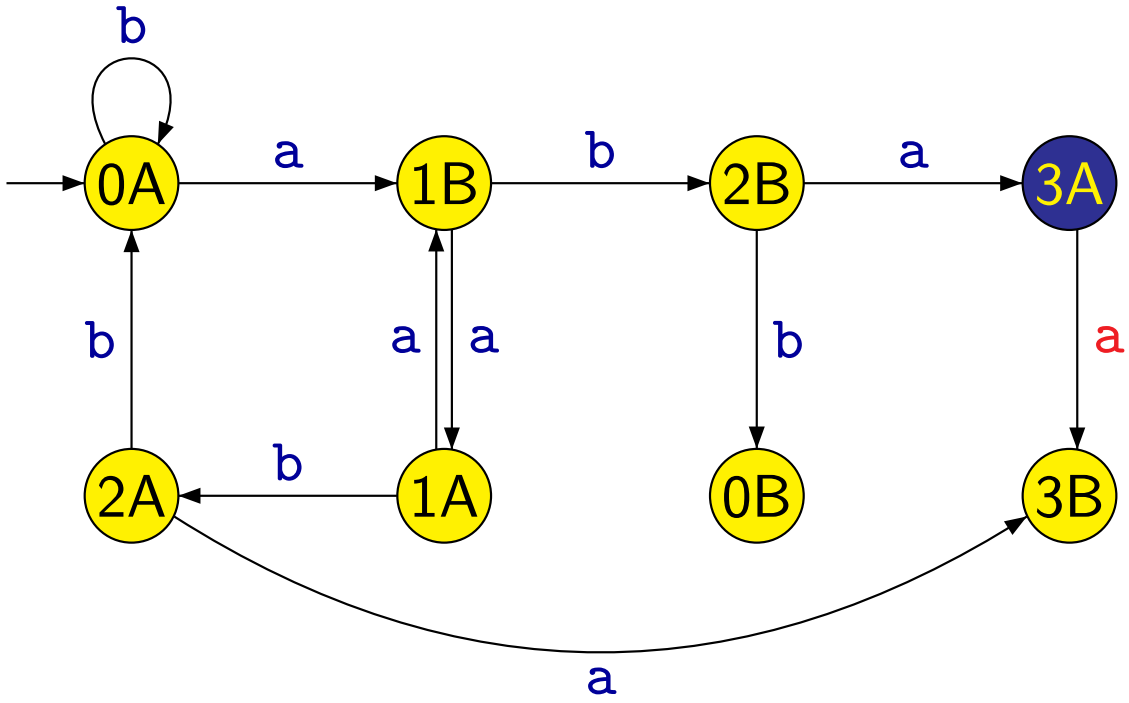




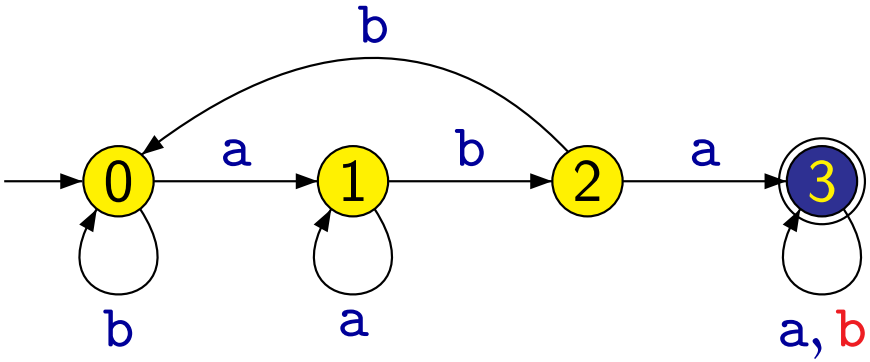
Automat pro průnik jazyků

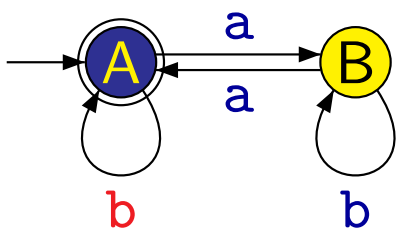


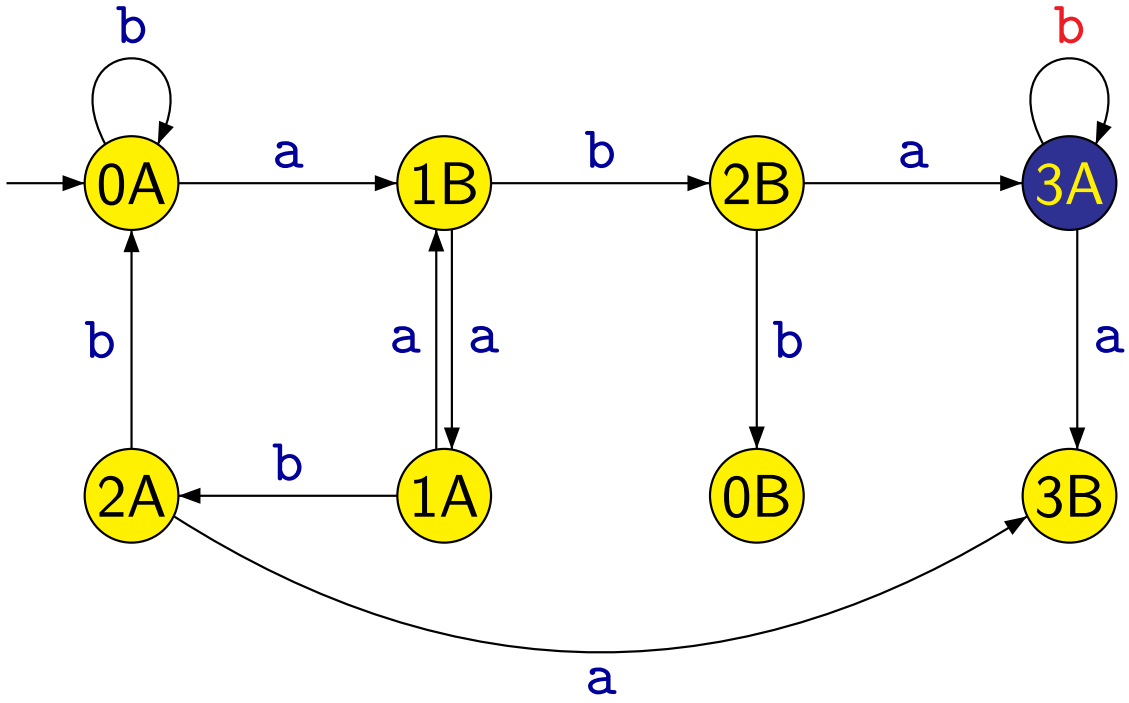




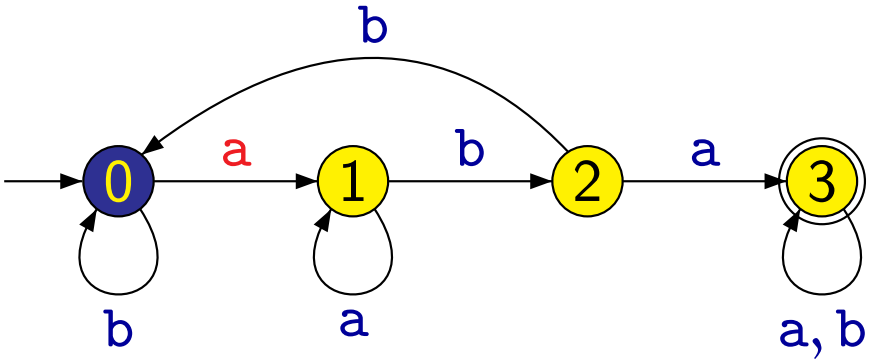
Automat pro průnik jazyků

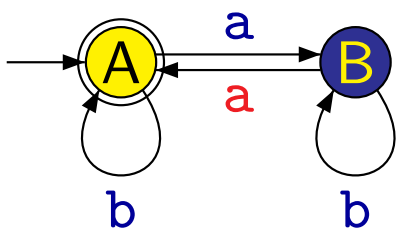


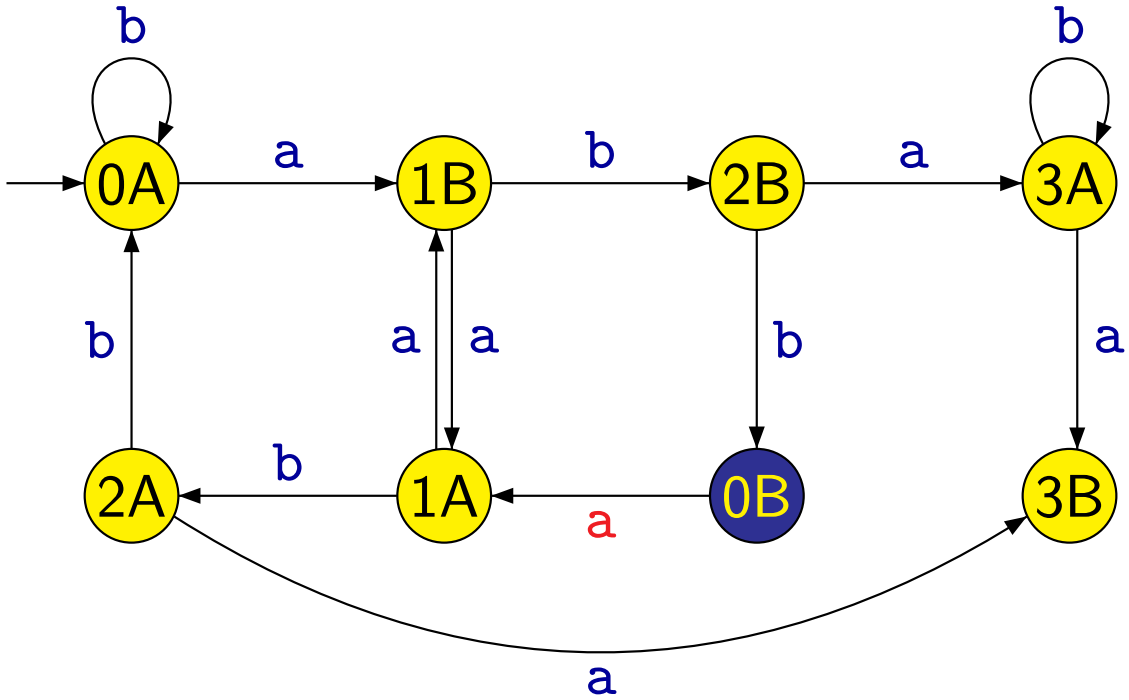




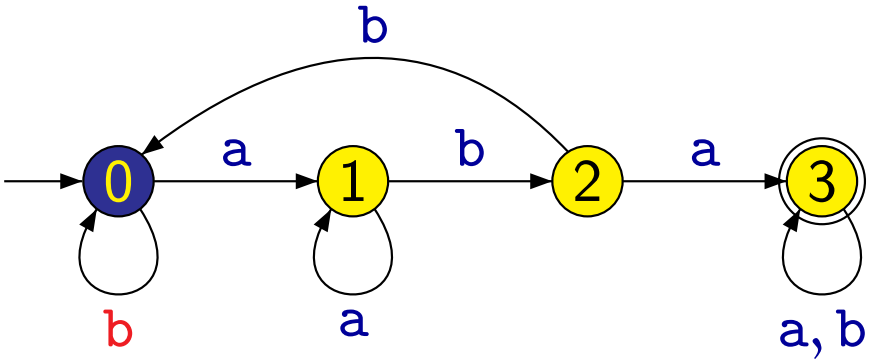
Automat pro průnik jazyků

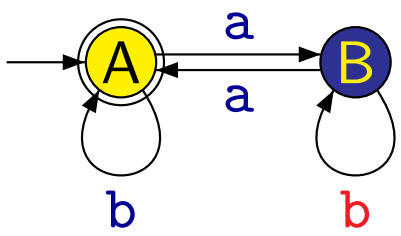


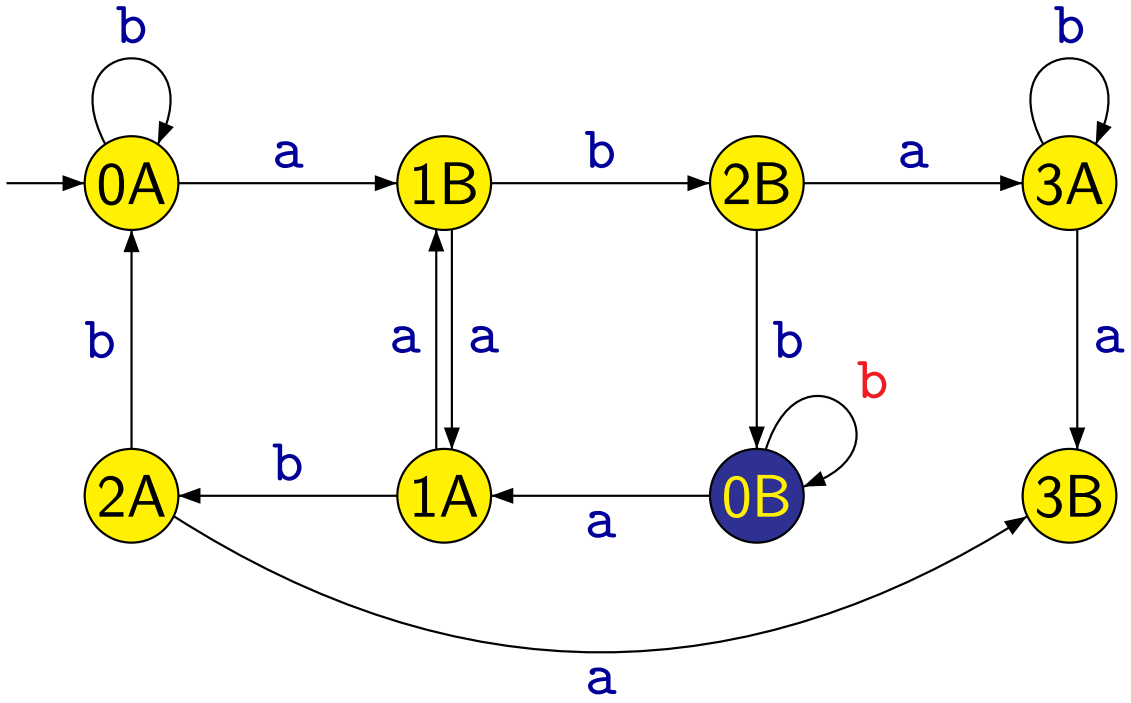




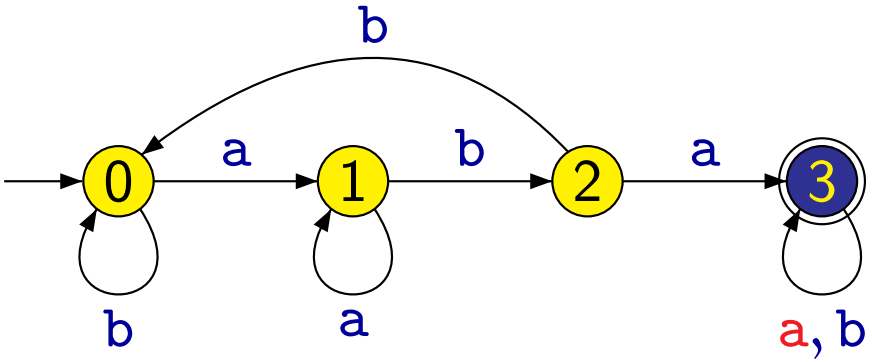
Automat pro průnik jazyků

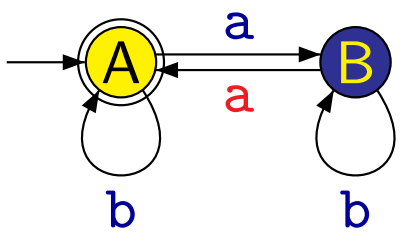


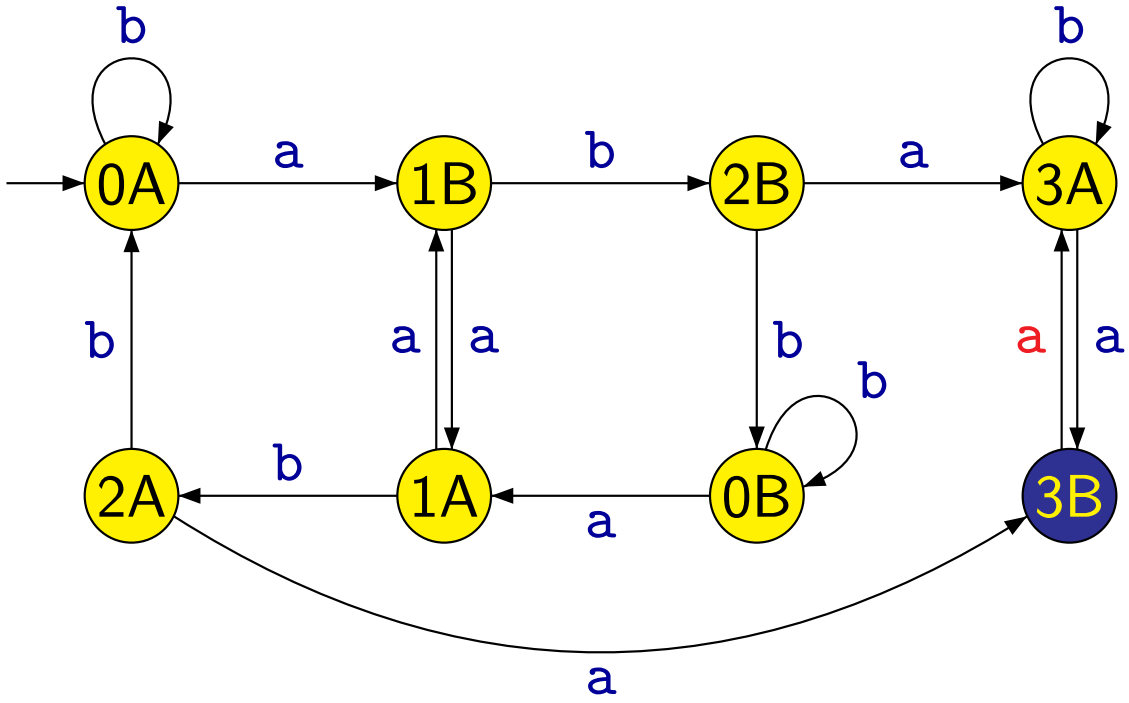




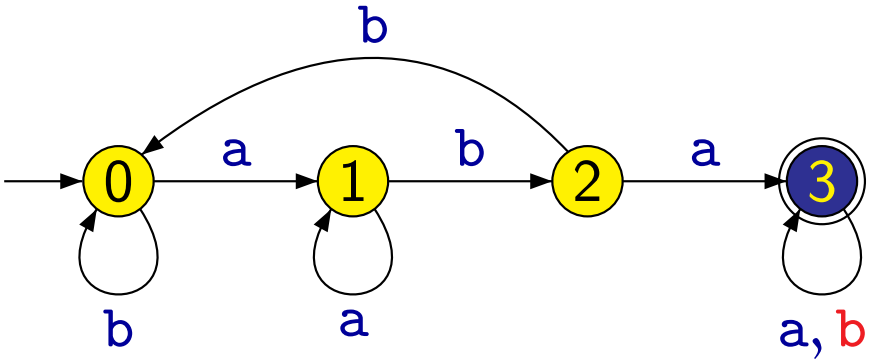
Automat pro průnik jazyků

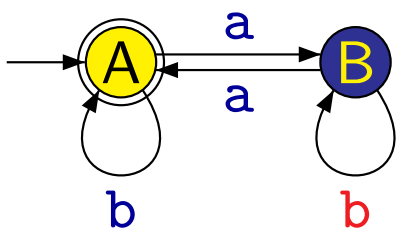


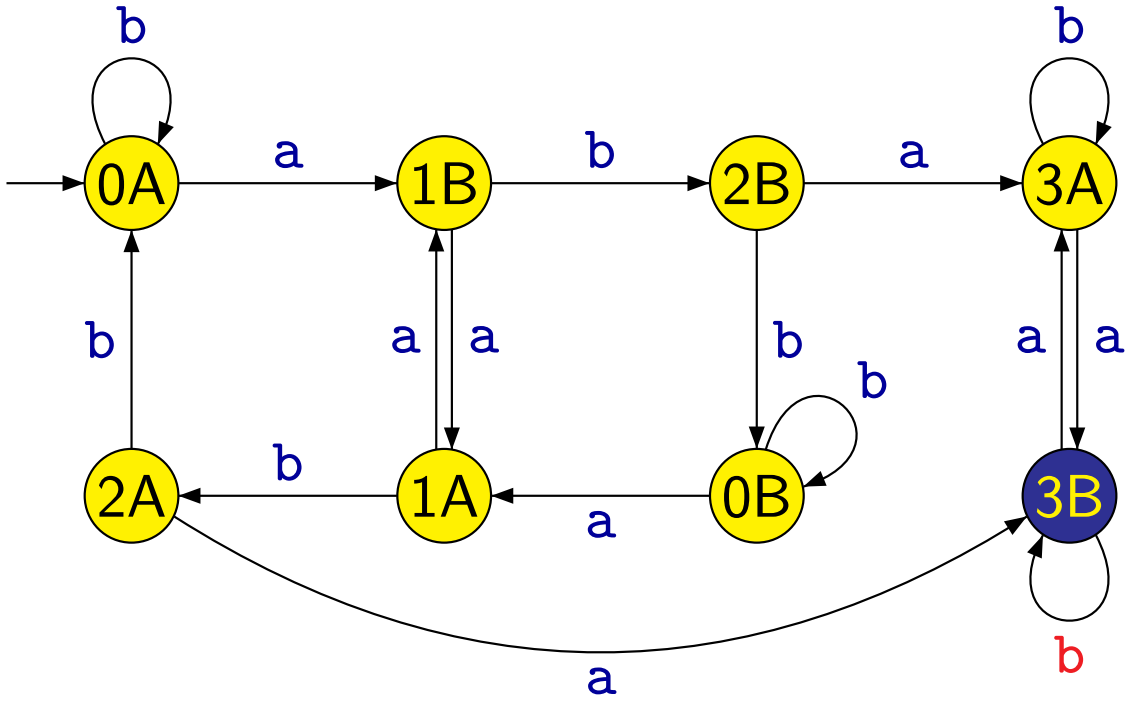




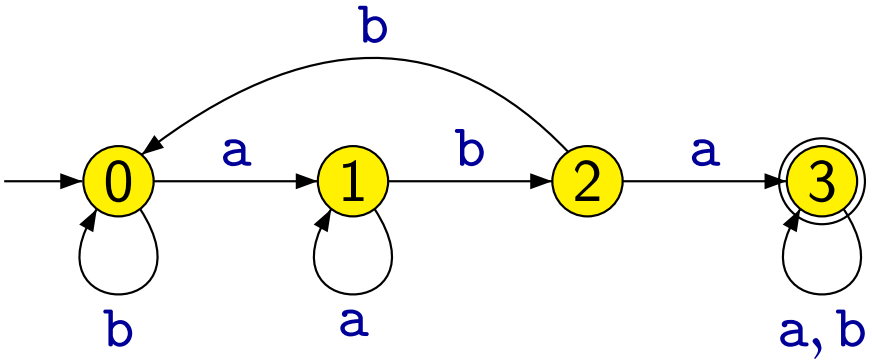
Automat pro průnik jazyků

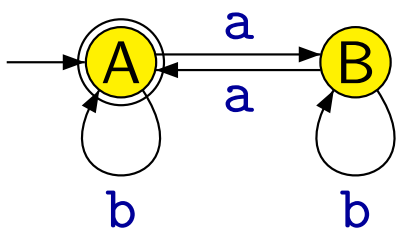


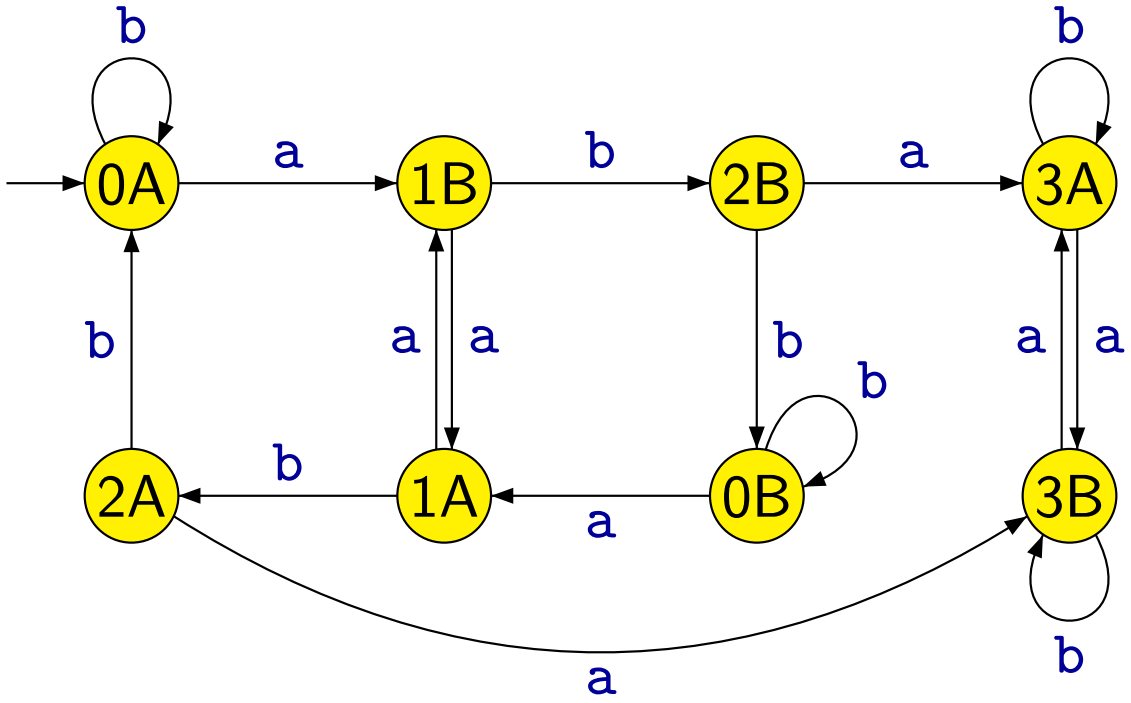




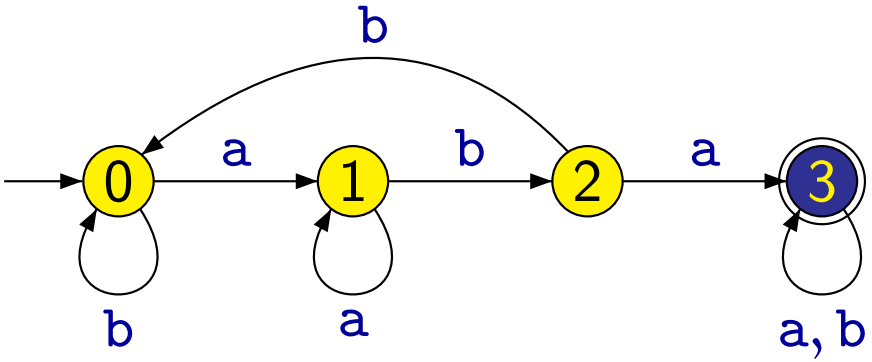
Automat pro průnik jazyků

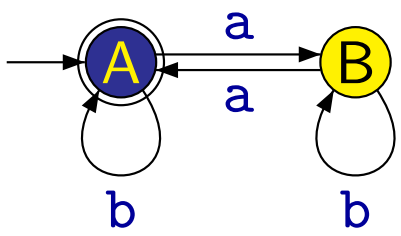


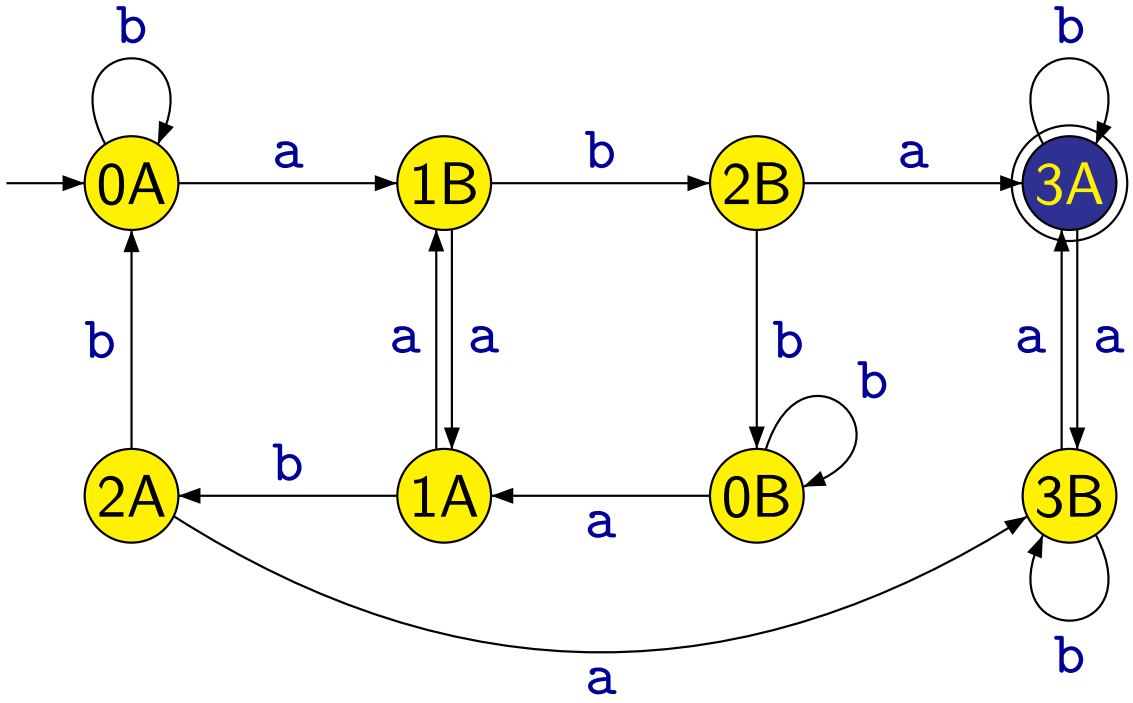




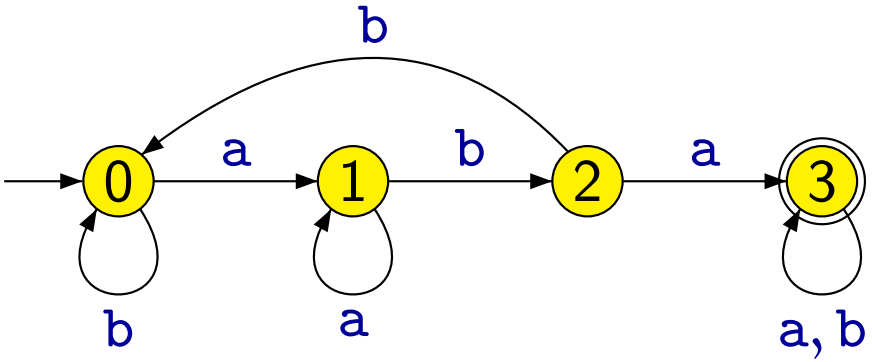
Automat pro průnik jazyků

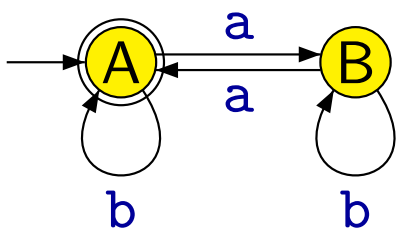


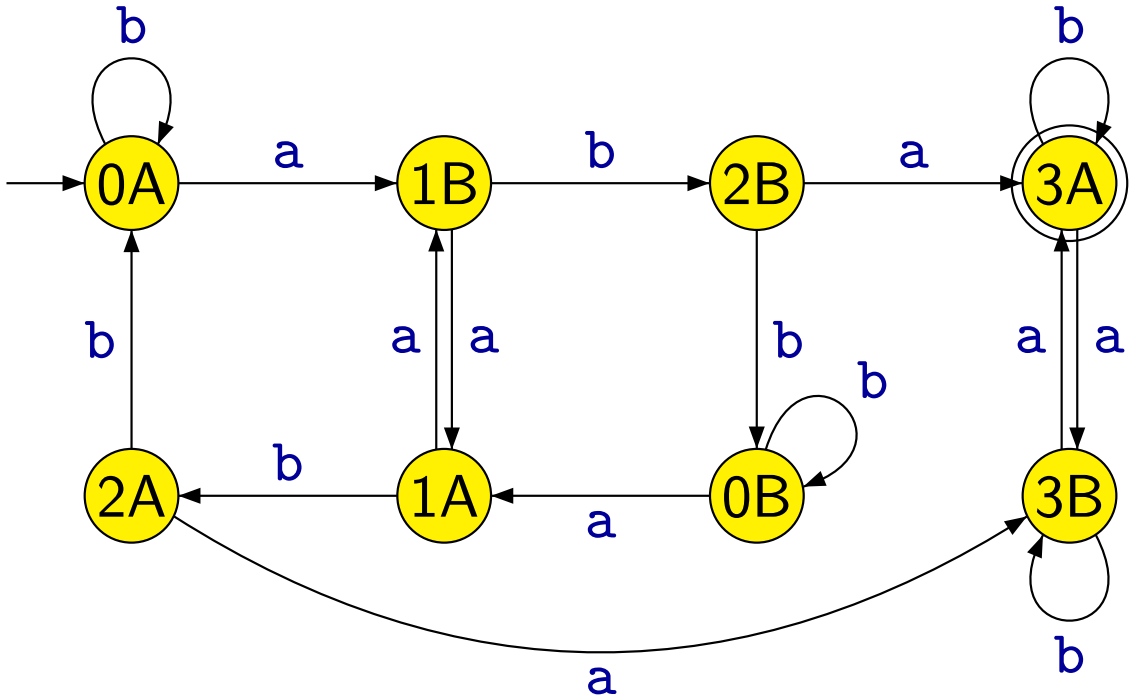




Automat pro průnik jazyků







Automat pro průnik jazyků

Formálně můžeme popsat tuto konstrukci následovně:

Předpokládáme, že máme dva deterministické konečné automaty $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ a $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$.

K nim setrojíme DKA $A = (Q, \Sigma, \delta, q_0, F)$ kde:

- $Q = Q_1 \times Q_2$
- $\delta(\,(q_1, q_2), a\,) = (\,\delta_1(q_1, a), \delta_2(q_2, a)\,)$ pro všechna $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$
- $q_0 = (q_{01}, q_{02})$
- $F = F_1 \times F_2$

Není težké ověřit, že pro libovolné slovo $w \in \Sigma^*$ platí, že $w \in L(A)$ právě tehdy, když $w \in L(A_1)$ a $w \in L(A_2)$, tj.

$$L(A) = L(A_1) \cap L(A_2)$$

Průnik regulárních jazyků

Věta

Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cap L_2$ je regulární.

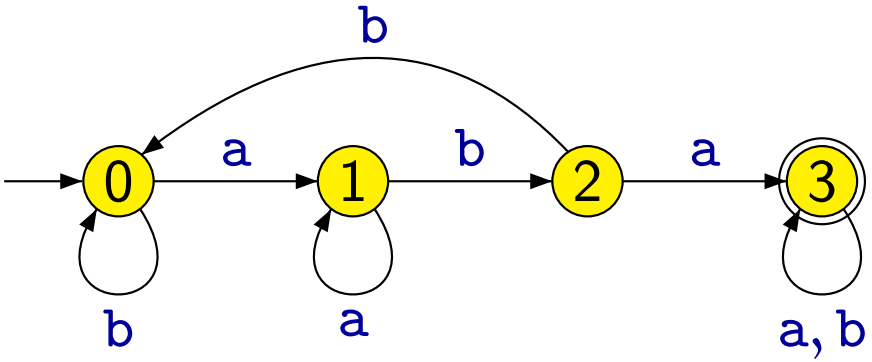
Důkaz: Předpokládejme, že A_1 a A_2 jsou deterministické konečné automaty takové, že

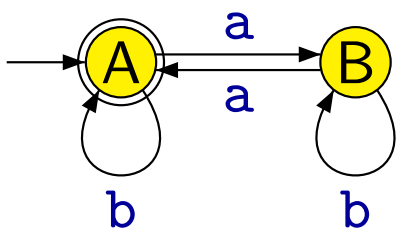
$$L_1 = L(A_1) \qquad L_2 = L(A_2)$$

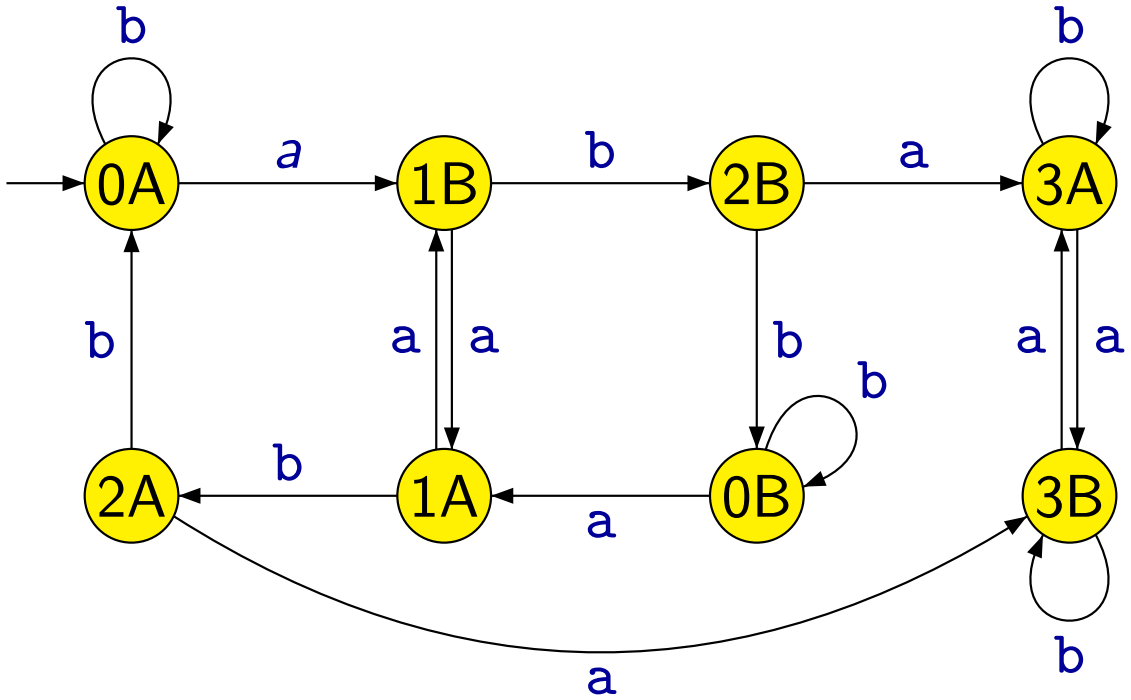
Popsanou konstrukcí k nim můžeme sestrojit deterministický konečný automat A takový, že

$$L(A) = L(A_1) \cap L(A_2) = L_1 \cap L_2$$

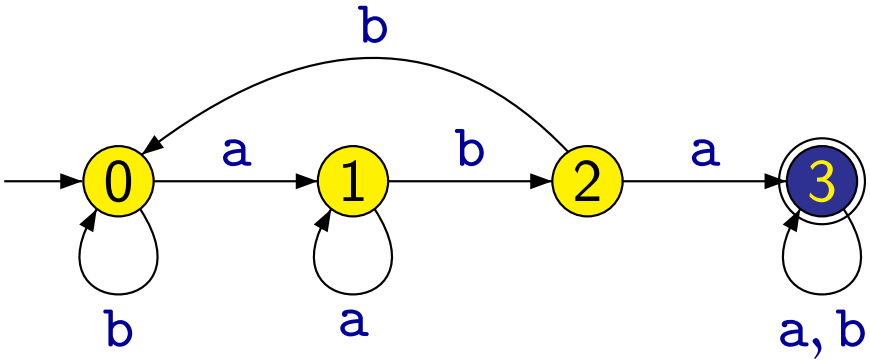
Automat pro sjednocení jazyků

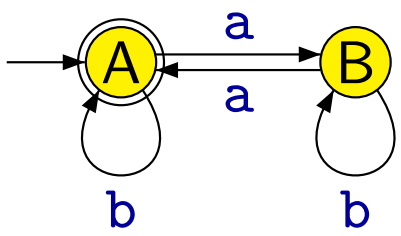


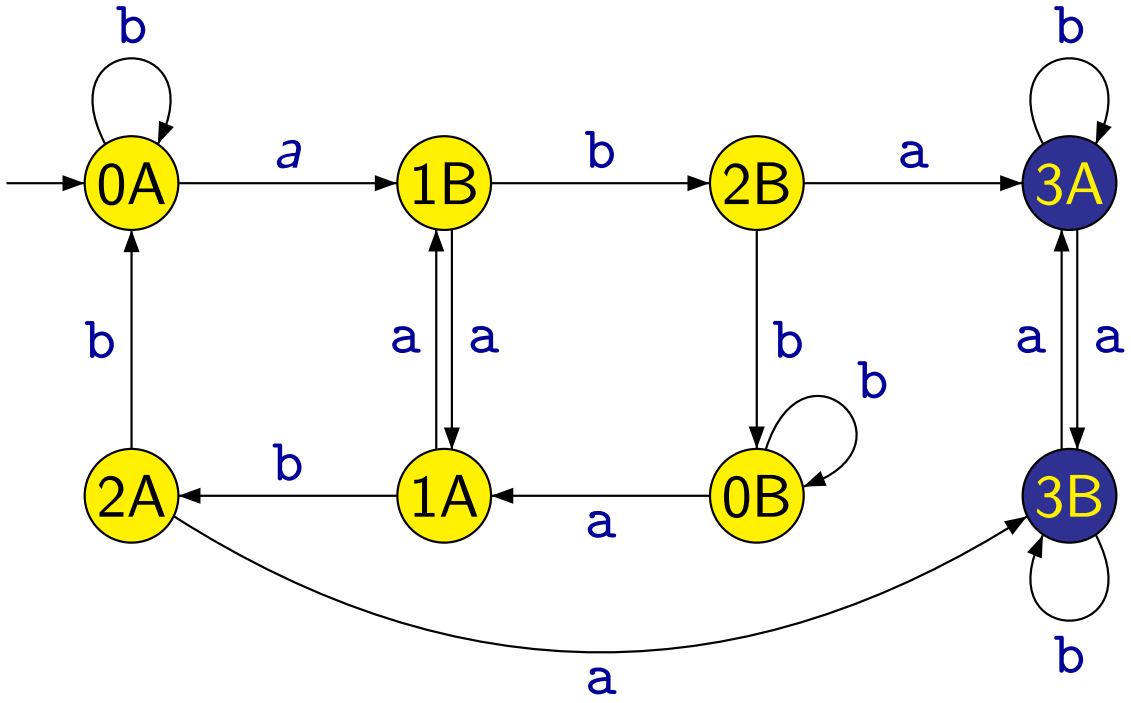




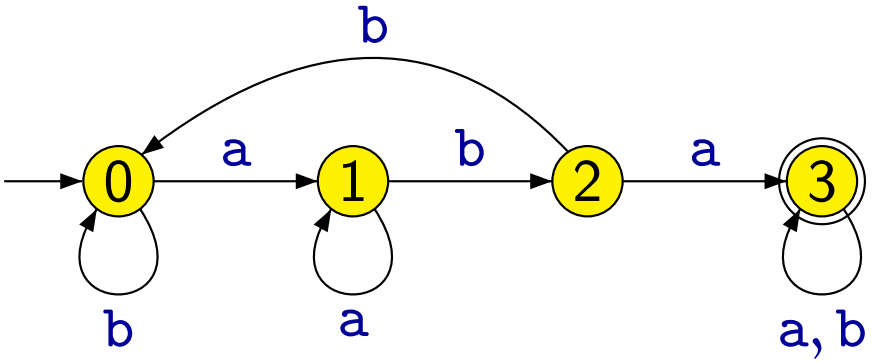
Automat pro sjednocení jazyků

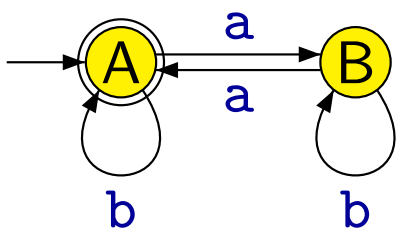


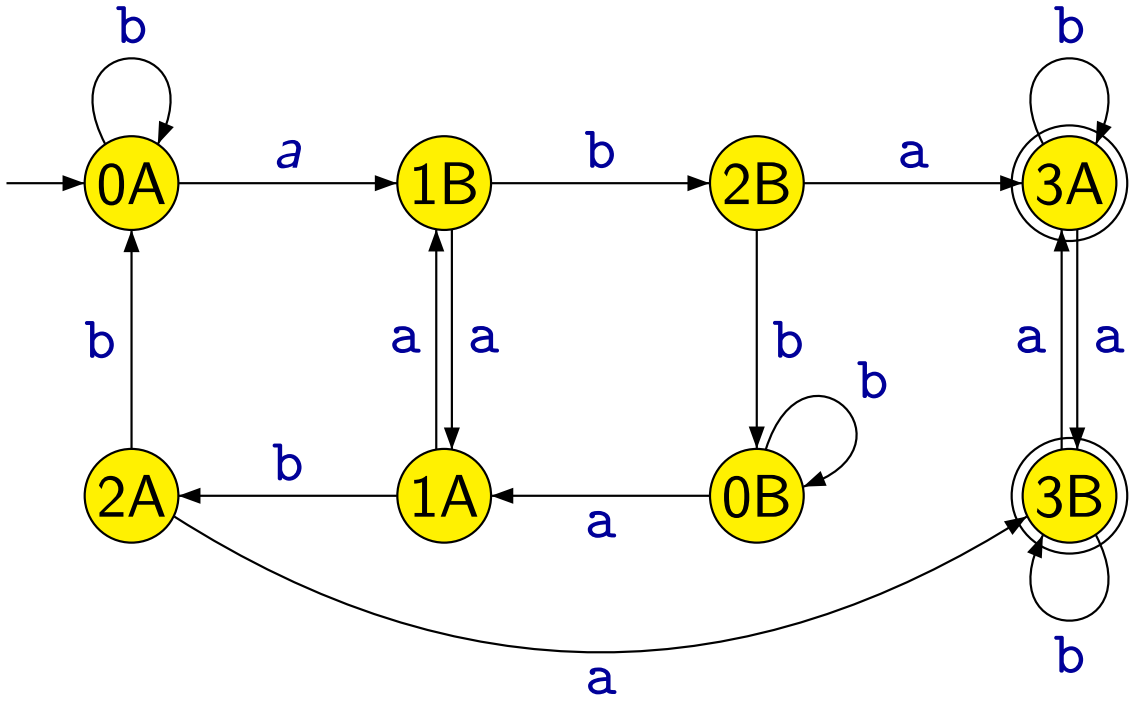




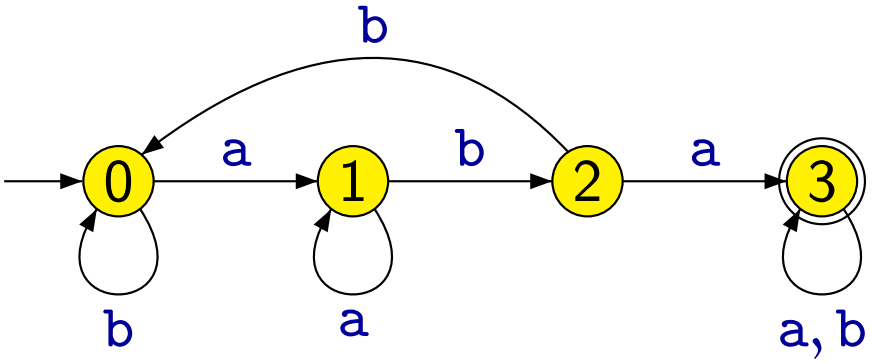
Automat pro sjednocení jazyků

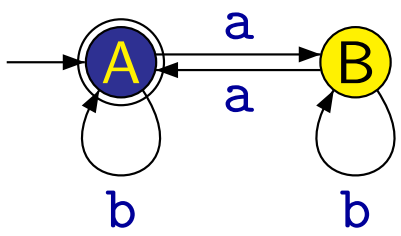


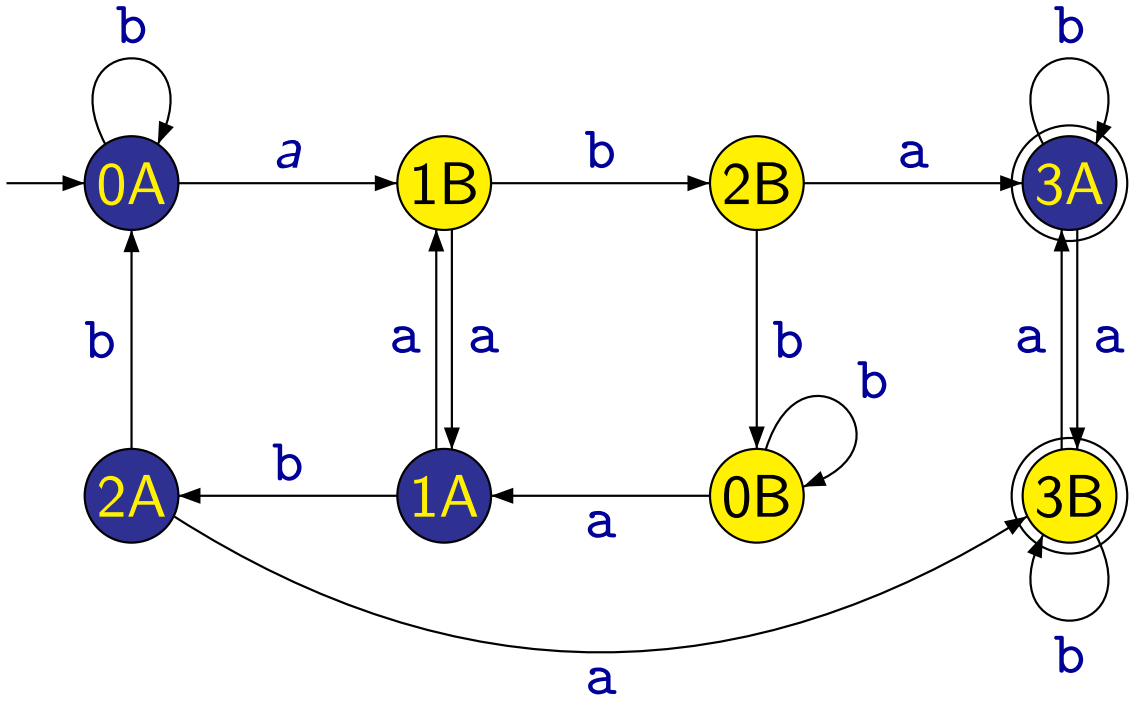




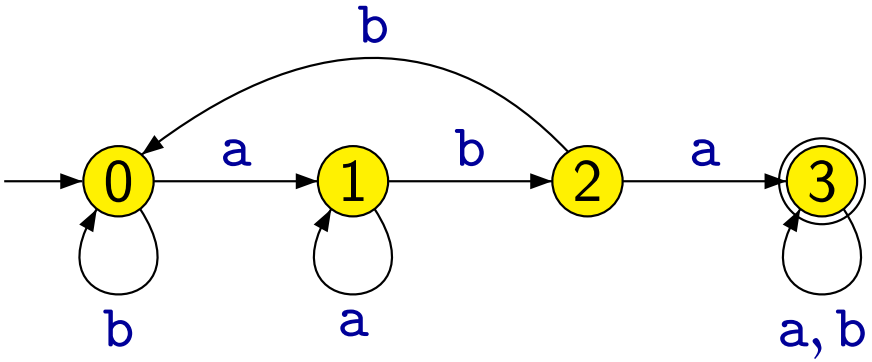
Automat pro sjednocení jazyků

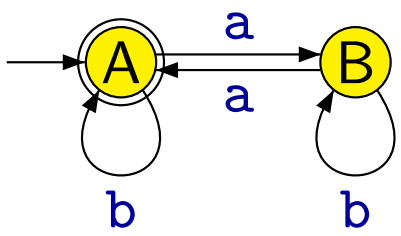


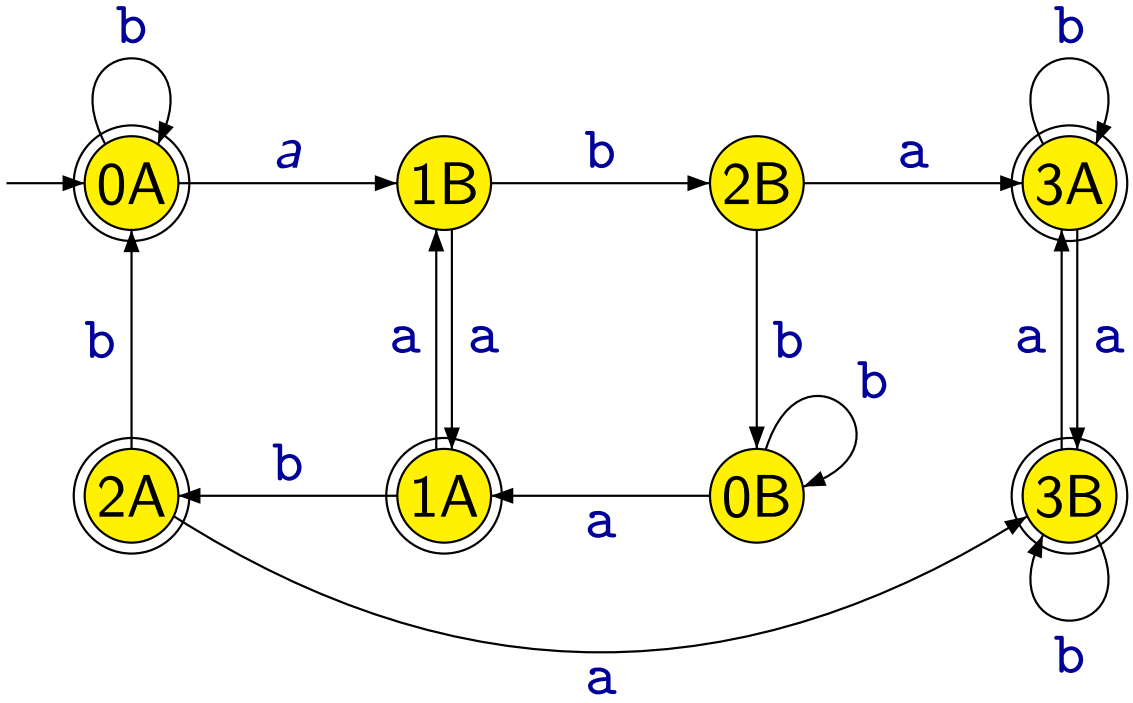




Automat pro sjednocení jazyků







Sjednocení regulárních jazyků

Konstukce automatu A, který přijímá **sjednocení** jazyků přijímaných automaty A_1 a A_2, tj. jazyk

$$L(A_1) \cup L(A_1)$$

je téměř stejná jako v případě automatu přijímajícího $L(A_1) \cap L(A_2)$.

Jediný rozdíl je v definici množiny přijímajících stavů:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Sjednocení regulárních jazyků

Konstukce automatu A, který přijímá **sjednocení** jazyků přijímaných automaty A_1 a A_2, tj. jazyk

$$L(A_1) \cup L(A_1)$$

je téměř stejná jako v případě automatu přijímajícího $L(A_1) \cap L(A_2)$.

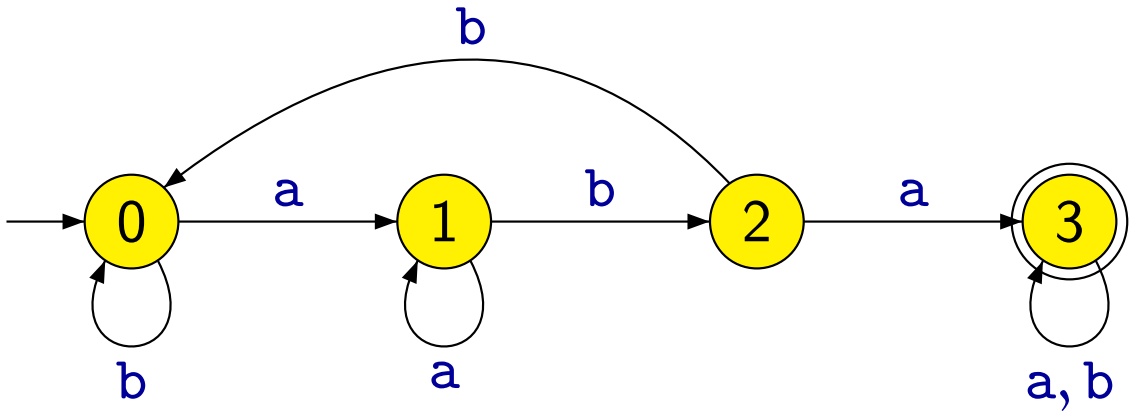
Jediný rozdíl je v definici množiny přijímajících stavů:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

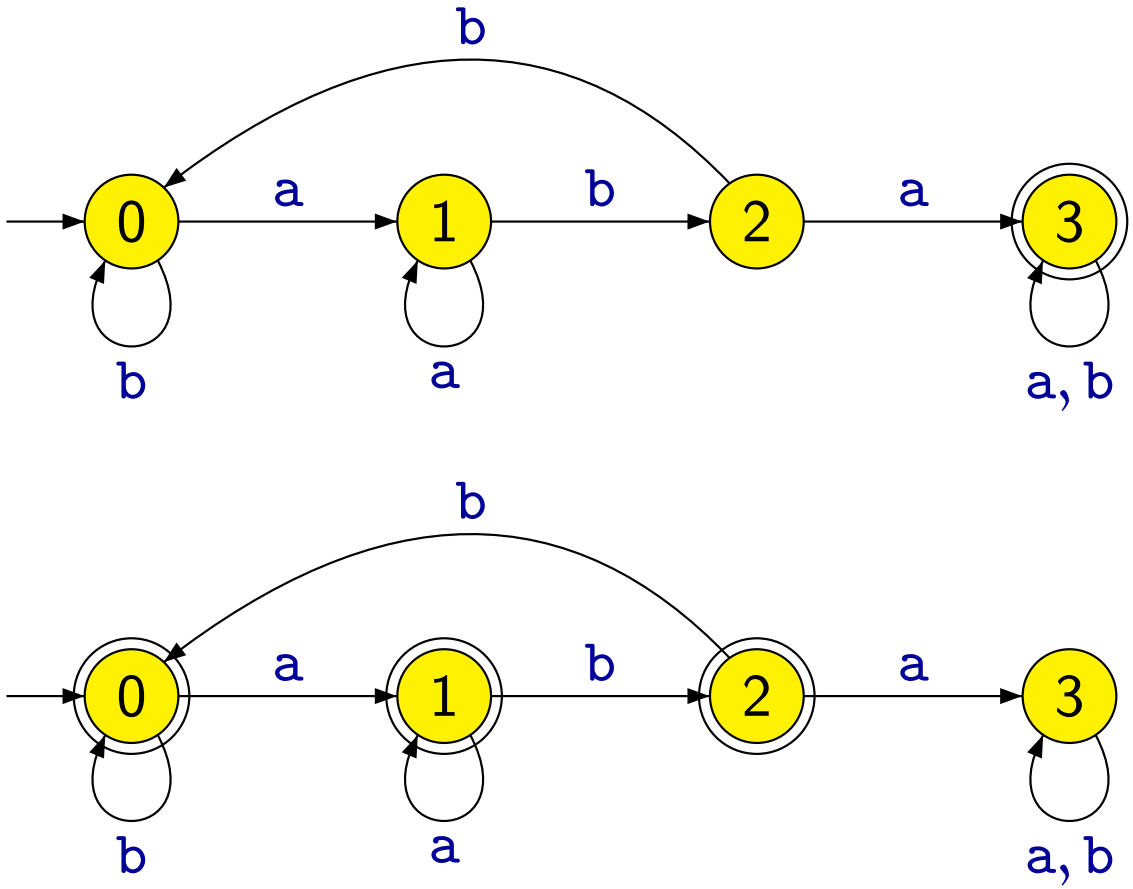
Věta

Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cup L_2$ je regulární.

Automat pro doplněk jazyka



Automat pro doplněk jazyka



Doplněk regulárního jazyka

K DKA $A = (Q, \Sigma, \delta, q_0, F)$ sestrojíme DKA $A' = (Q, \Sigma, \delta, q_0, Q - F)$.

Je očividné, že pro každé slovo $w \in \Sigma^*$ platí, že $w \in L(A')$ právě tehdy, když $w \notin L(A)$, tj.

$$L(A') = \overline{L(A)}$$

Doplněk regulárního jazyka

K DKA $A = (Q, \Sigma, \delta, q_0, F)$ sestrojíme DKA $A' = (Q, \Sigma, \delta, q_0, Q - F)$.

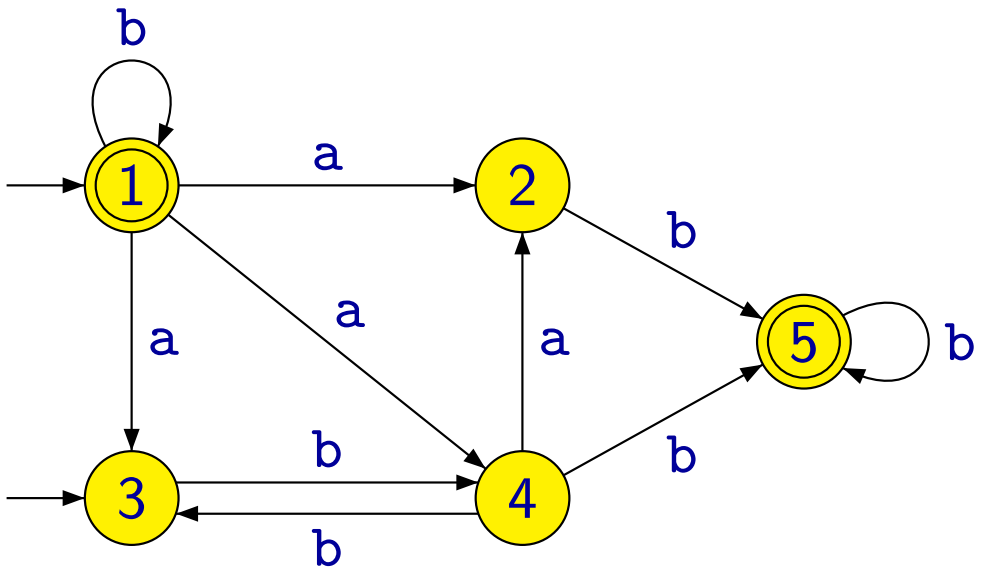
Je očividné, že pro každé slovo $w \in \Sigma^*$ platí, že $w \in L(A')$ právě tehdy, když $w \notin L(A)$, tj.

$$L(A') = \overline{L(A)}$$

Věta

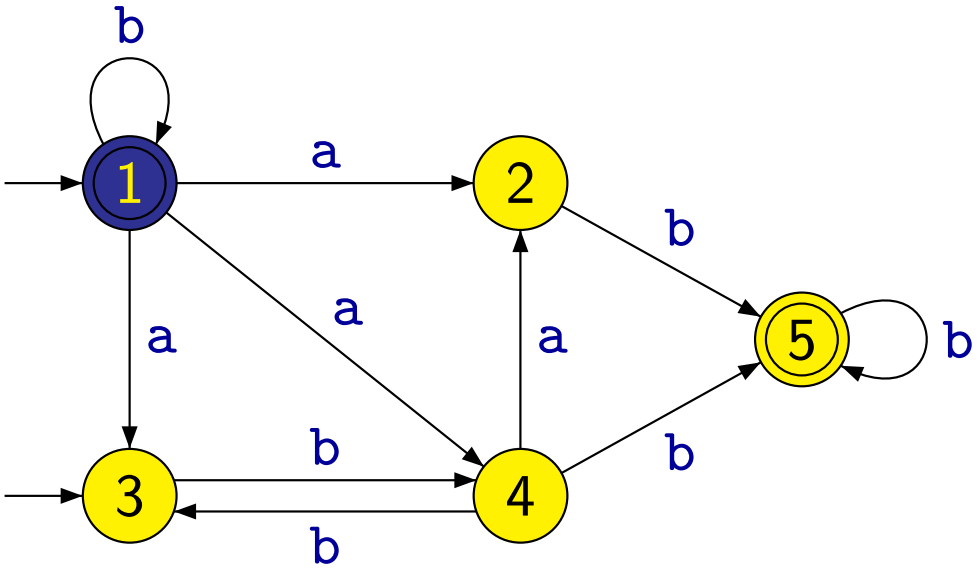
Jestliže jazyk L je regulární, pak také jeho doplněk $\overline{L}$ je regulární.

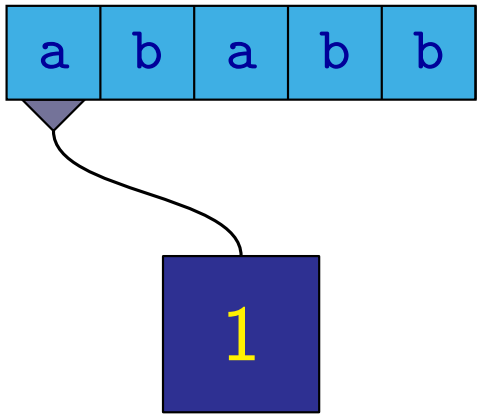
Nedeterministický konečný automat



- Z jednoho stavu může vézt libovolný (i nulový) počet přechodů označených stejným symbolem.
- V automatu může být víc než jeden počáteční stav.

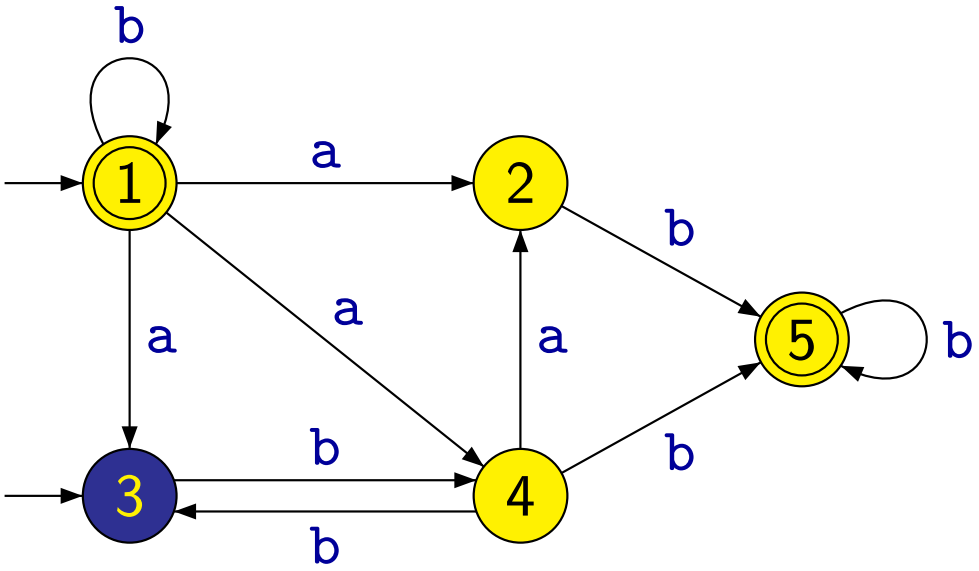
Nedeterministický konečný automat

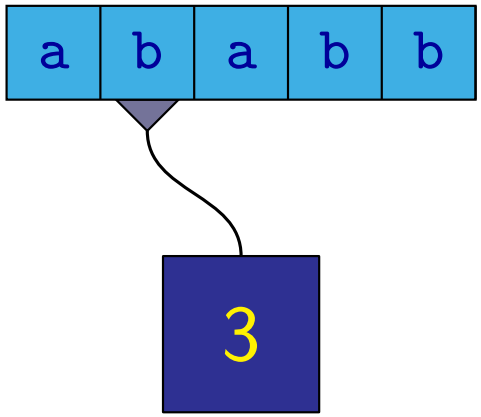




1

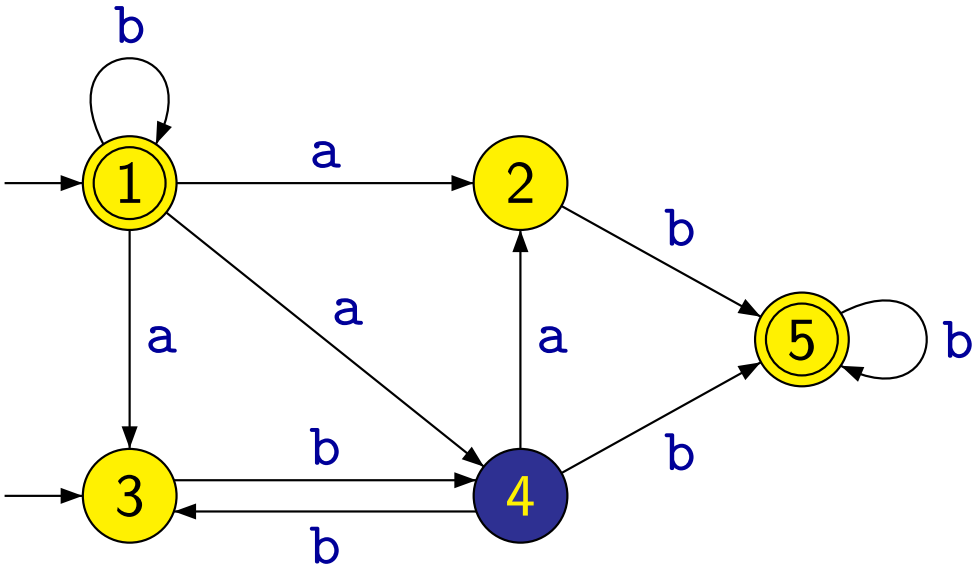
Nedeterministický konečný automat

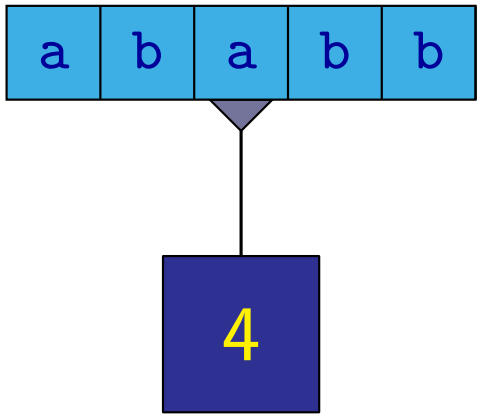




$1 \xrightarrow{a} 3$

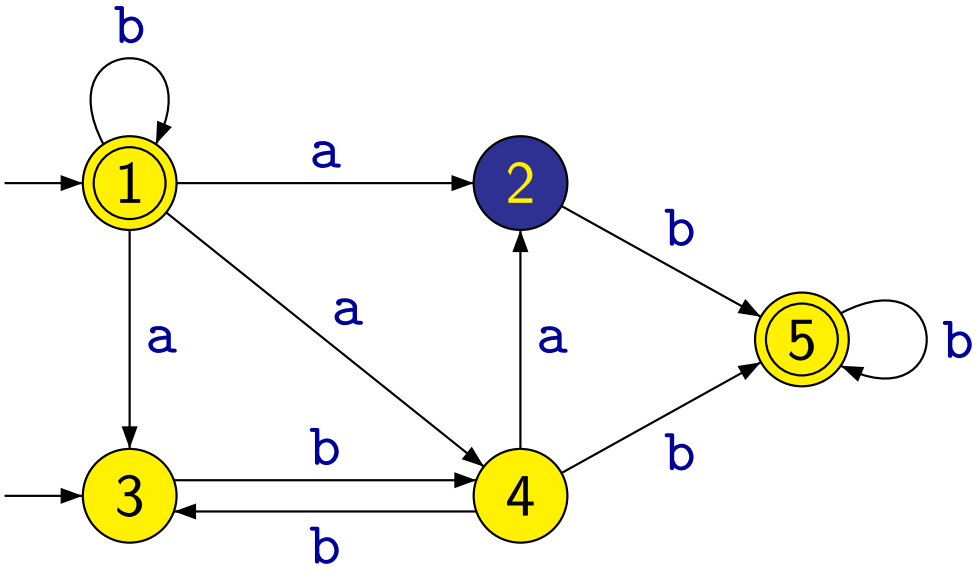
Nedeterministický konečný automat

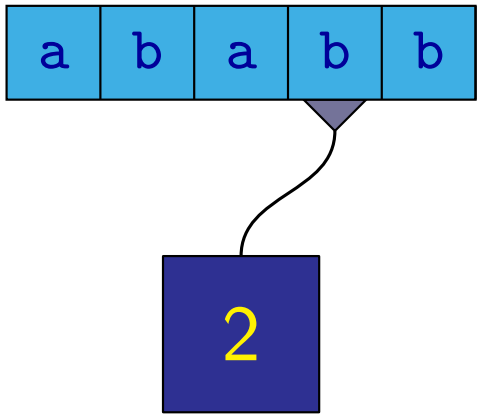




$1 \xrightarrow{a} 3 \xrightarrow{b} 4$

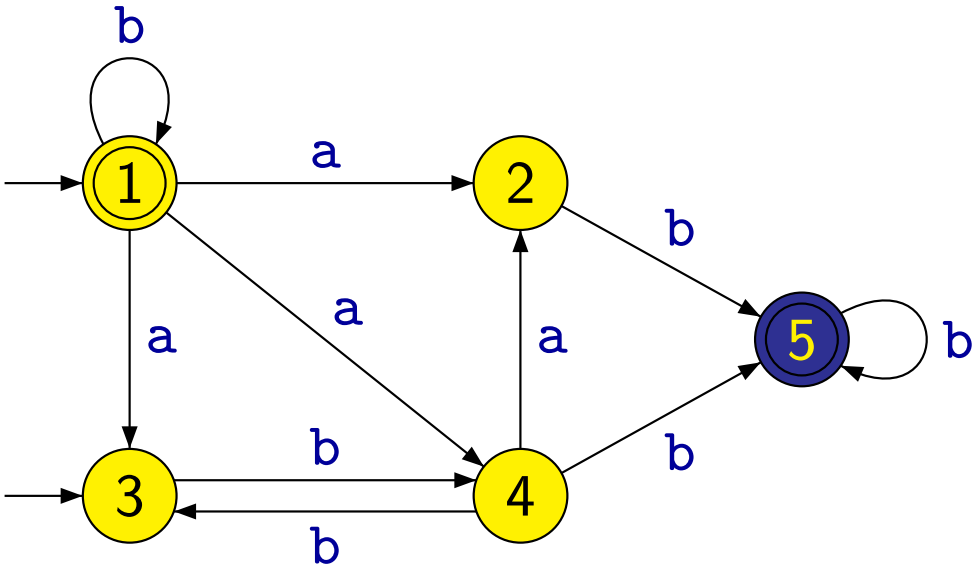
Nedeterministický konečný automat

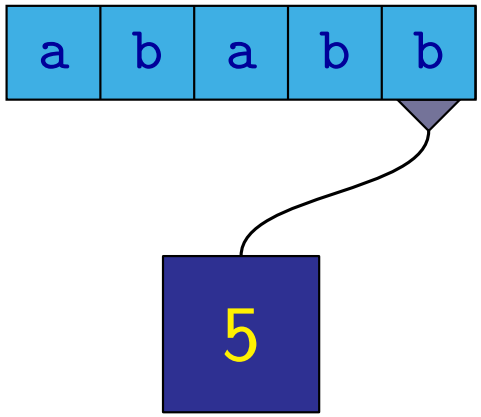




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2$$

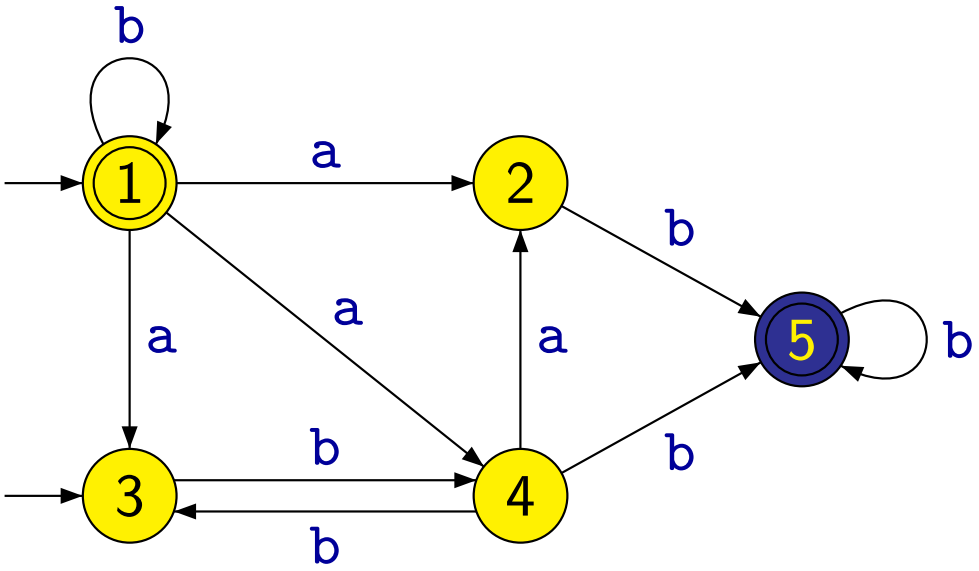
Nedeterministický konečný automat

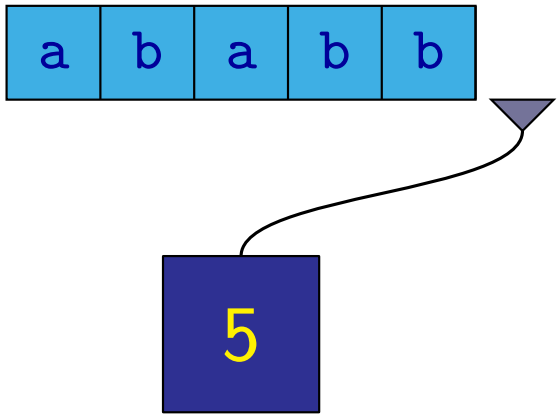




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5$$

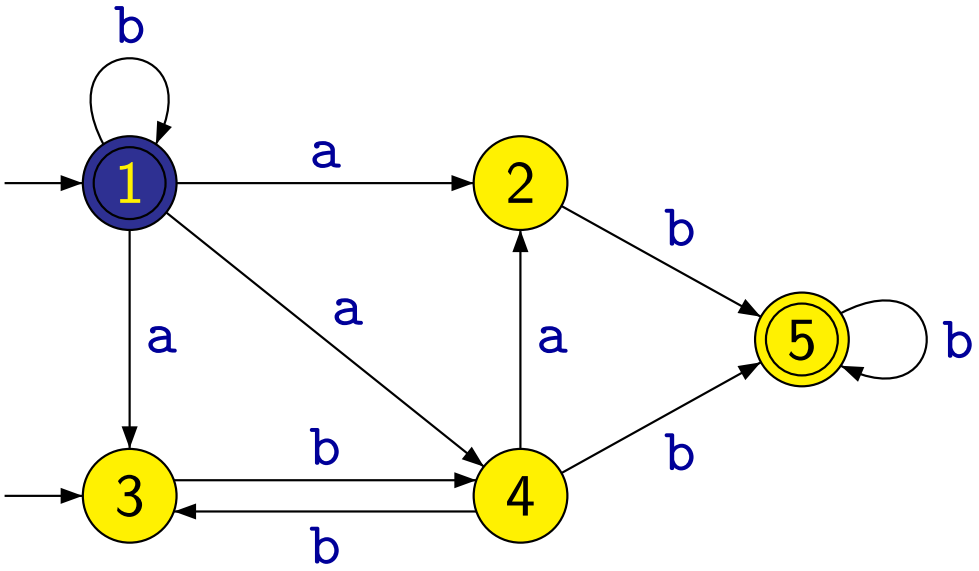
Nedeterministický konečný automat

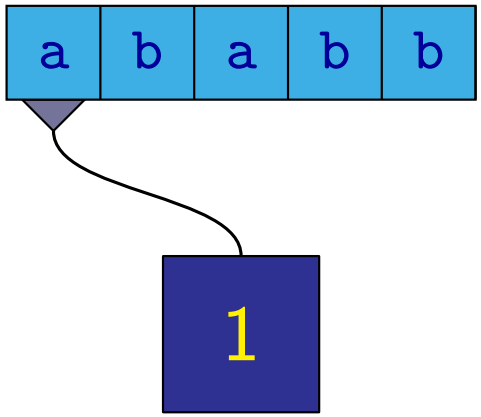




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$$

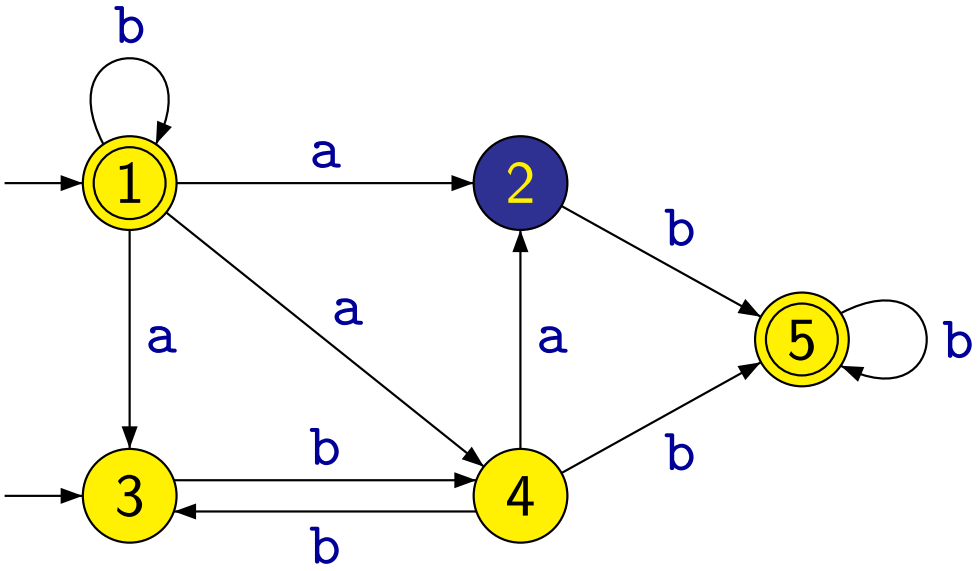
Nedeterministický konečný automat

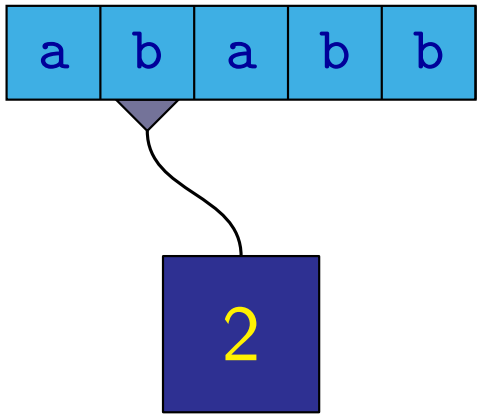




1

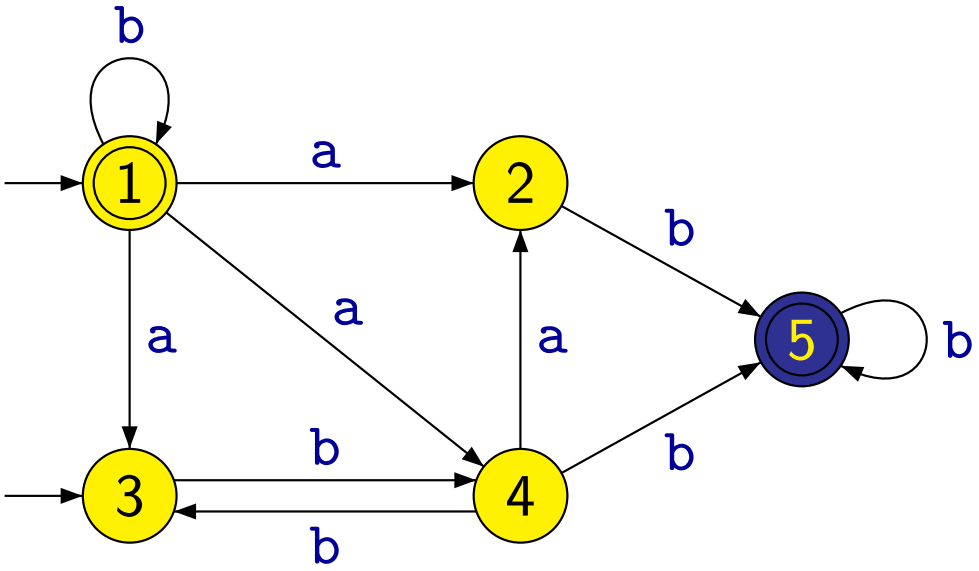
Nedeterministický konečný automat

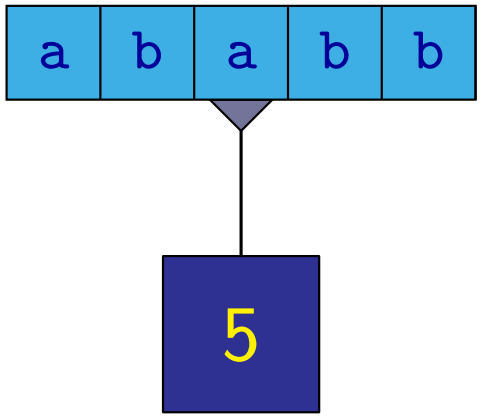




$1 \xrightarrow{a} 2$

Nedeterministický konečný automat

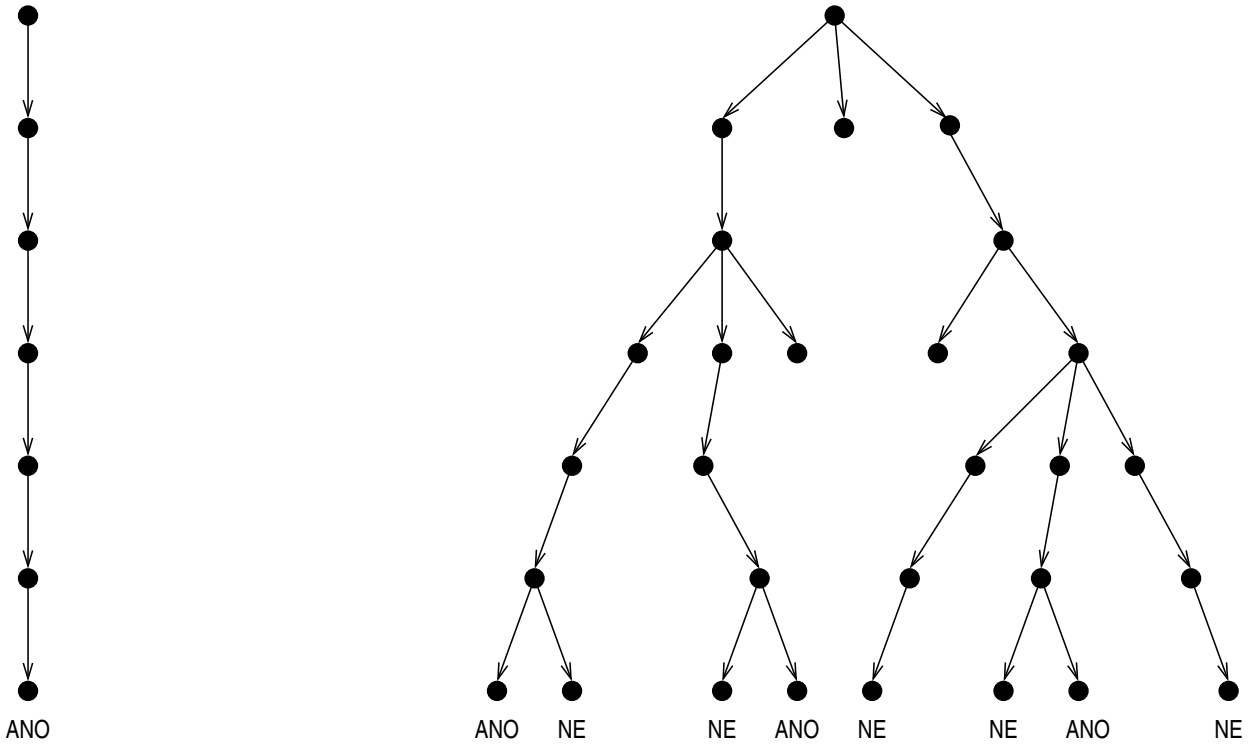




$$1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

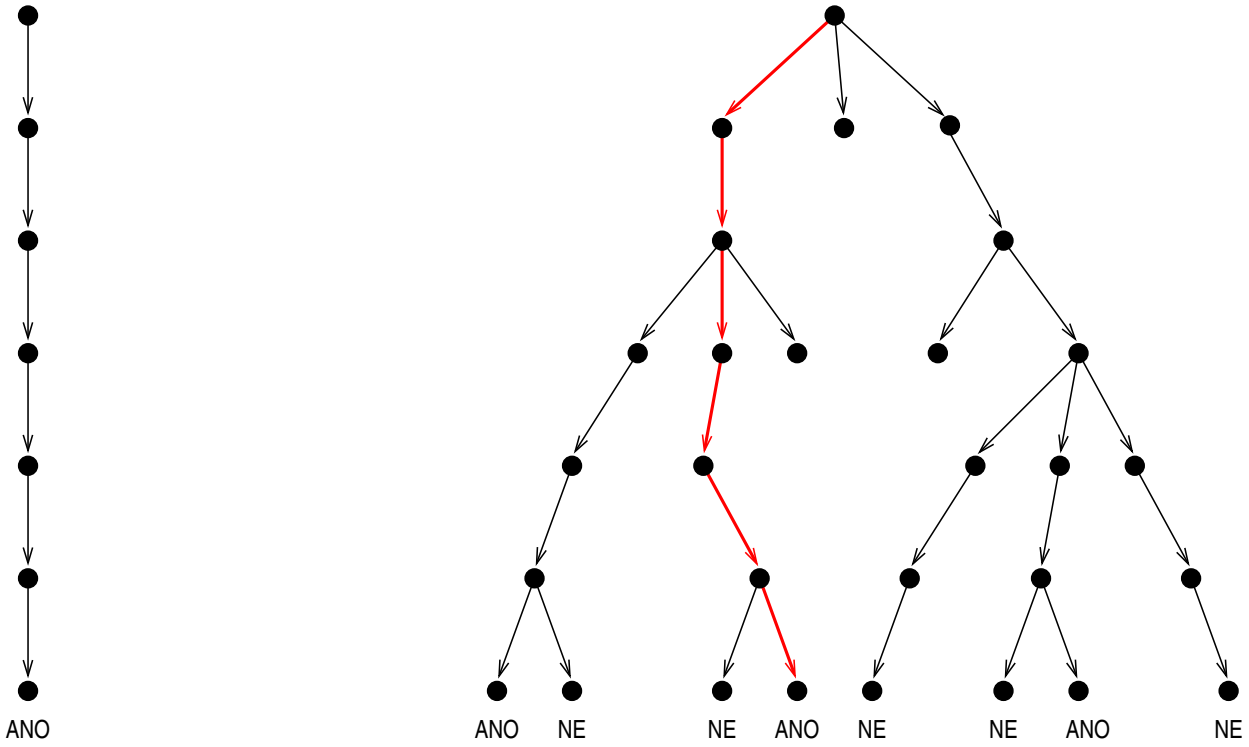
Nedeterministický konečný automat

Nedeterministický konečný automat přijímá dané slovo, jestliže **existuje** alespoň jeden jeho výpočet, který vede k přijetí tohoto slova.



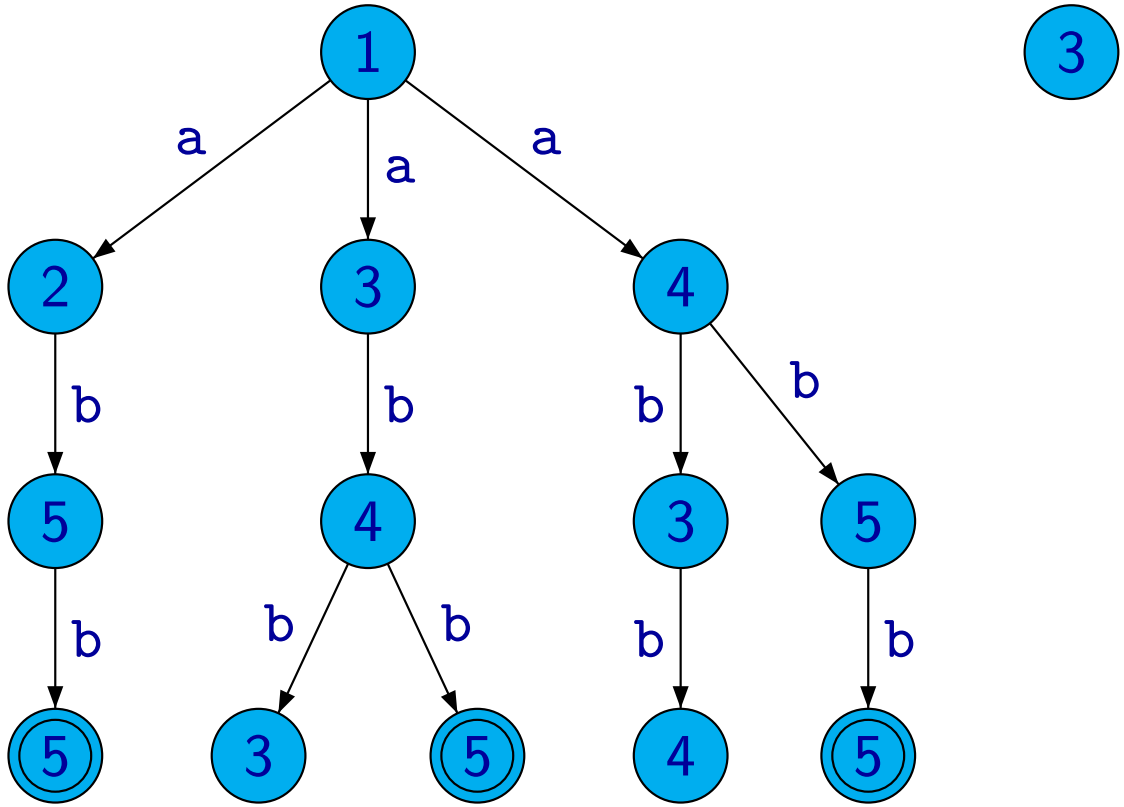
Nedeterministický konečný automat

Nedeterministický konečný automat přijímá dané slovo, jestliže **existuje** alespoň jeden jeho výpočet, který vede k přijetí tohoto slova.



Nedeterministický konečný automat

	a	b
$\leftrightarrow 1$	$2, 3, 4$	1
2	$-$	5
$\rightarrow 3$	$-$	4
4	2	$3, 5$
$\leftarrow 5$	$-$	5



Příklad: Les reprezentující všechny možné výpočty nad slovem abb.

Nedeterministický konečný automat

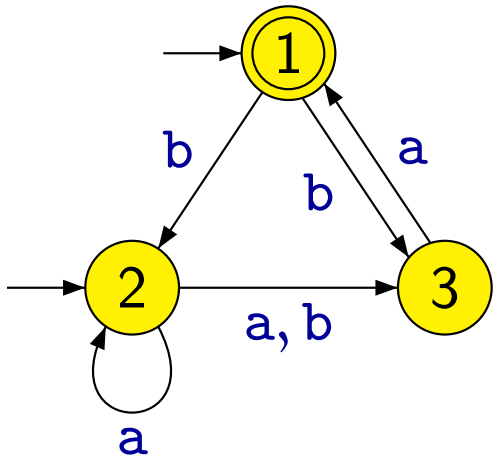
Formálně je **nedeterministický konečný automat** (**NKA**) definován jako pětice

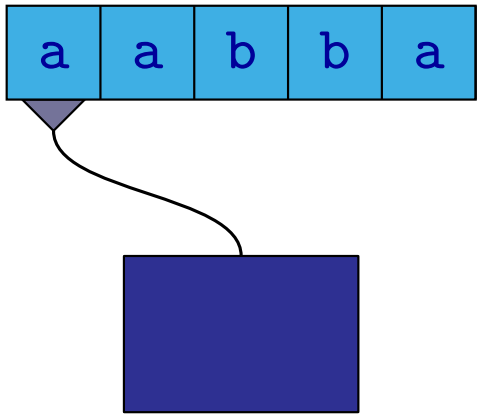
$$(Q, \Sigma, \delta, I, F)$$

kde:

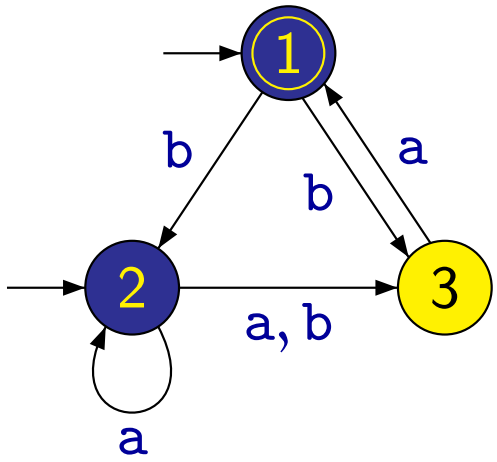
- Q je konečná množina **stavů**
- Σ je konečná **abeceda**
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ je **přechodová funkce**
- $I \subseteq Q$ je množina **počátečních stavů**
- $F \subseteq Q$ je množina **přijímajících stavů**

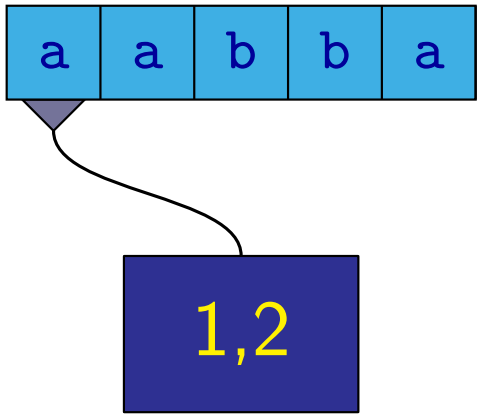
Převod NKA na DKA



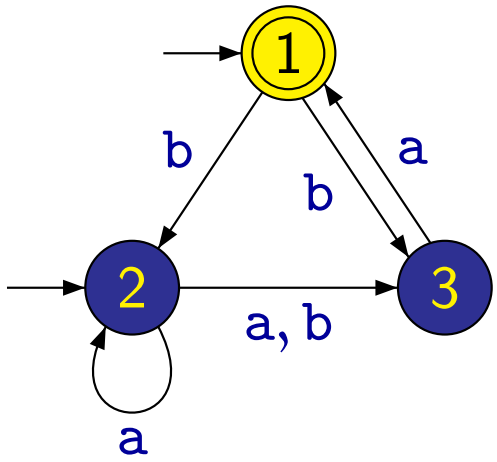


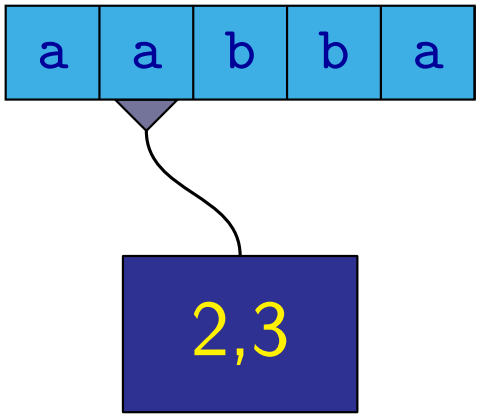
Převod NKA na DKA



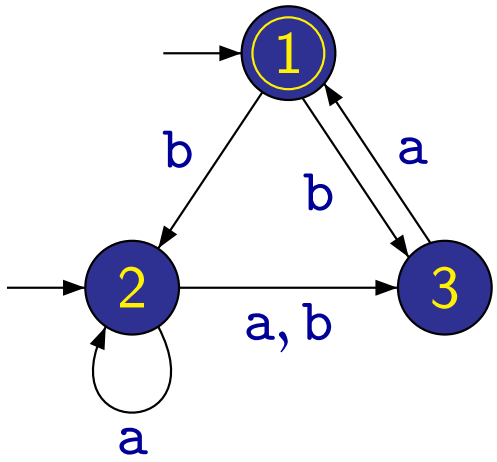


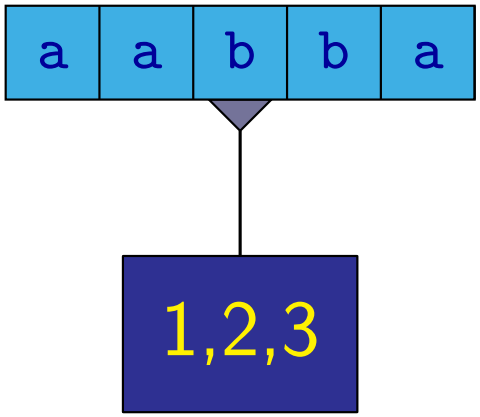
Převod NKA na DKA



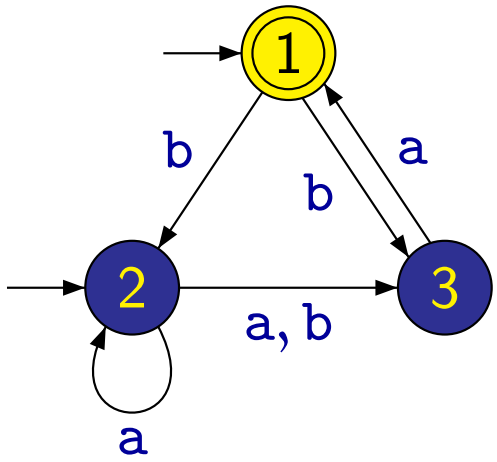


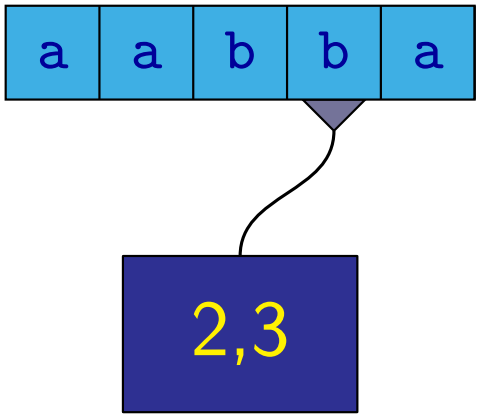
Převod NKA na DKA



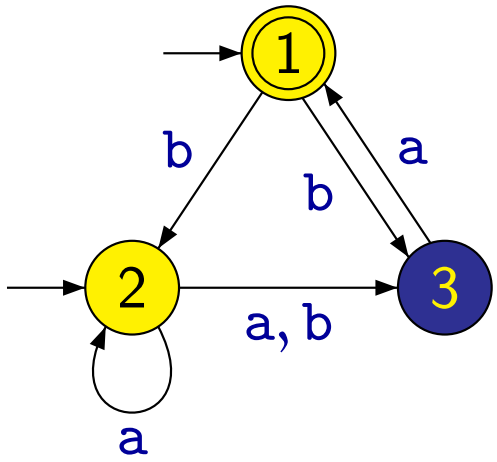


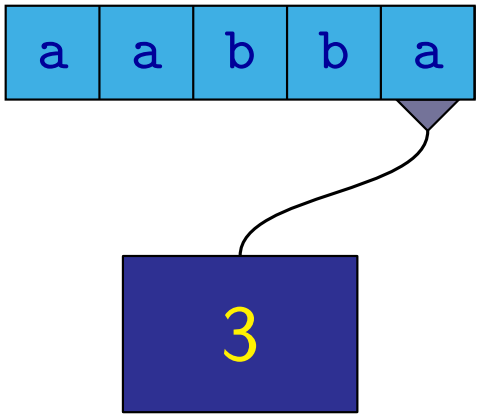
Převod NKA na DKA



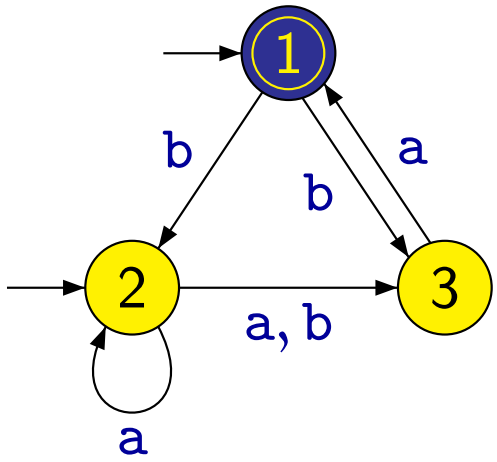


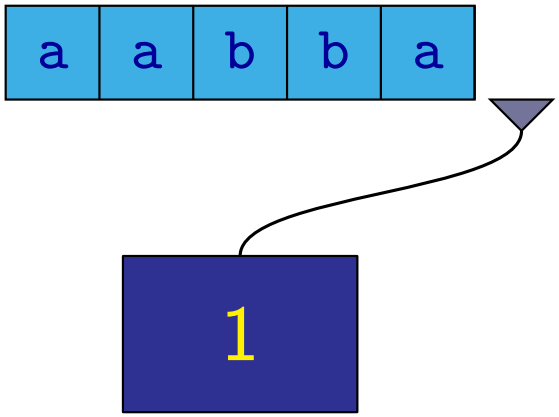
Převod NKA na DKA



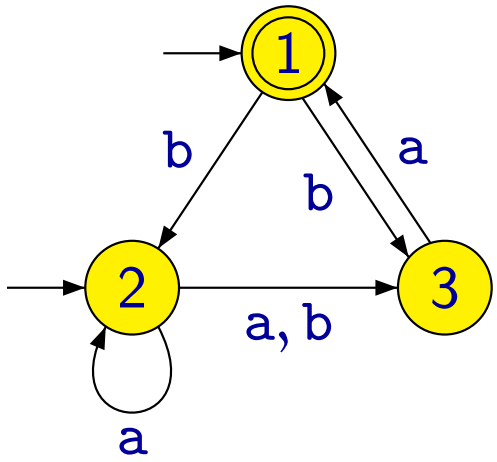


Převod NKA na DKA

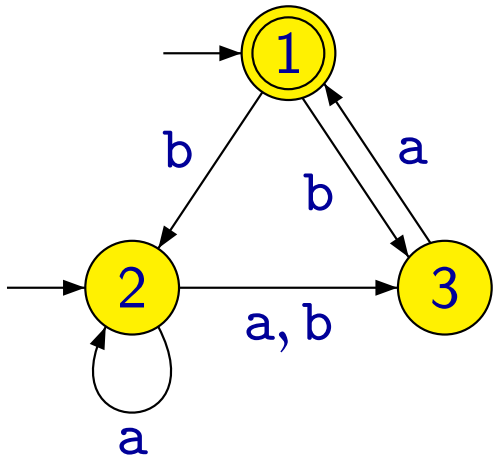


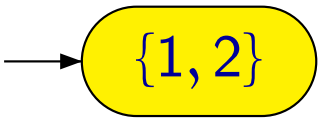


Převod NKA na DKA

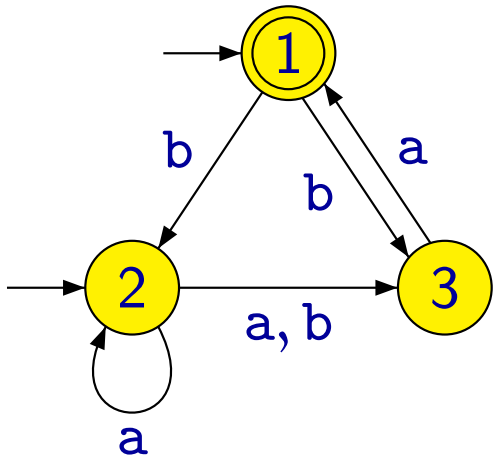


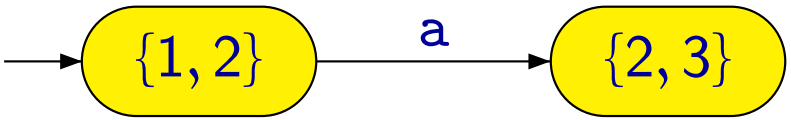
Převod NKA na DKA



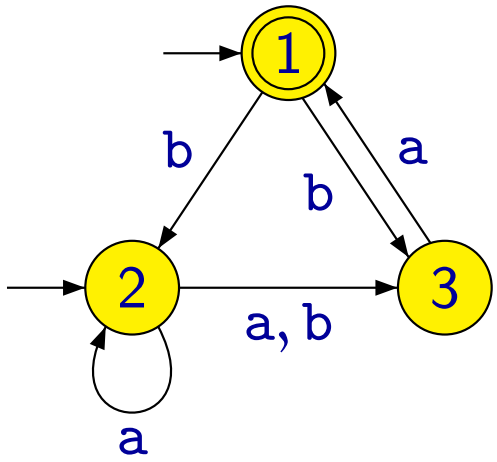


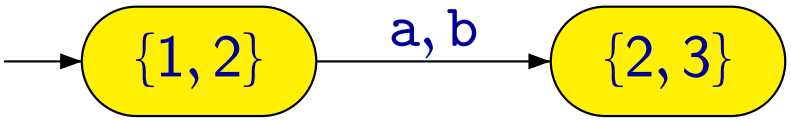
Převod NKA na DKA



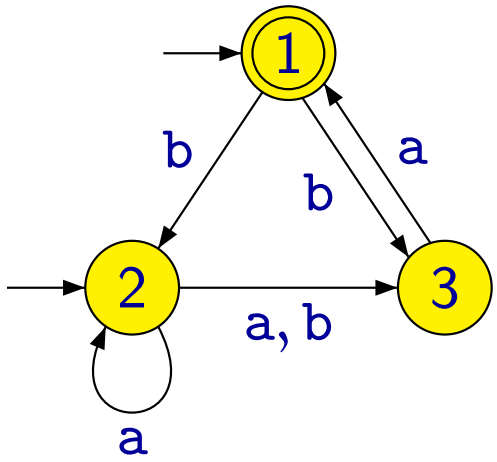


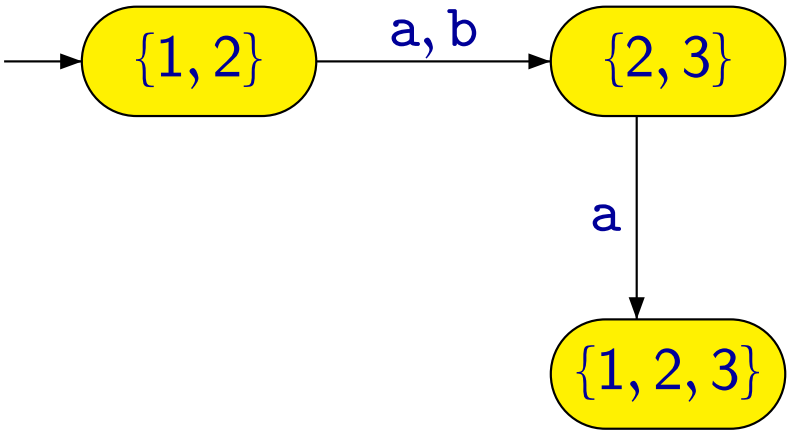
Převod NKA na DKA



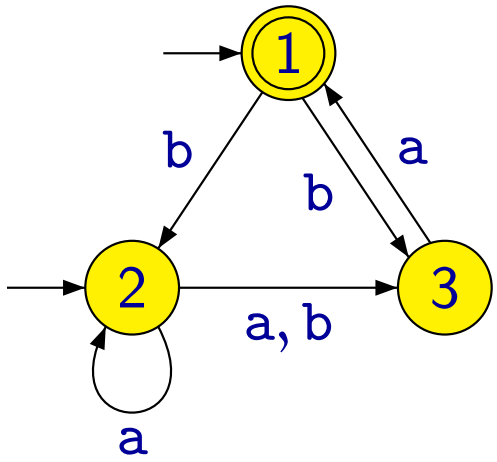


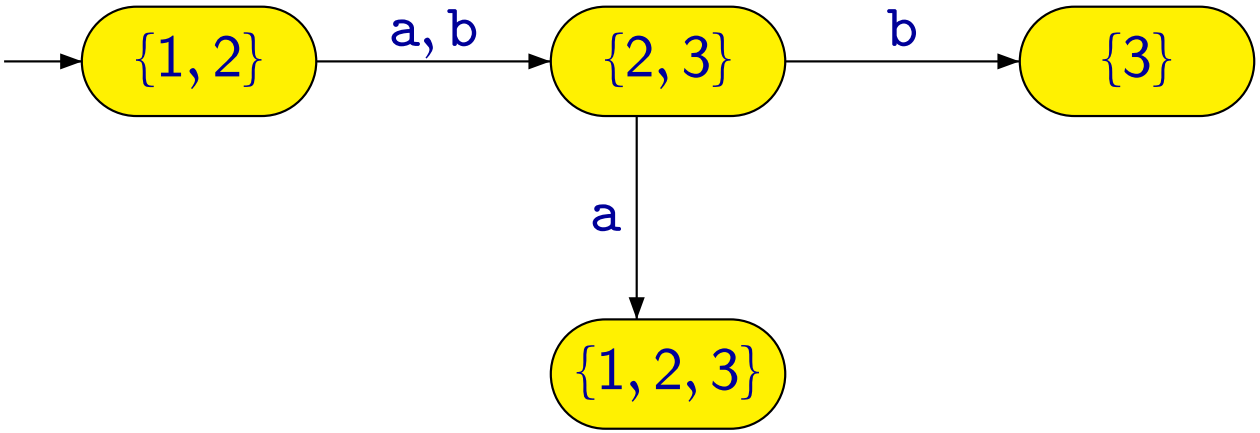
Převod NKA na DKA



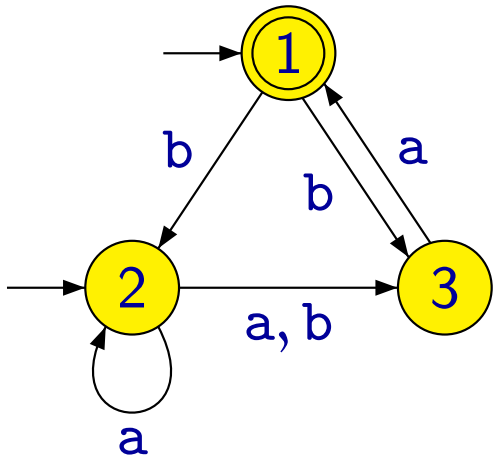


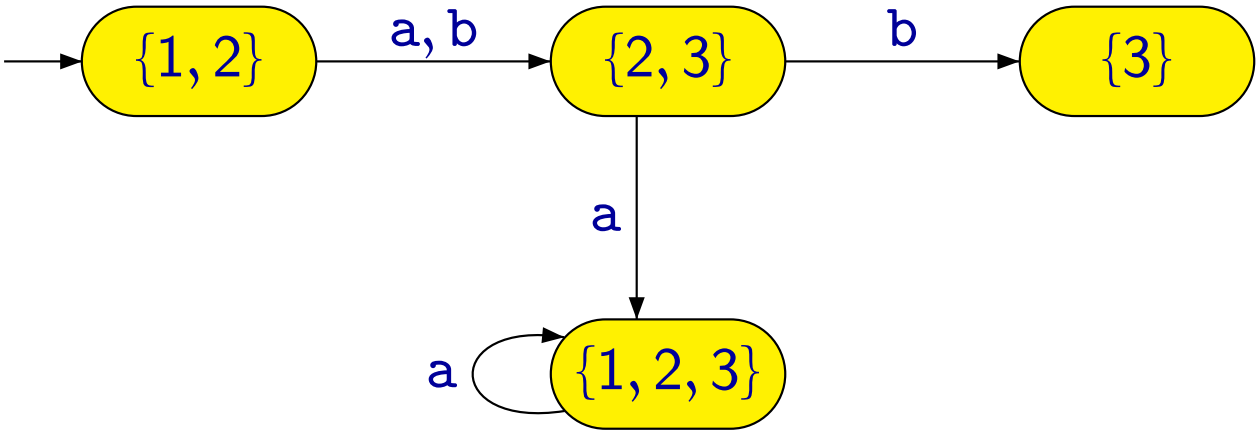
Převod NKA na DKA



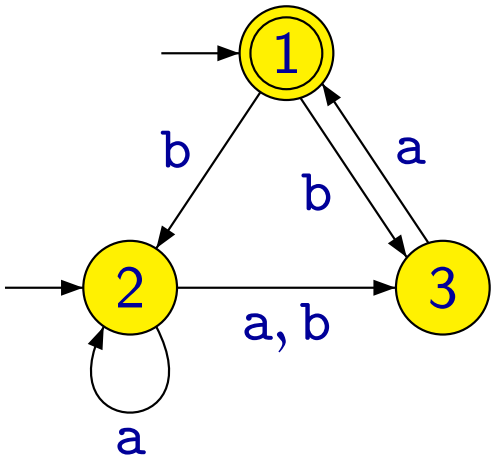


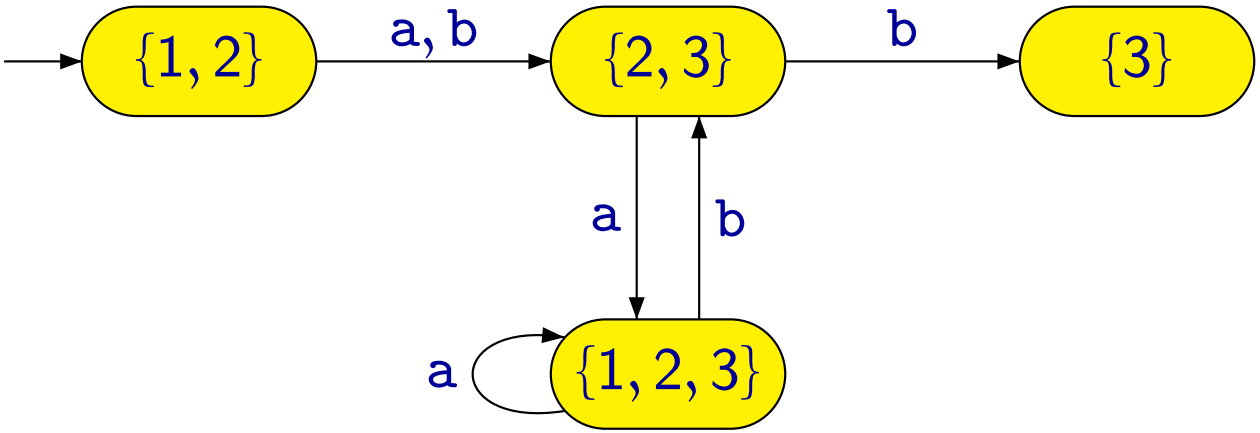
Převod NKA na DKA



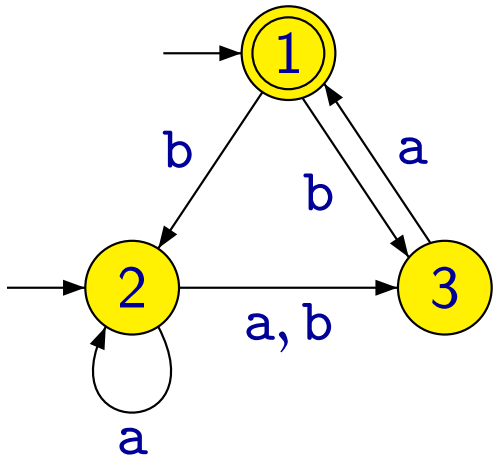


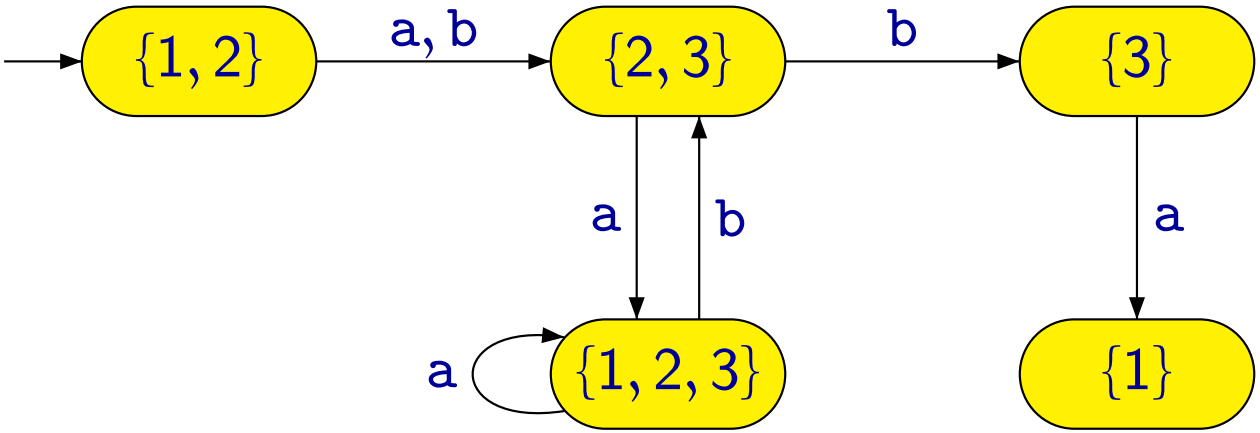
Převod NKA na DKA



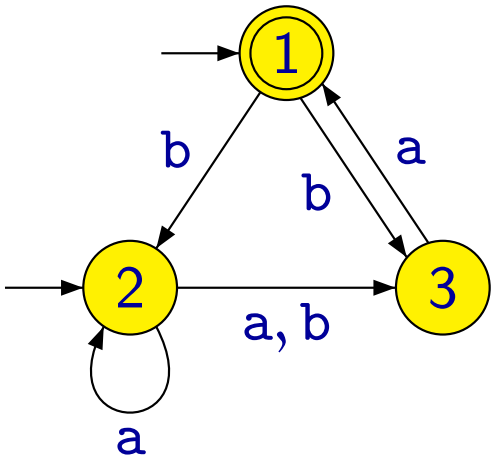


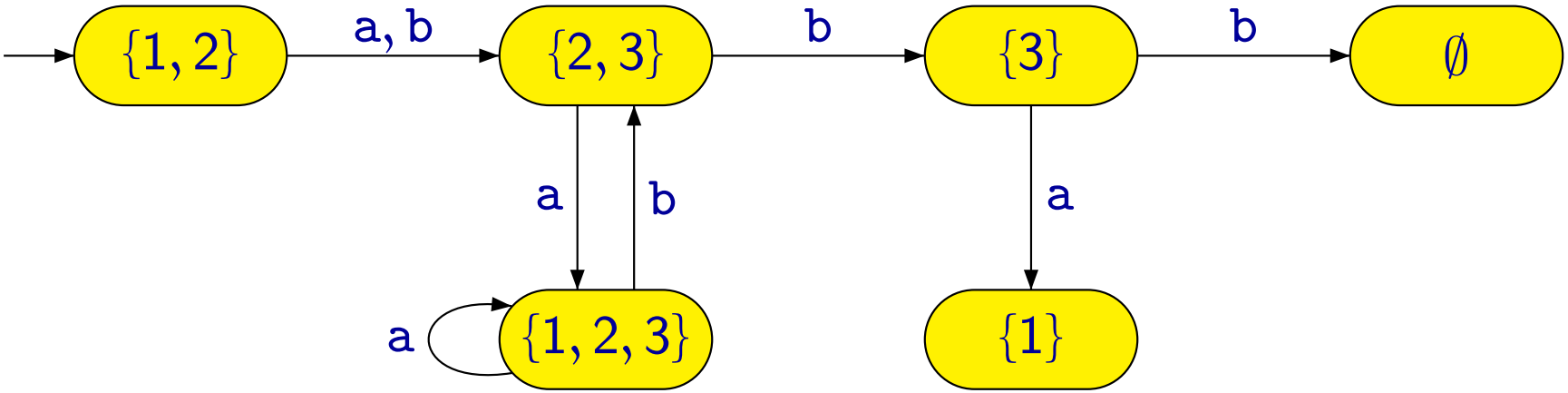
Převod NKA na DKA



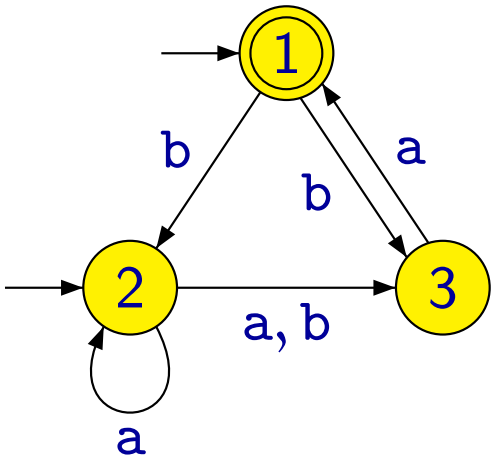


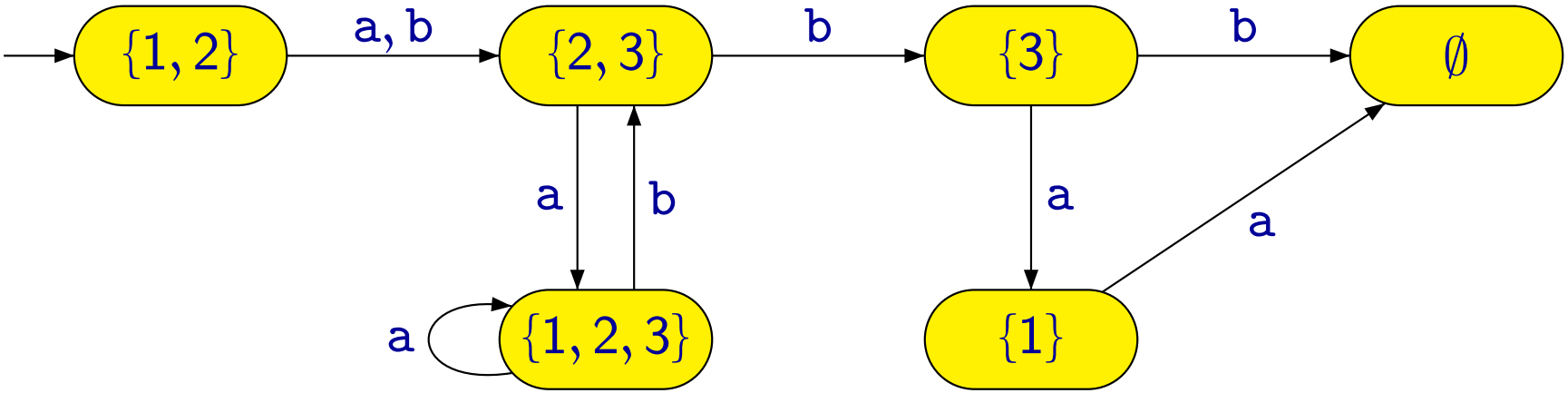
Převod NKA na DKA



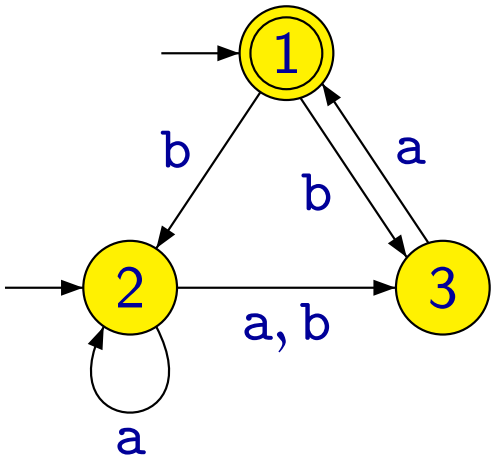


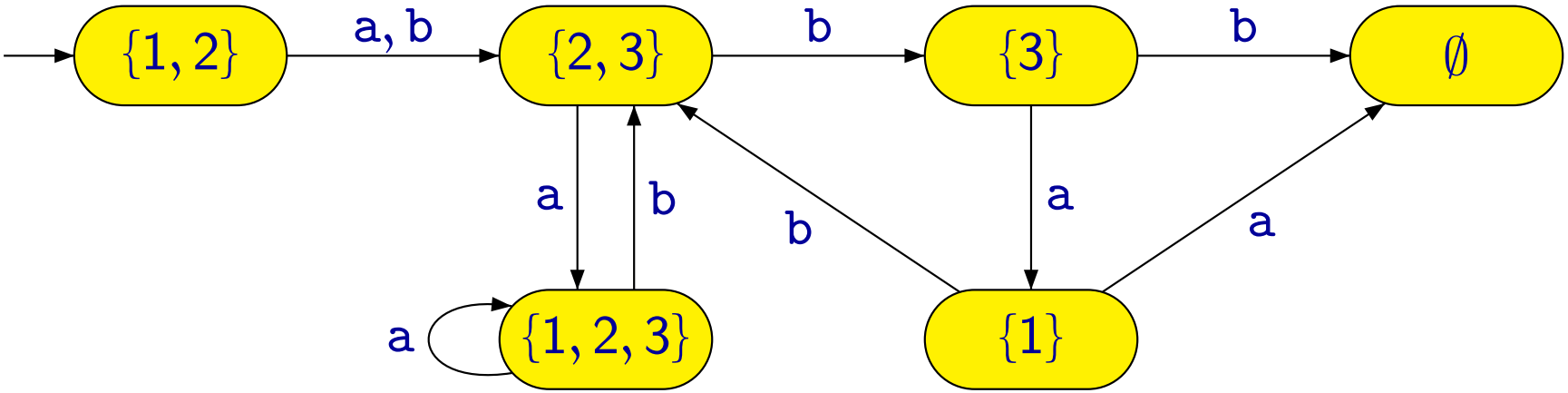
Převod NKA na DKA



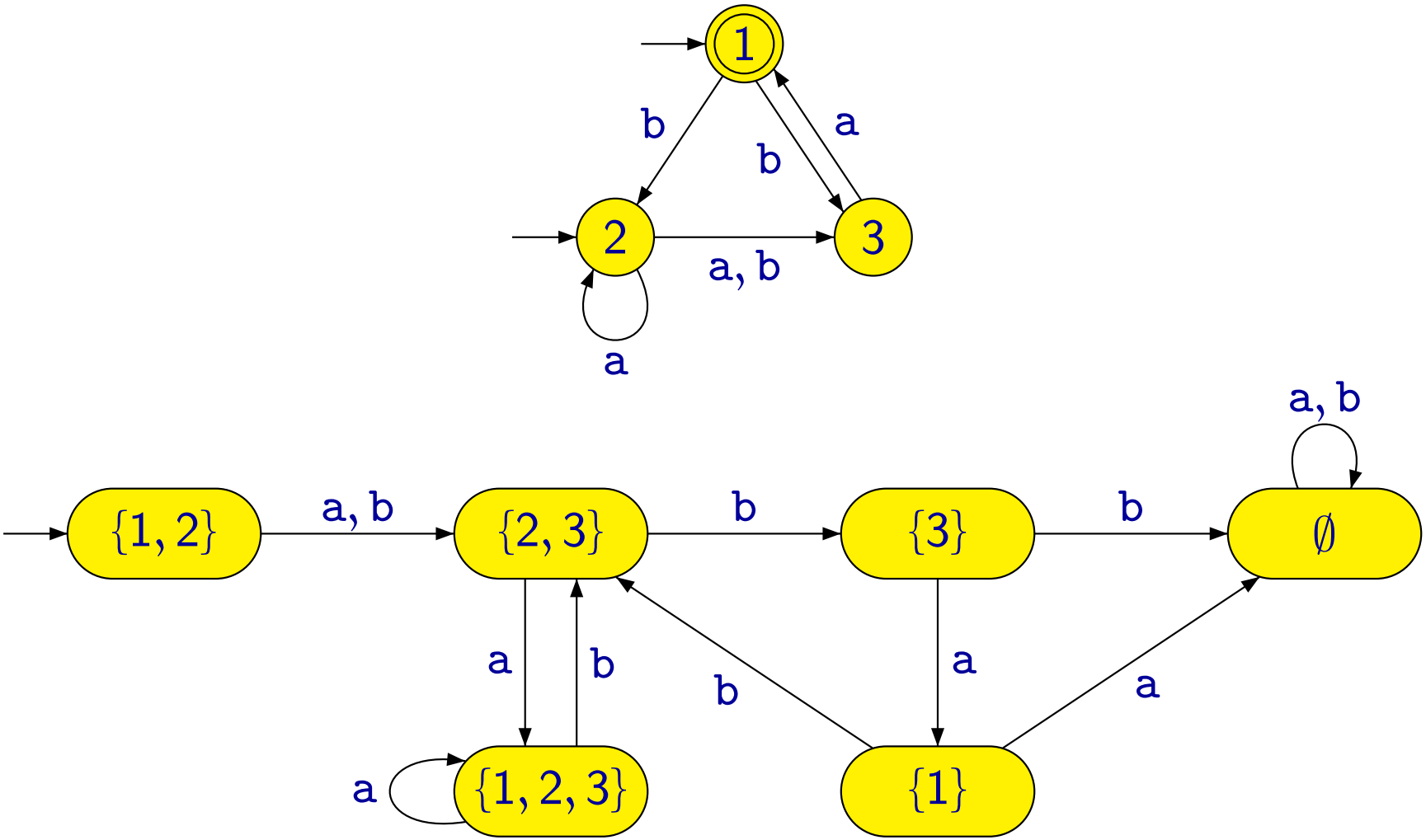


Převod NKA na DKA

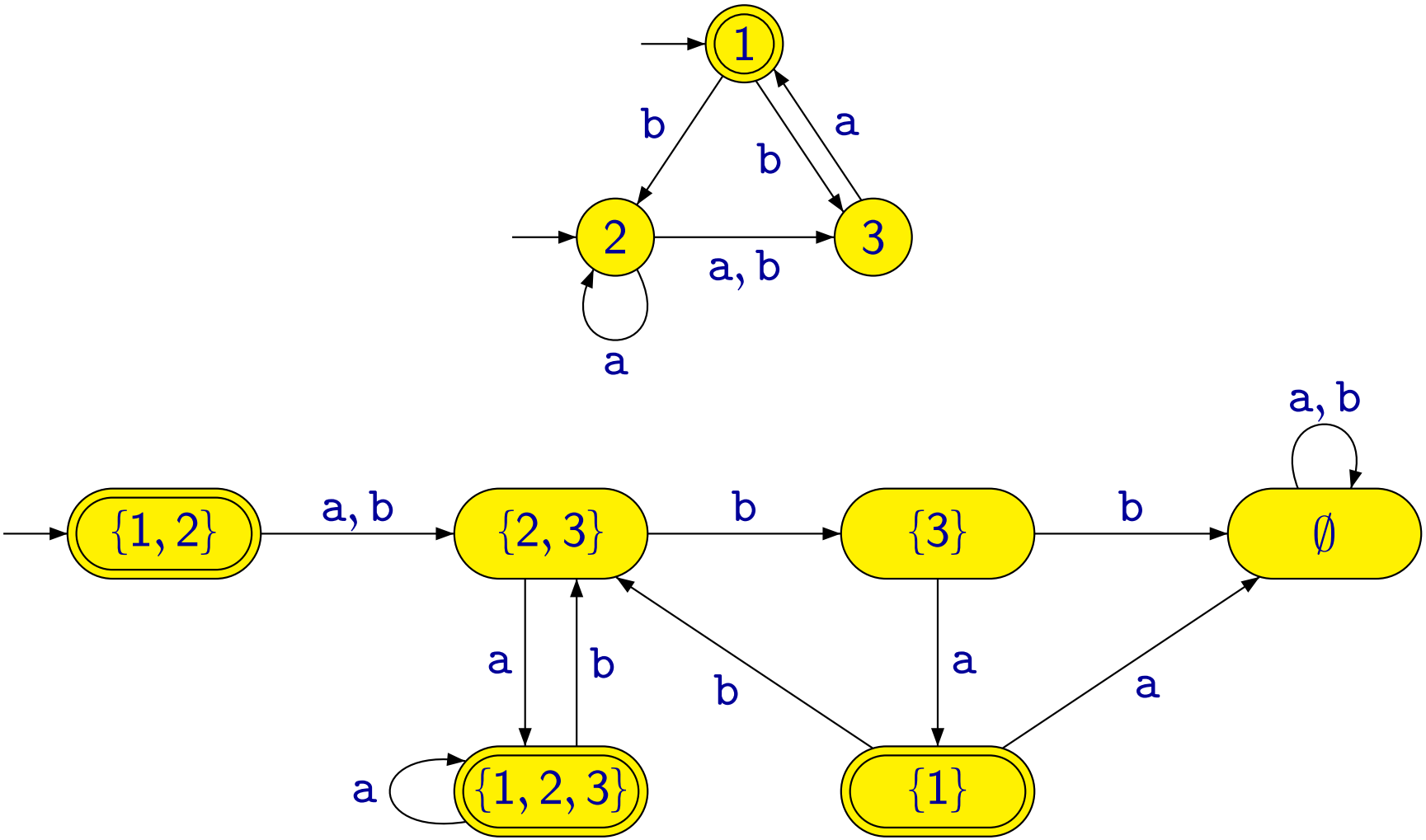




Převod NKA na DKA



Převod NKA na DKA



Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2, 3$
$\rightarrow 2$	$2, 3$	3
3	1	—

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	
$\{2,3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	
$\leftarrow \{1,2,3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$		
$\{3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	
$\{3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	
$\leftarrow \{1\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$		
$\emptyset$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	
$\emptyset$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2,3\}$
$\emptyset$		

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$

Převod NKA na DKA

	a	b
$\leftrightarrow 1$	—	$2,3$
$\rightarrow 2$	$2,3$	3
3	1	—

	a	b
$\leftrightarrow \{1,2\}$	$\{2,3\}$	$\{2,3\}$
$\{2,3\}$	$\{1,2,3\}$	$\{3\}$
$\leftarrow \{1,2,3\}$	$\{1,2,3\}$	$\{2,3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2,3\}$
$\emptyset$	$\emptyset$	$\emptyset$

	a	b
$\leftrightarrow 1$	2	2
2	3	4
$\leftarrow 3$	3	2
4	5	6
$\leftarrow 5$	6	2
6	6	6

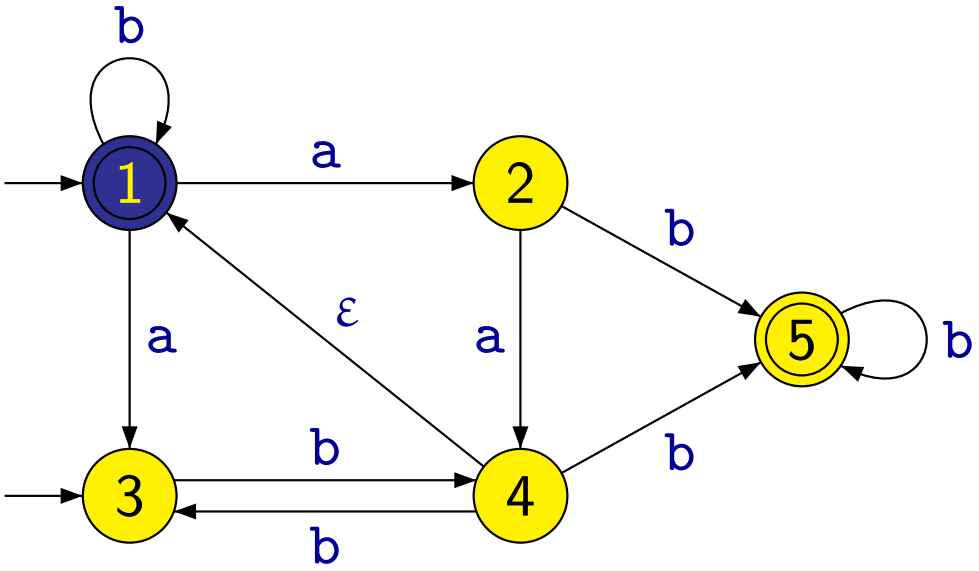
Převod NKA na DKA

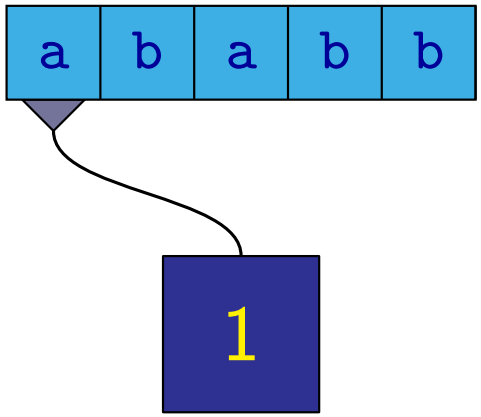
Poznámka: Při převodu nedeterministického automatu, který má n stavů, může mít výsledný deterministický automat až 2^n stavů.

Například při převodu automatu, který má 20 stavů, může vzniknout automat, který má $2^{20} = 1048576$ stavů.

Často má sice výsledný automat podstatně méně než 2^n stavů, nicméně tyto nejhorší případy občas nastávají.

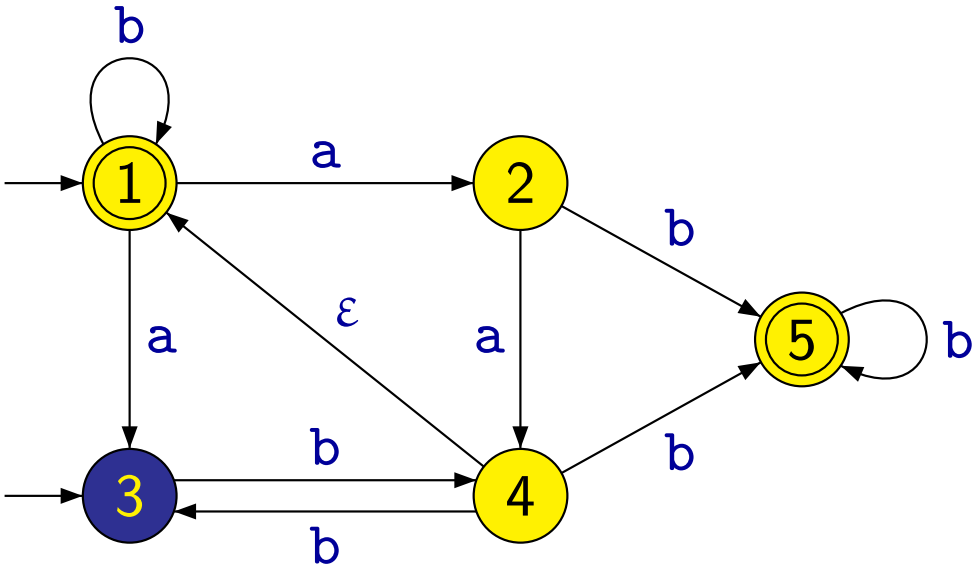
Zobecněný nedeterministický konečný automat

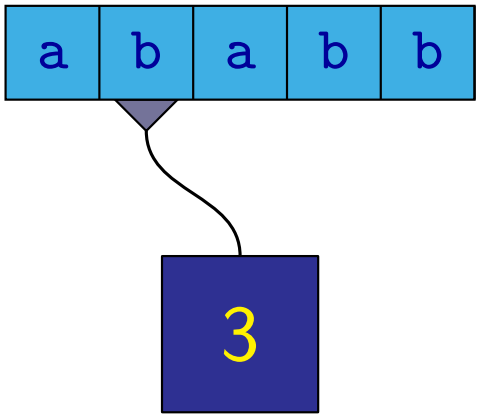




1

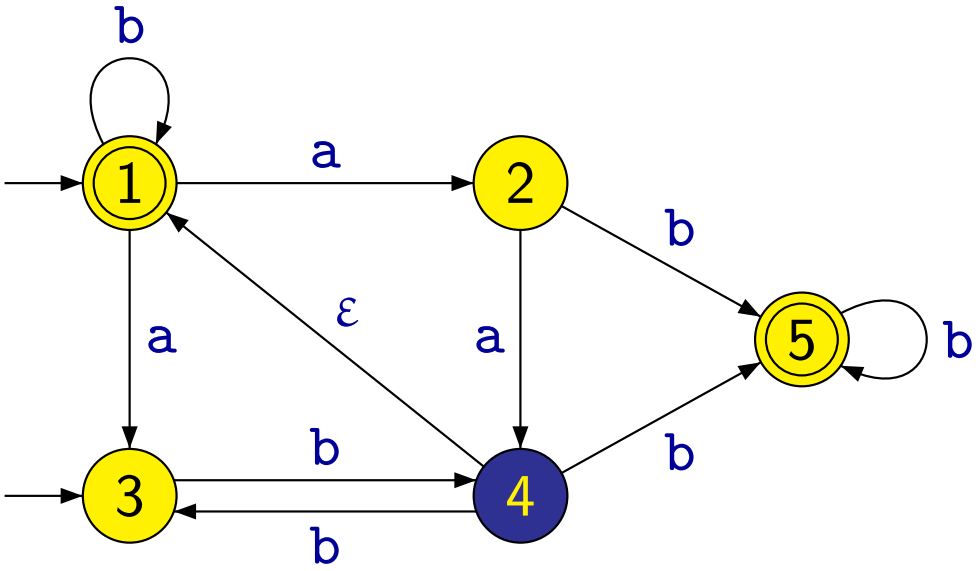
Zobecněný nedeterministický konečný automat

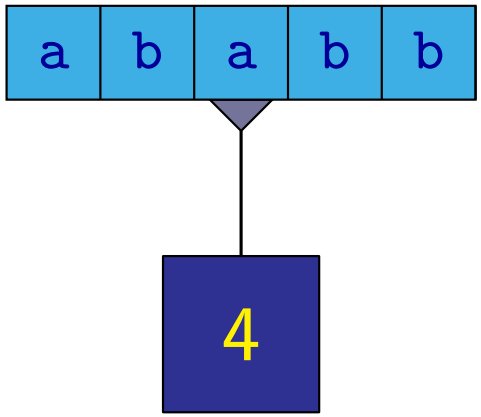




$1 \xrightarrow{a} 3$

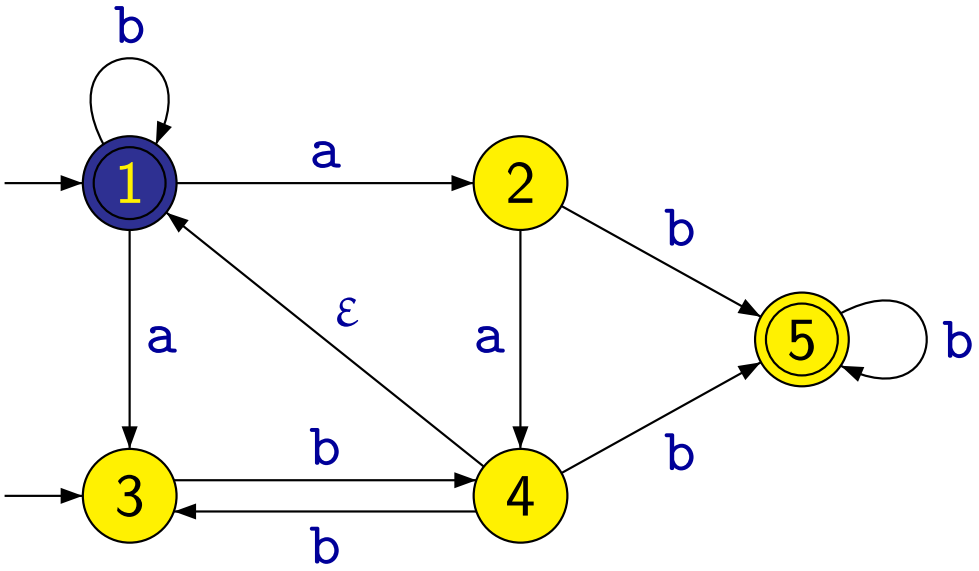
Zobecněný nedeterministický konečný automat

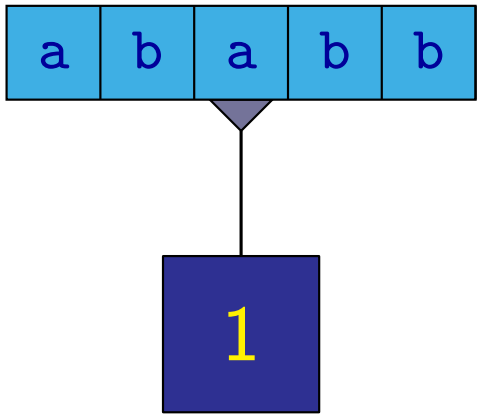




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

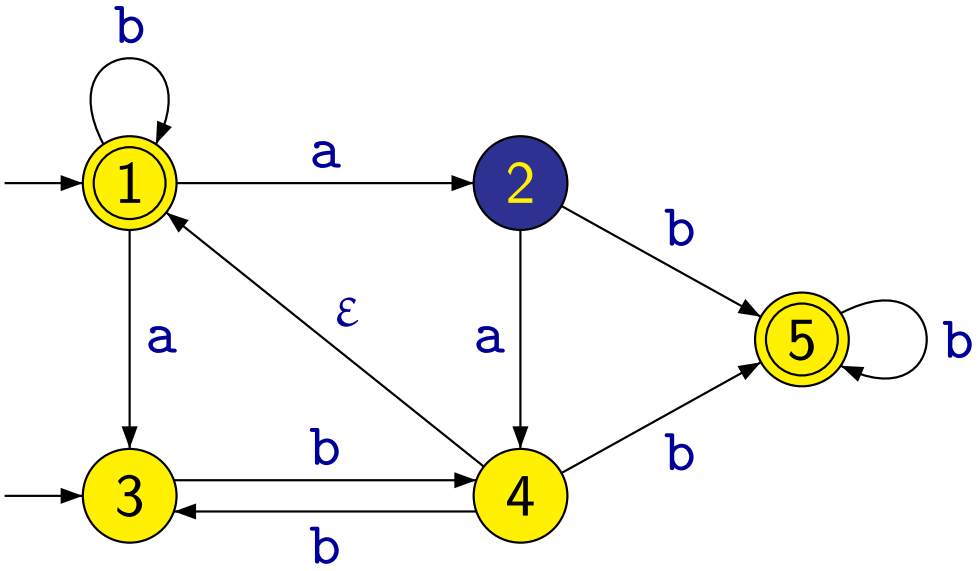
Zobecněný nedeterministický konečný automat

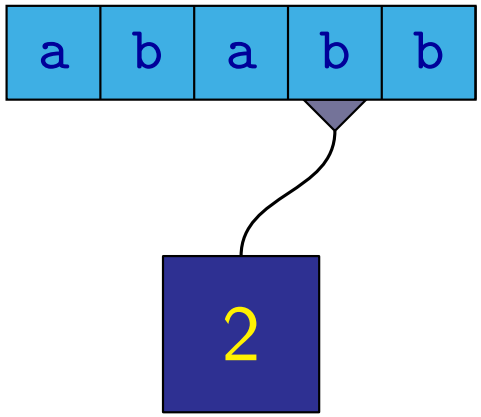




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1$$

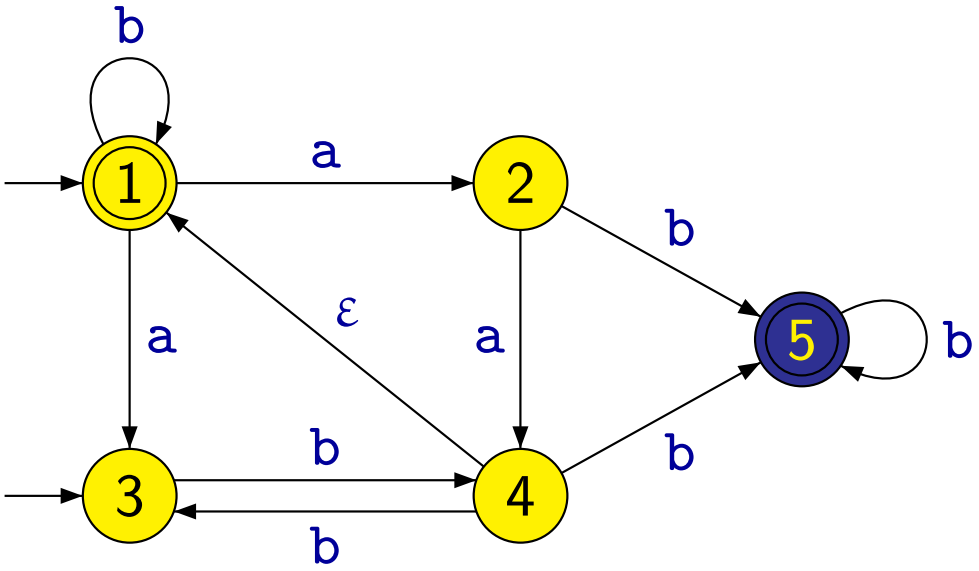
Zobecněný nedeterministický konečný automat

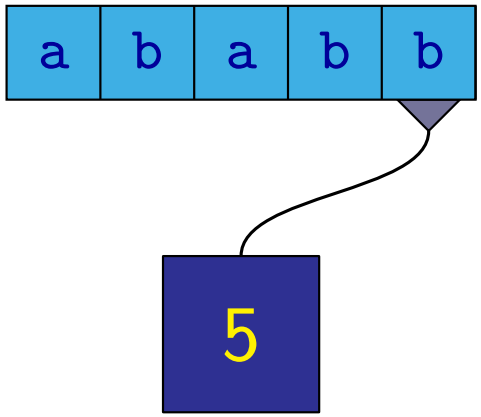




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2$$

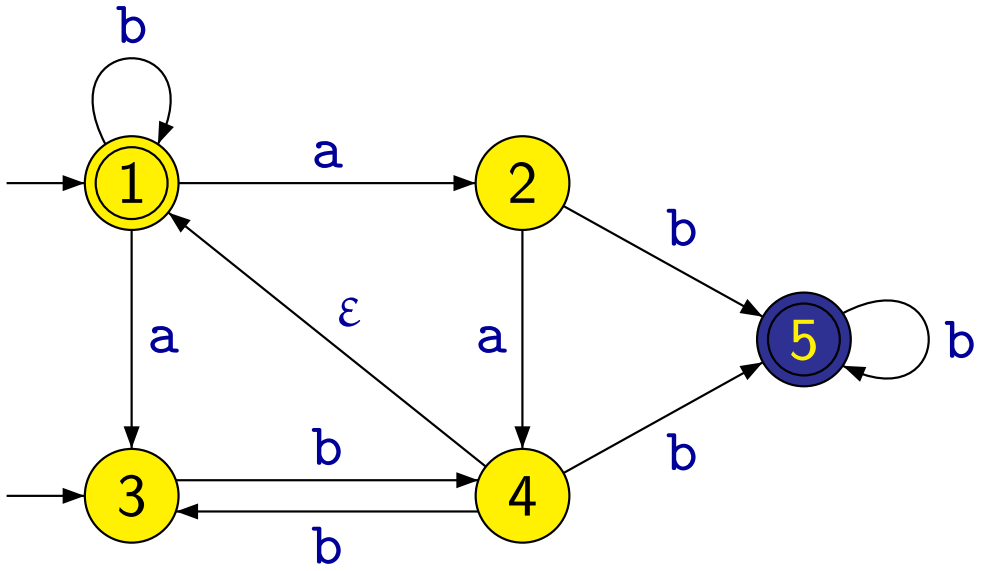
Zobecněný nedeterministický konečný automat

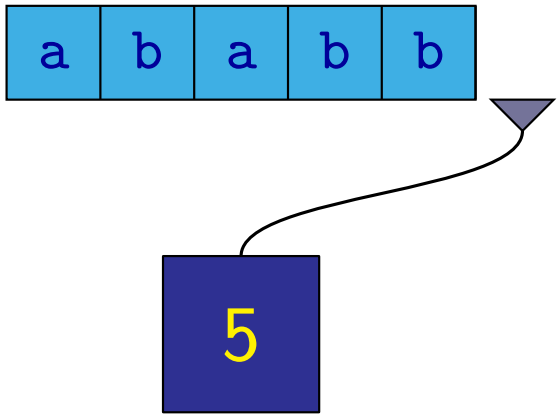




$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

Zobecněný nedeterministický konečný automat





$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$$

Zobecněný nedeterministický konečný automat

Oproti nedeterministickému konečnému automatu má **zobecněný nedeterministický konečný automat** tzv. ε**-přechody**, tj. přechody označené symbolem ε.

Při provádění ε-přechodu se mění pouze stav řídící jednotky, ale hlava na pásce se neposouvá.

Poznámka: Výpočty zobecněného nedeterministického automatu mohou být libovolně dlouhé a dokonce i nekonečné (pokud graf obsahuje cyklus tvořený ε-přechody) bez ohledu na délku slova na pásce.

Zobecněný nedeterministický konečný automat

Formálně je **zobecněný nedeterministický konečný automat** (**ZNKA**) definován jako pětice

$$(Q, \Sigma, \delta, I, F)$$

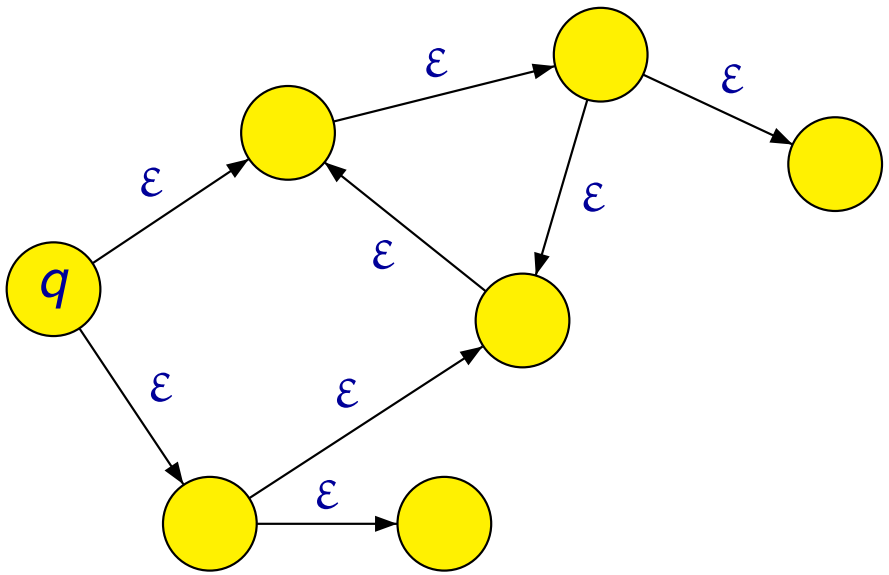
kde:

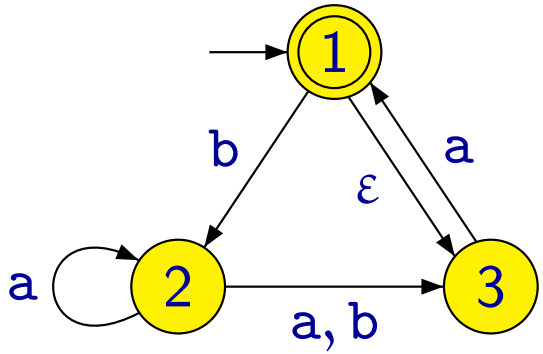
- Q je konečná množina **stavů**
- Σ je konečná **abeceda**
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ je **přechodová funkce**
- $I \subseteq Q$ je množina **počátečních stavů**
- $F \subseteq Q$ je množina **přijímajících stavů**

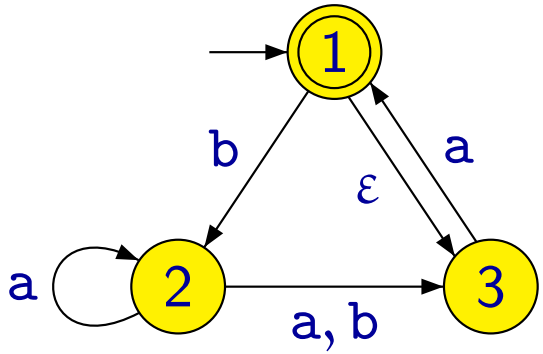
Poznámka: Na NKA můžeme nahlížet jako na speciální případ ZNKA, kde $\delta(q, \varepsilon) = \emptyset$ pro všechna $q \in Q$.

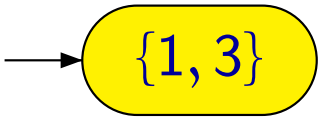
Převod na deterministický konečný automat

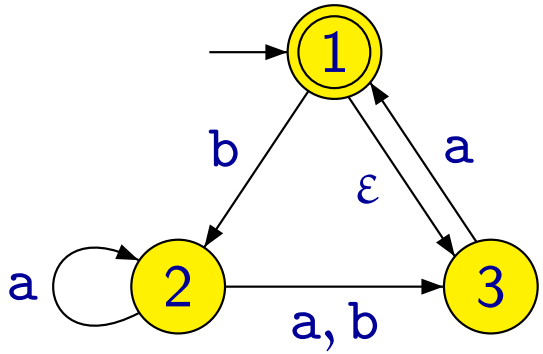
Zobecněný nedeterministický konečný automat je možné převést na deterministický podobnou konstrukcí jako nedeterministický konečný automat, s tím rozdílem, že do množin stavů musíme vždy přidat navíc i všechny stavy dosažitelné z již přidaných stavů nějakou sekvencí ε-přechodů.

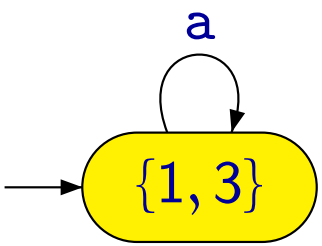


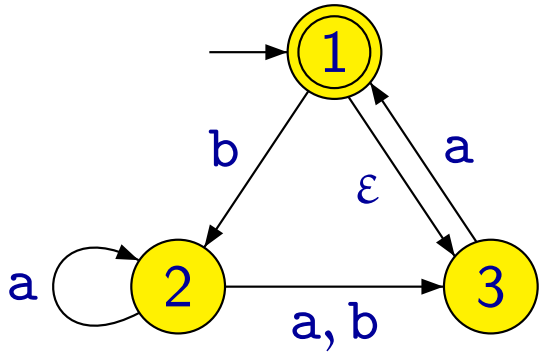

1
2
3
b
ε
a
a
a, b

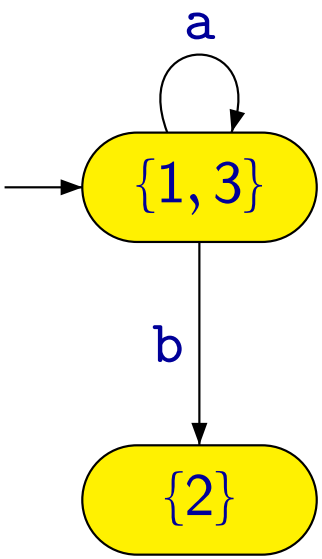

1
b
a
ε
a
2
3
a, b

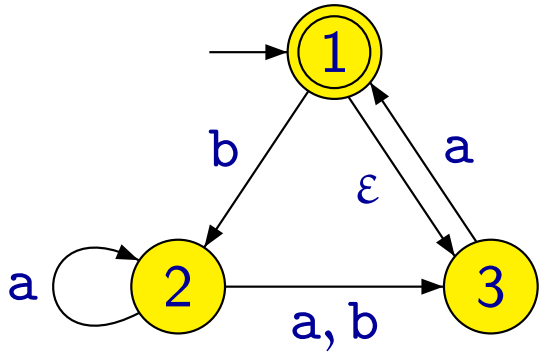

{1, 3}

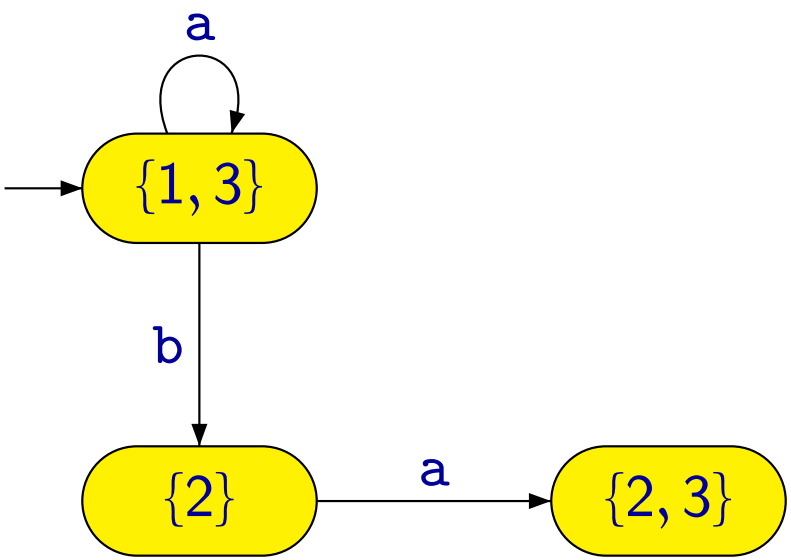

1
b
a
ε
a
2
3
a, b

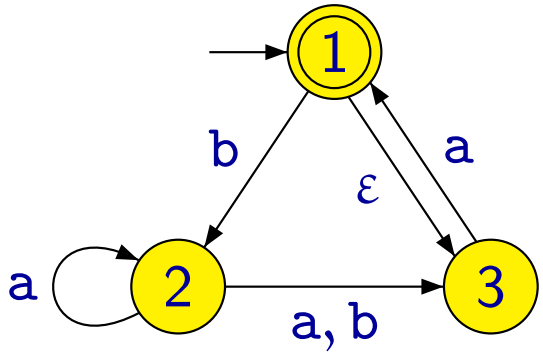

a
{1, 3}

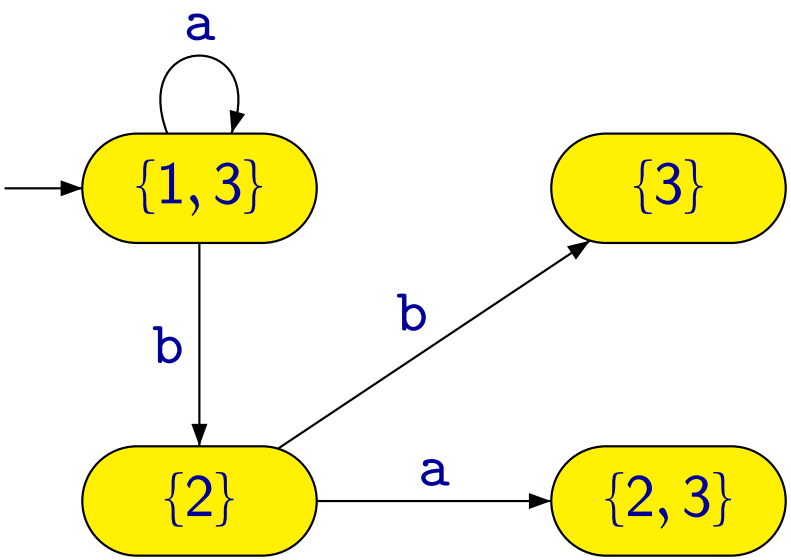

1
b
a
ε
a
2
3
a, b

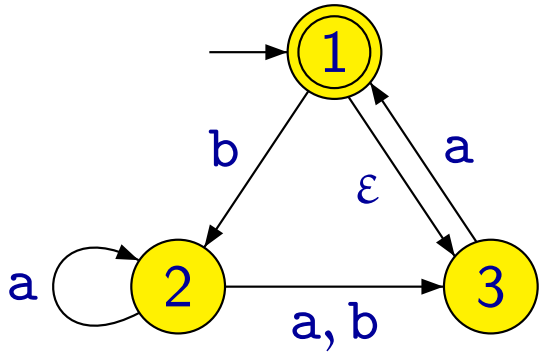

a
{1, 3}
b
{2}

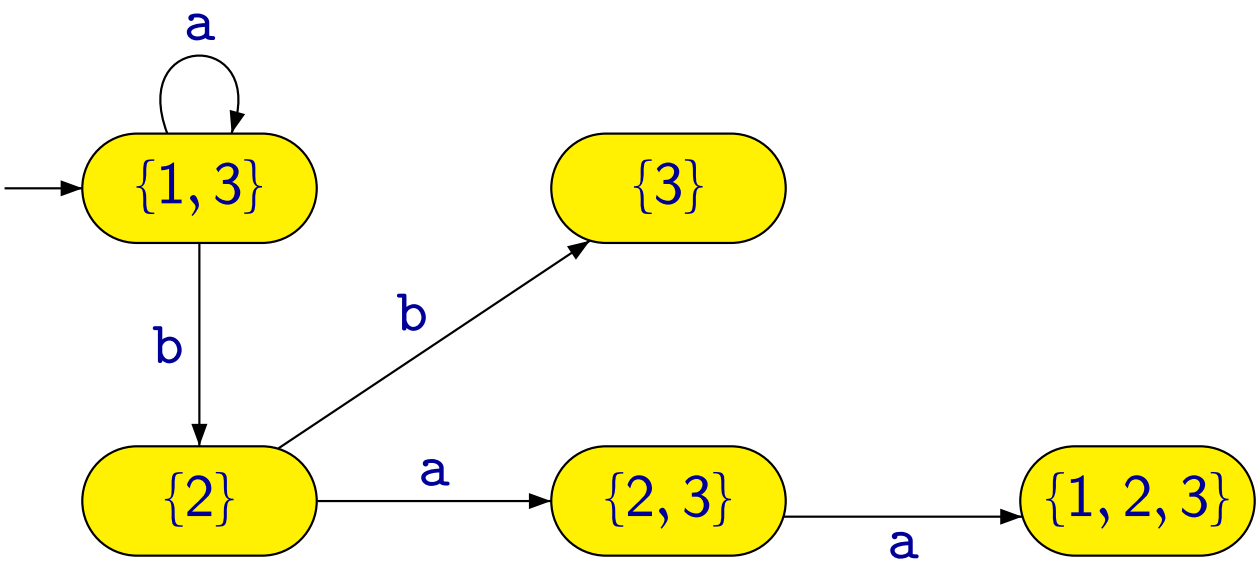

1
b
ε
a
a
2
3
a, b

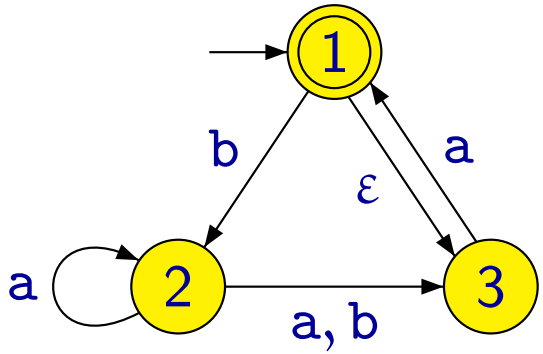

a
{1, 3}
b
{2}
a
{2, 3}

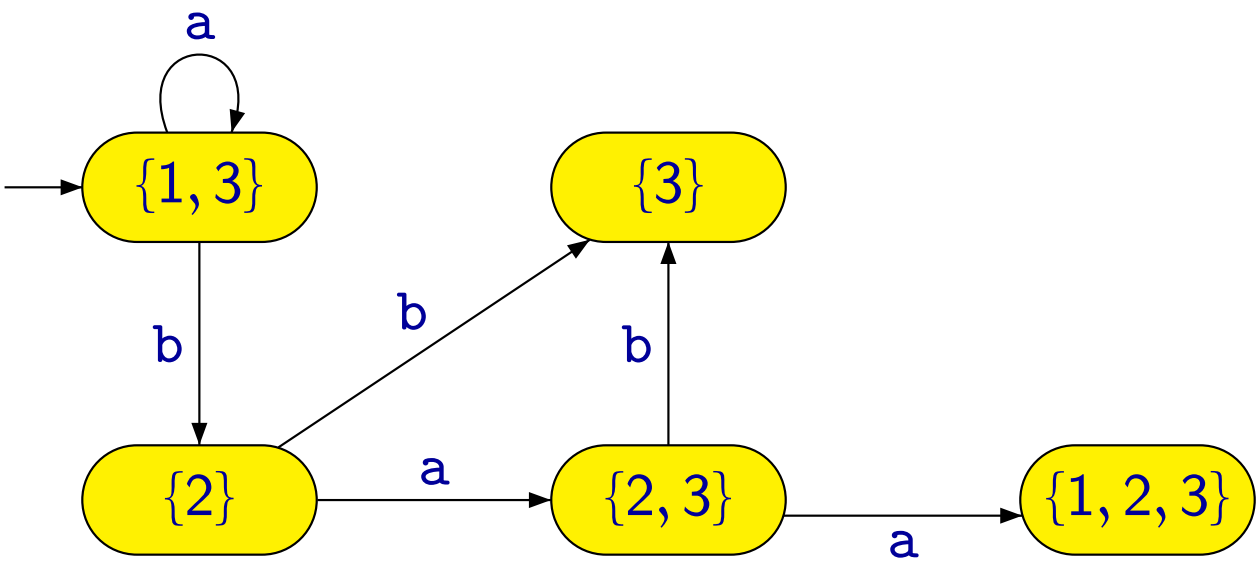

1
b
ε
a
a
2
3
a, b

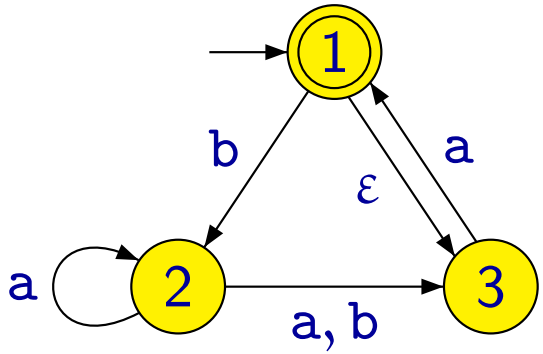

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}

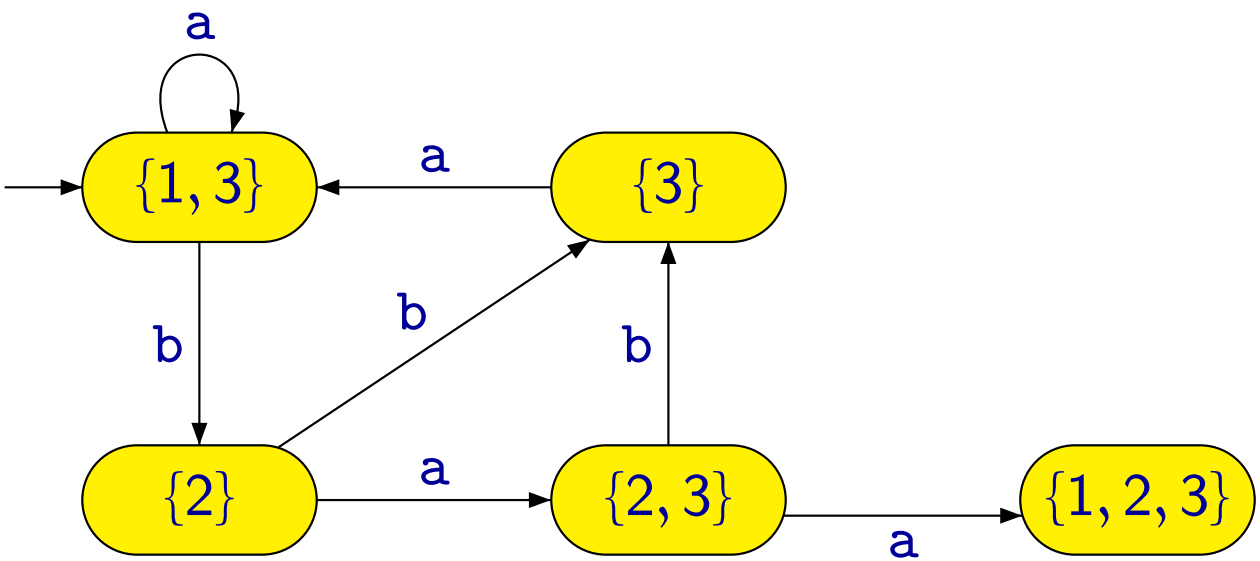

1
b
a
ε
a
2
3
a, b

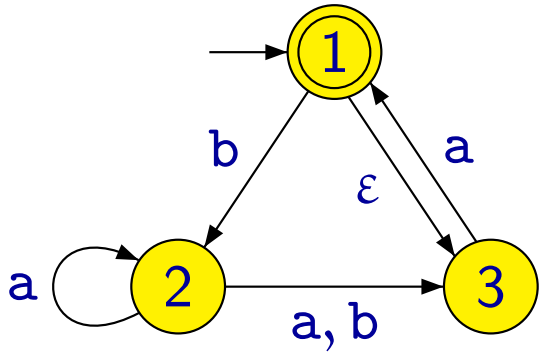

a
{1, 3}
{3}
b
b
{2}
a
{2, 3}
{1, 2, 3}
a

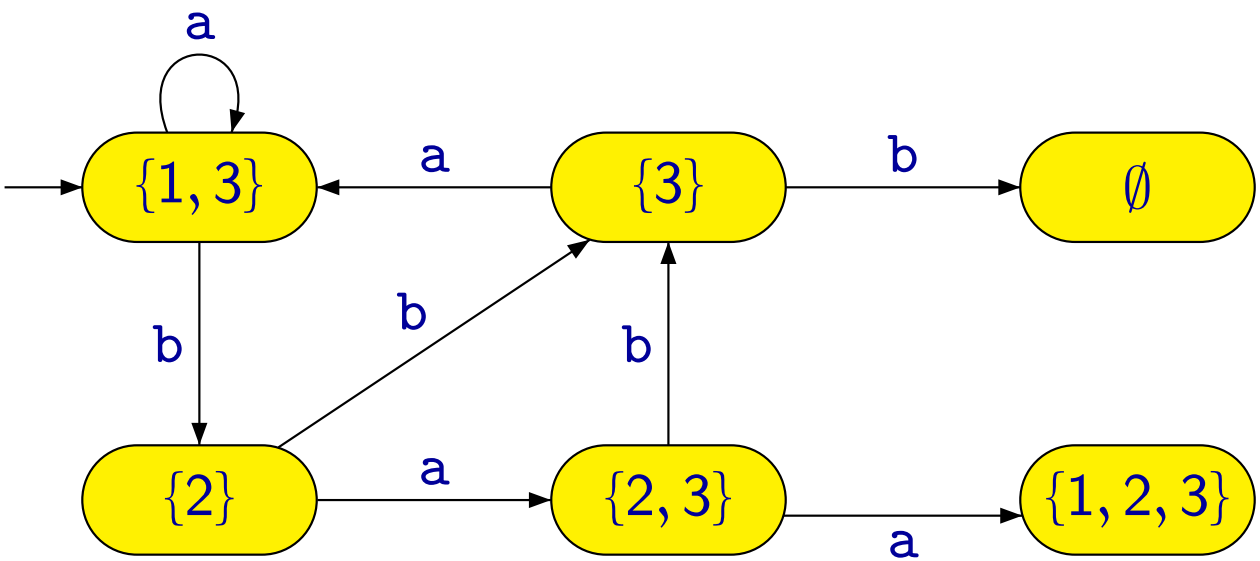

1
b
a
ε
a
2
3
a, b

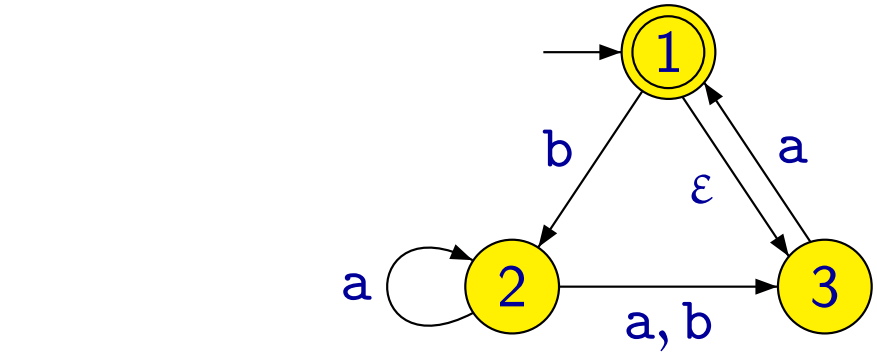

a
{1, 3}
{3}
b
b
b
{2}
a
{2, 3}
{1, 2, 3}
a

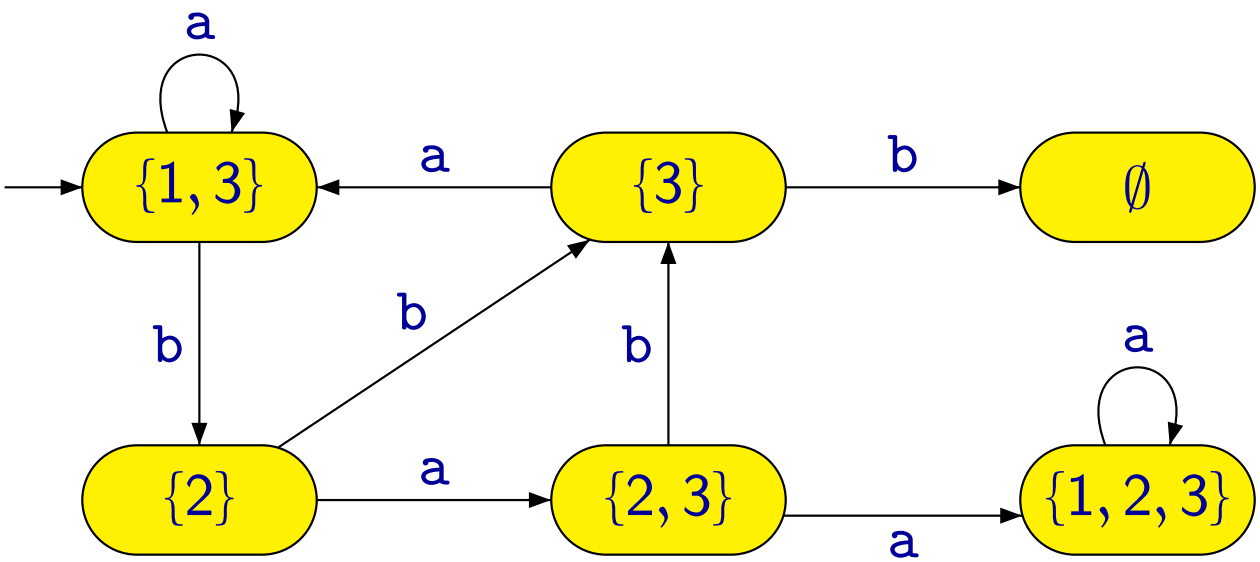

1
b
ε
a
a
2
3
a, b

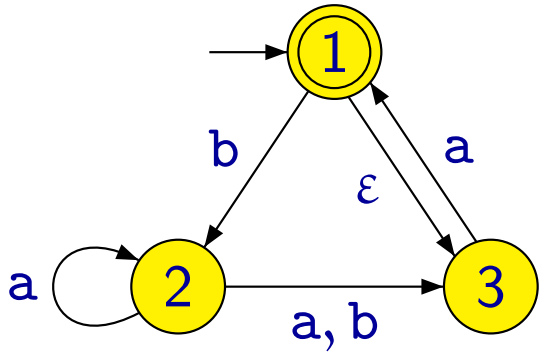

a
{1, 3}
a
{3}
b
b
b
{2}
a
{2, 3}
a
{1, 2, 3}

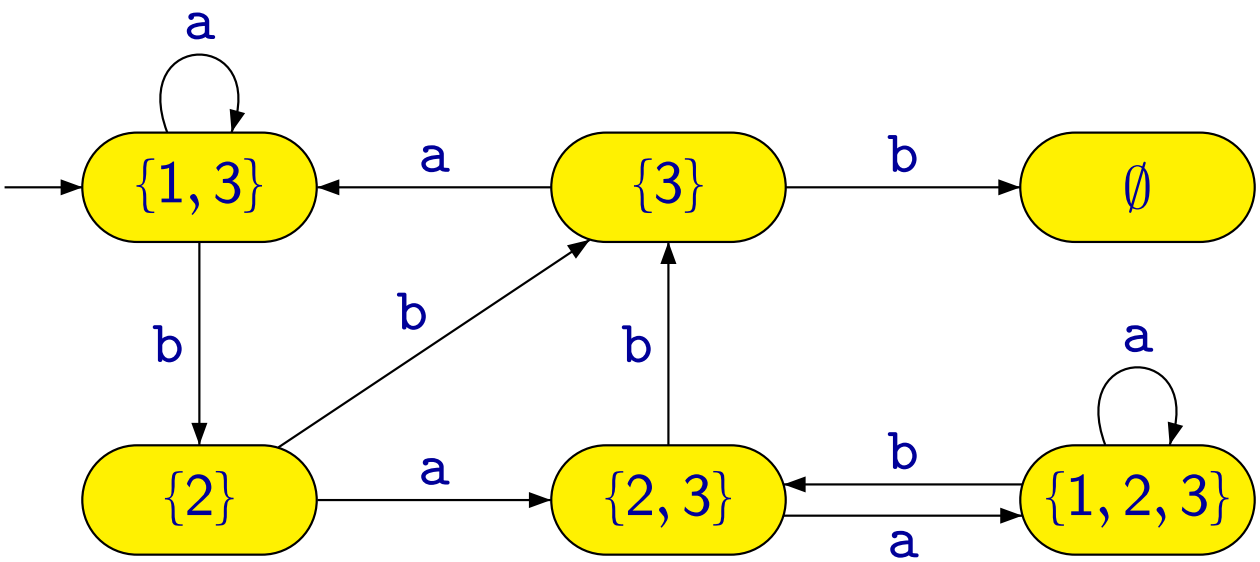

1
b
a
ε
a
2
3
a, b

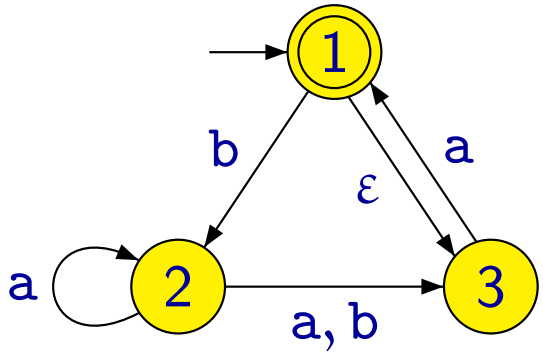

a
{1, 3}
a
{3}
b
∅
b
b
b
{2}
a
{2, 3}
{1, 2, 3}
a

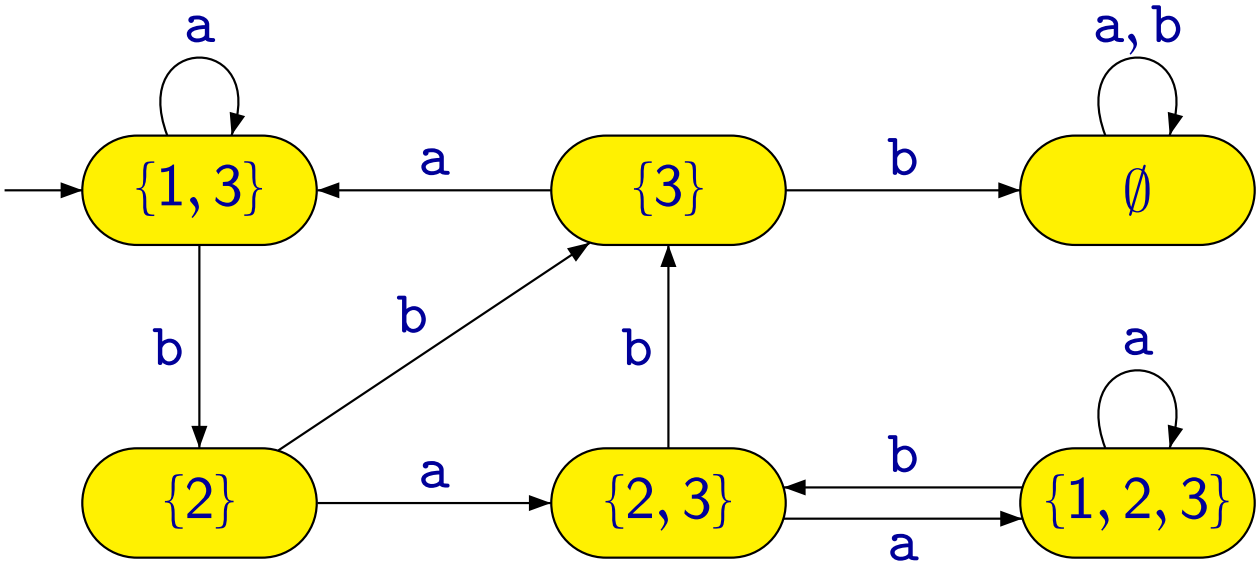

1
2
3
b
ε
a
a
a, b

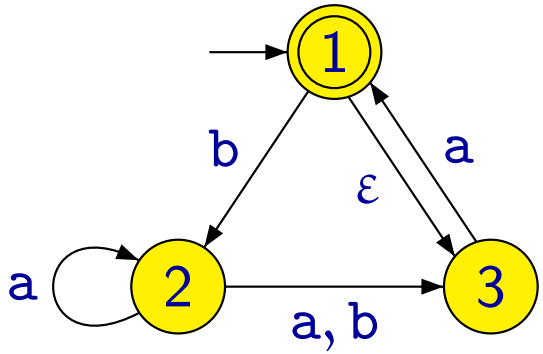

a
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
{1, 2, 3}
a

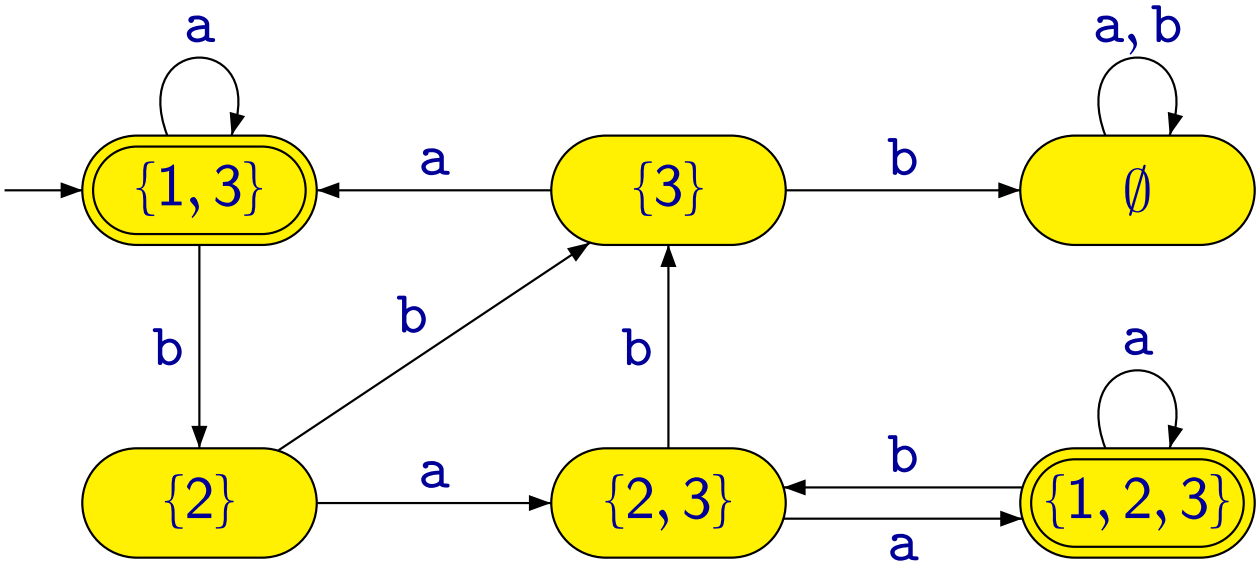

1
2
3
b
ε
a
a
a, b


a
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
a
{1, 2, 3}


1
2
3
b
ε
a
a
a, b


a
a, b
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
{1, 2, 3}
a


1
b
ε
a
a
2
3
a, b


a
a, b
{1, 3}
a
{3}
b
∅
b
b
b
a
{2}
a
{2, 3}
b
{1, 2, 3}
a

Převod ZNKA na DKA

Předtím, než formálně popíšeme převod ZNKA na DKA, zaveďme si několik pomocných definic.

Předpokládejme nějaký daný ZNKA $A = (Q, \Sigma, \delta, I, F)$.

Definujme funkci $\hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ tak, že pro $K \subseteq Q$ a $a \in \Sigma \cup \{\varepsilon\}$ je

$$\hat{\delta}(K, a) = \bigcup_{q \in K} \delta(q, a)$$

Převod ZNKA na DKA

Pro $K \subseteq Q$ označme $Cl_\varepsilon(K)$ množinu všech stavů dosažitelných ze stavů z množiny K nějakou libovolnou sekvencí ε-přechodů.

To znamená, že funkce $Cl_\varepsilon : \mathcal{P}(Q) \to \mathcal{P}(Q)$ je definována tak, že pro $K \subseteq Q$ je $Cl_\varepsilon(K)$ nejmenší (vzledem k inkluzi) množina splňující následující dvě podmínky:

- $K \subseteq Cl_\varepsilon(K)$
- Pro každé $q \in Cl_\varepsilon(K)$ platí, že $\delta(q, \varepsilon) \subseteq Cl_\varepsilon(K)$.

Poznámka: Všimněme si, že pro libovolné K je $Cl_\varepsilon(Cl_\varepsilon(K)) = Cl_\varepsilon(K)$.

Všimněme si také, že v případě NKA (kde $\delta(q, \varepsilon) = \emptyset$ pro každé $q \in Q$) je $Cl_\varepsilon(K) = K$.

Převod ZNKA na DKA

K danému ZNKA $A = (Q, \Sigma, \delta, I, F)$ nyní můžeme sestrojit DKA $A' = (Q', \Sigma, \delta', q_0', F')$, kde:

- $Q' = \mathcal{P}(Q)$ ($K \in Q'$ tedy znamená, že $K \subseteq Q$)
- $\delta' : Q' \times \Sigma \to Q'$ je definová tak, že pro $K \in Q'$ a $a \in \Sigma$ je

$$\delta'(K, a) = Cl_\varepsilon(\hat{\delta}(Cl_\varepsilon(K), a))$$

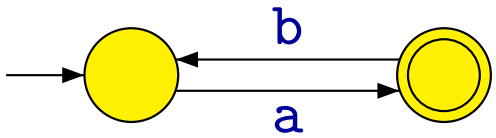
- $q_0' = Cl_\varepsilon(I)$
- $F' = \{K \in Q' \mid Cl_\varepsilon(K) \cap F \neq \emptyset\}$

Není težké ověřit, že $L(A) = L(A')$.

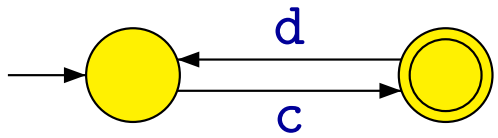
Zřetězení jazyků

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

A_1:



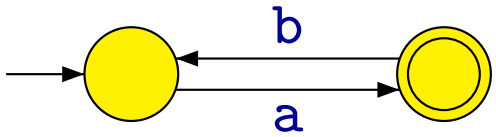
A_2:



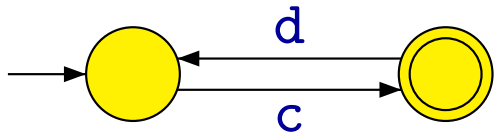
Zřetězení jazyků

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

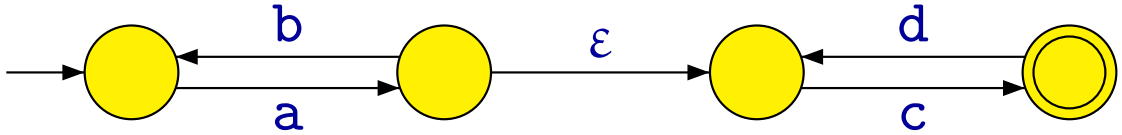
A_1:



A_2:



A:

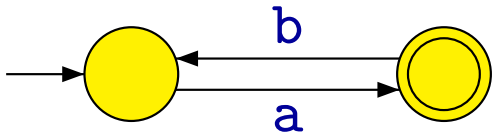


$L(A) = L(A_1) \cdot L(A_2)$

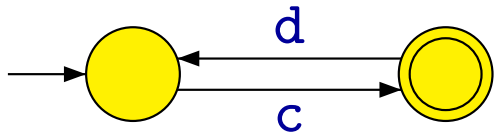
Zřetězení jazyků

$\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$

A_1:

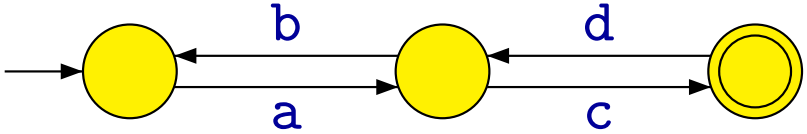


A_2:



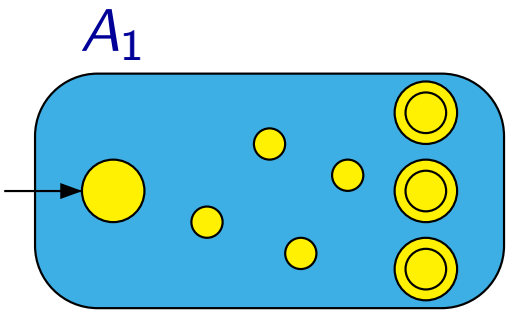
Chybná konstrukce:

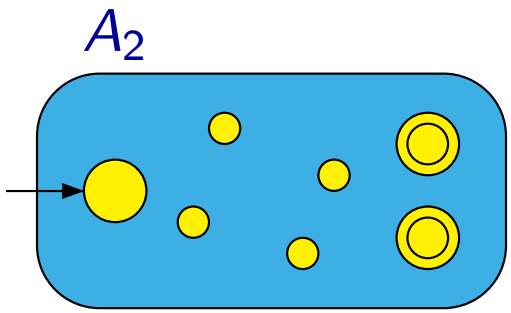
A:



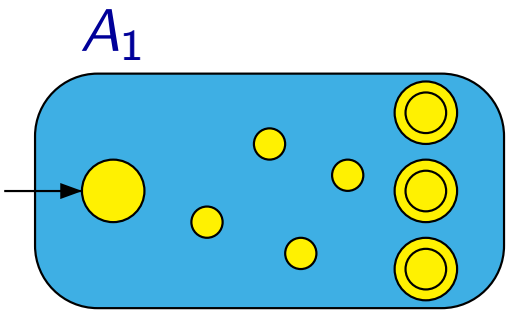
$\mathtt{acdbac} \in L(A)$, ale $\mathtt{acdbac} \notin L(A_1) \cdot L(A_2)$

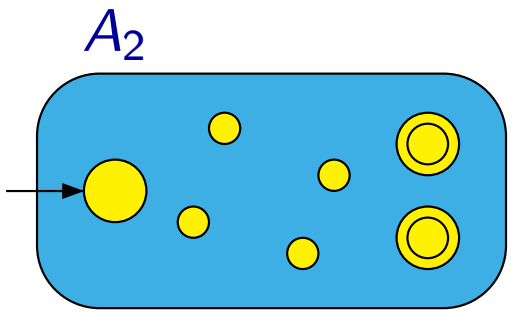
Zřetězení jazyků

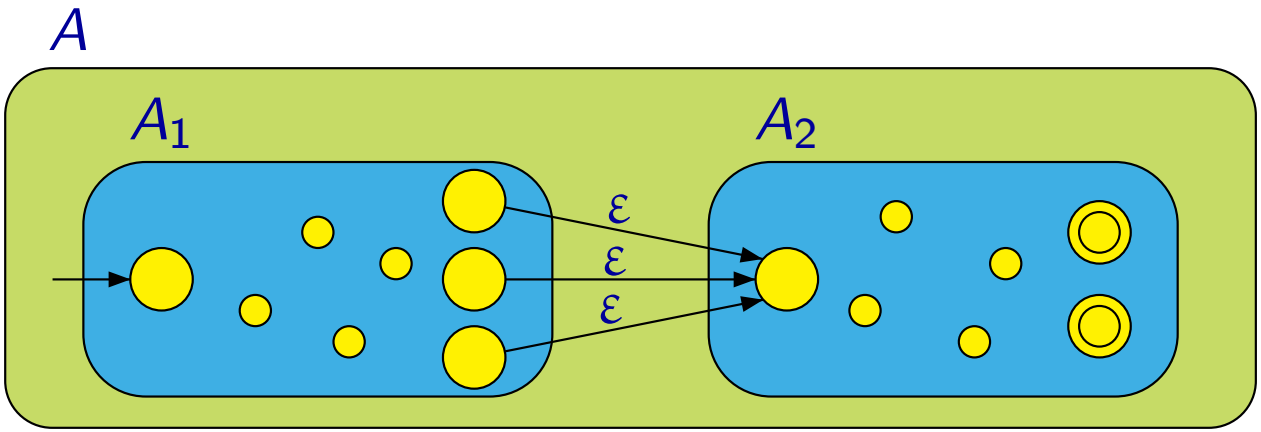




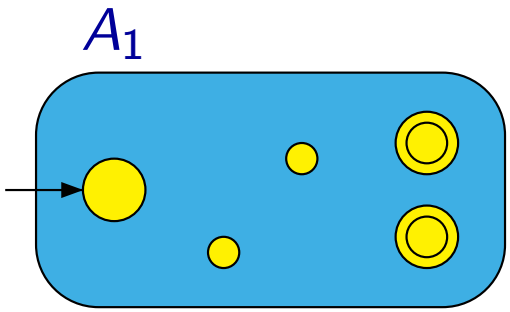
Zřetězení jazyků



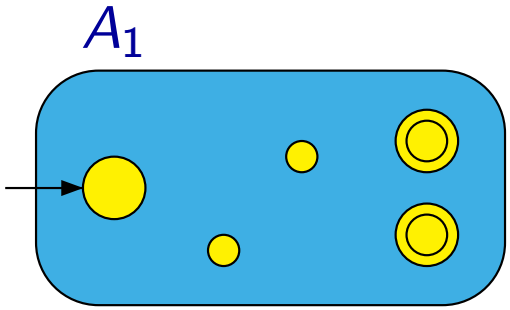


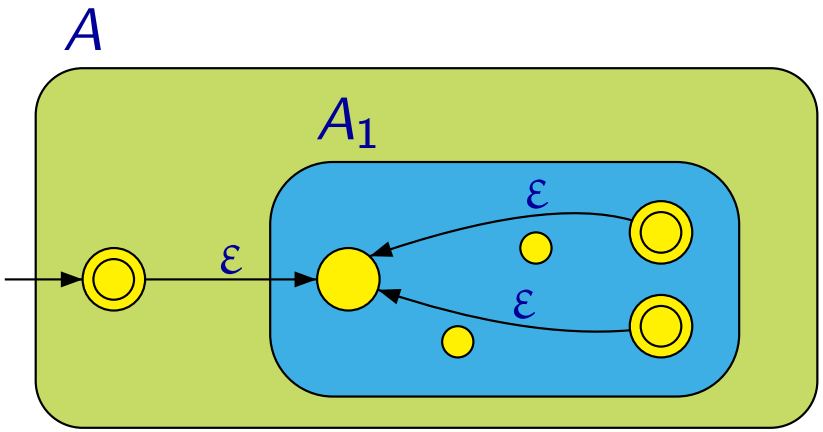


Iterace jazyka



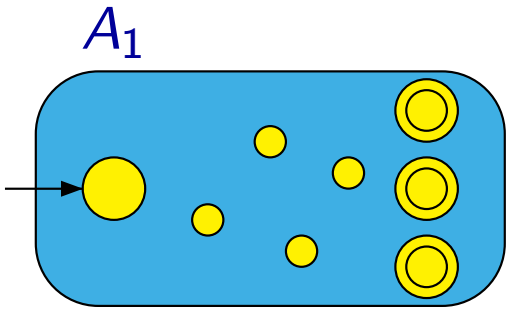
Iterace jazyka

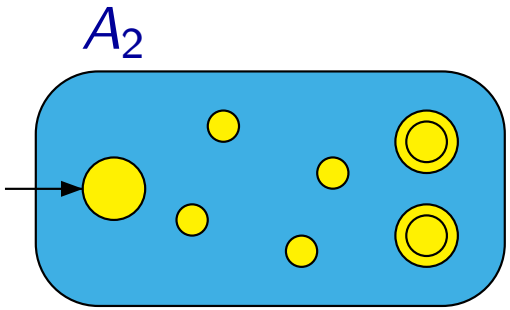




Sjednocení jazyků

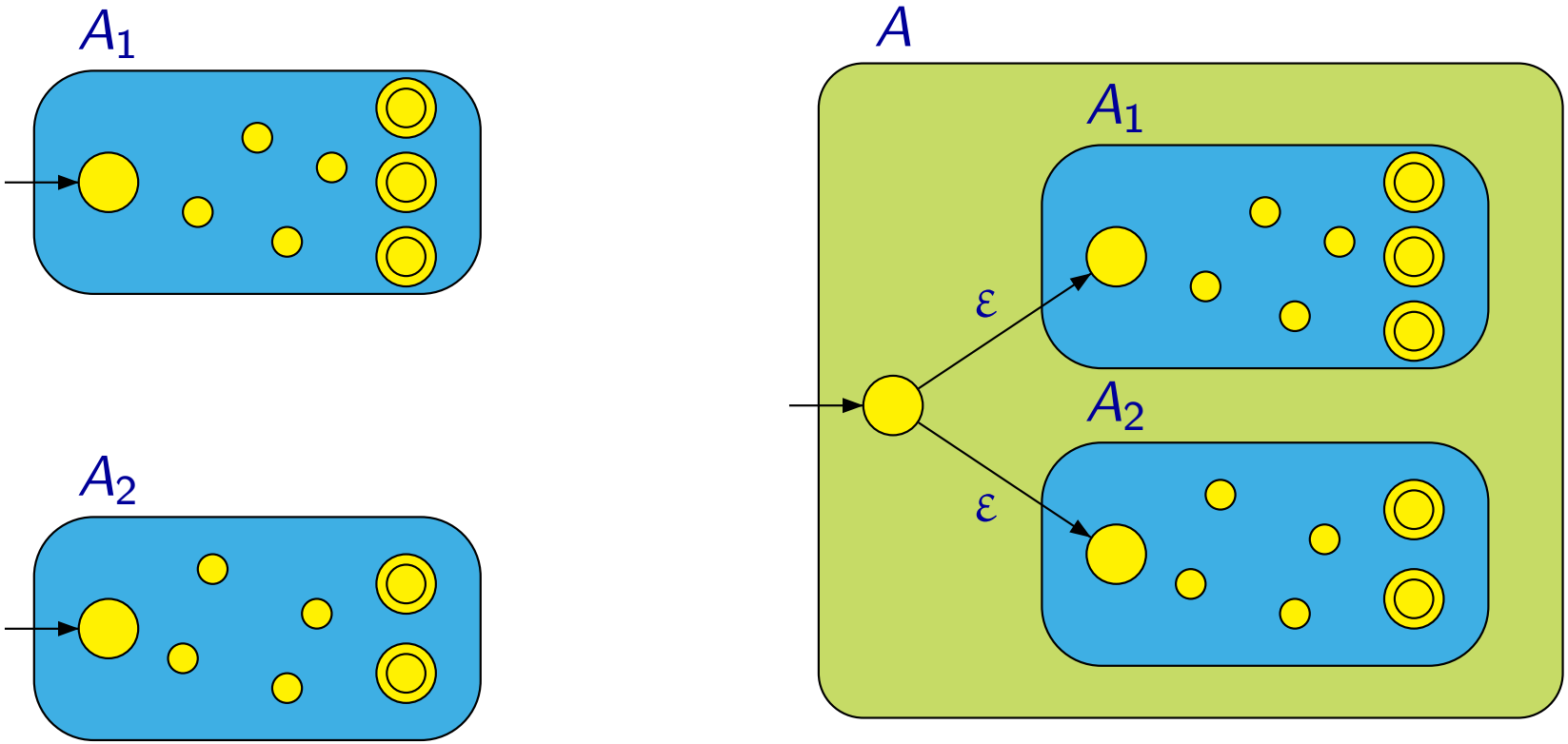
Alternativní konstrukce pro sjednocení jazyků:





Sjednocení jazyků

Alternativní konstrukce pro sjednocení jazyků:



Uzavřenost třídy regulárních jazyků

Množina (všech) regulárních jazyků je uzavřená vůči operacím:

- sjednocení
- průnik
- doplněk
- zřetězení
- iterace
- ...

Převod regulárního výrazu na konečný automat

Tvrzení

Každý jazyk, který je možné vyjádřit regulárním výrazem, je regulární (tj. rozpoznávaný nějakým konečným automatem).

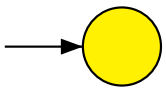
Důkaz: Stačí ukázat, jak k danému regulárnímu výrazu α zkonstruovat konečný automat, který rozpoznává jazyk $[\alpha]$.

Konstrukce je rekurzivní a postupuje podle struktury výrazu α:

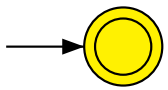
- Pokud je α elementární výraz (tj. $\emptyset$, ε nebo a):
  - Sestrojíme přímo odpovídající automat.
- Pokud je α tvaru $(\beta + \gamma)$, $(\beta \cdot \gamma)$ nebo (β^*):
  - Rekurzivně sestrojíme automaty rozpoznávající jazyky $[\beta]$ a $[\gamma]$.
  - Z nich sestrojíme automat rozpoznávající jazyk $[\alpha]$.

Převod regulárního výrazu na konečný automat

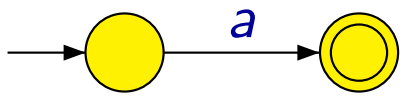
Automaty pro elementární výrazy:



$\emptyset$



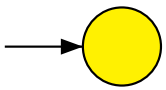
ε



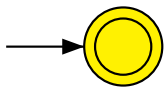
a

Převod regulárního výrazu na konečný automat

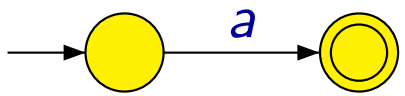
Automaty pro elementární výrazy:



$\emptyset$

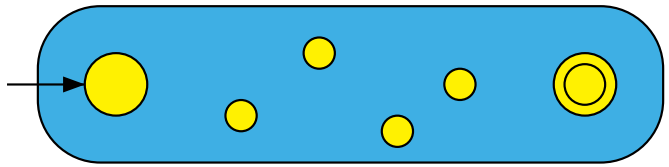


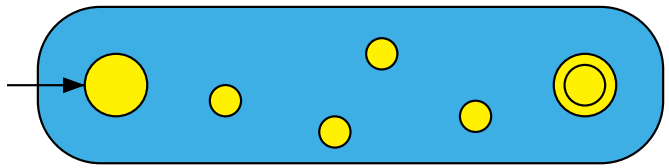
ε



a

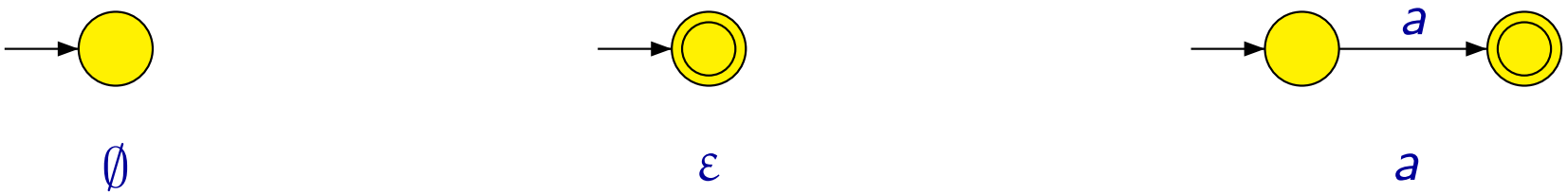
Konstrukce pro sjednocení:



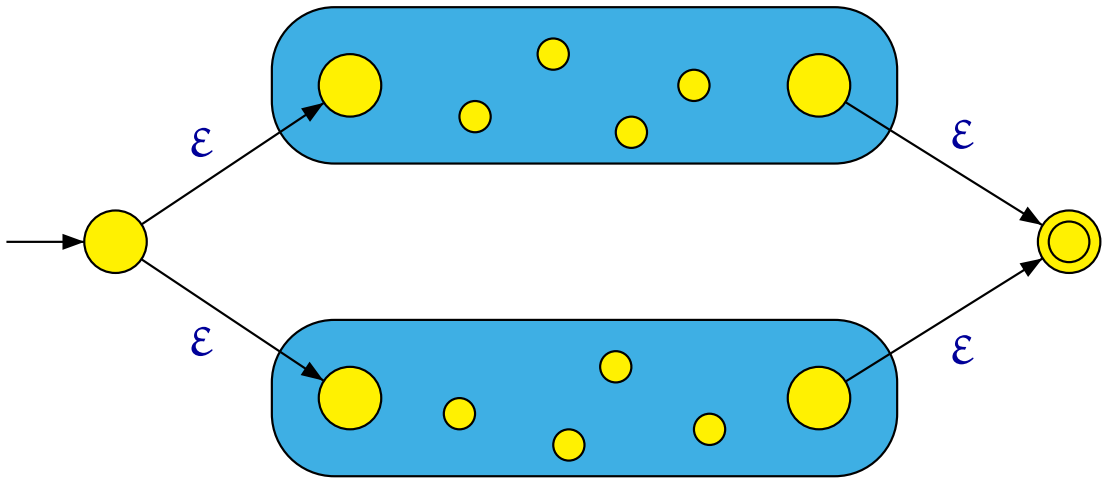


Převod regulárního výrazu na konečný automat

Automaty pro elementární výrazy:

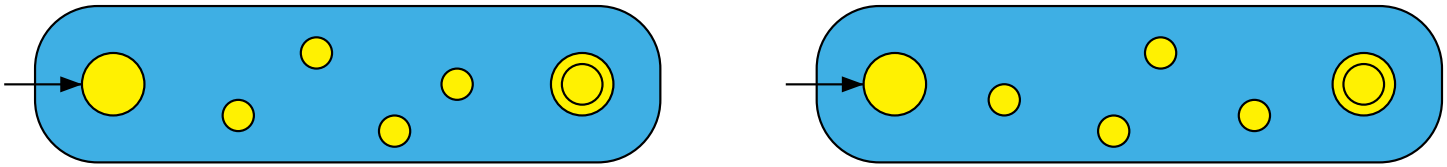


Konstrukce pro sjednocení:



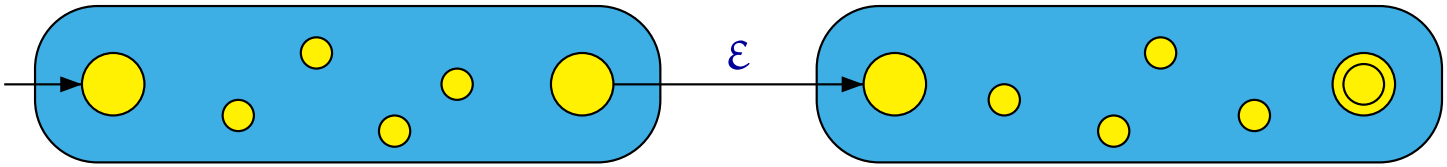
Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:



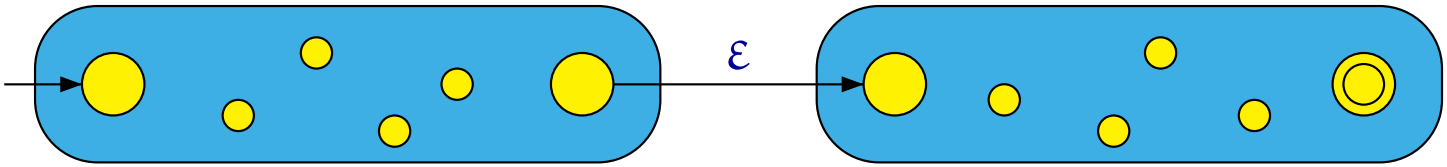
Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:

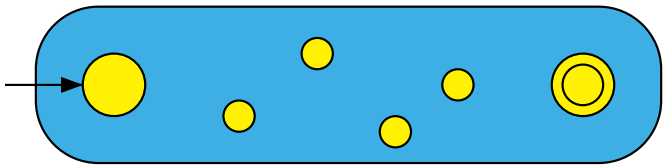


Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:

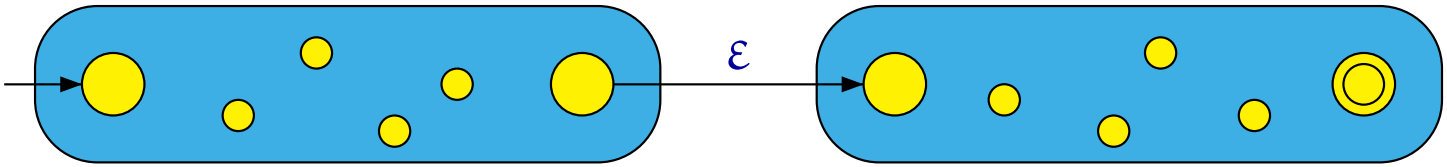


Konstrukce pro iteraci:

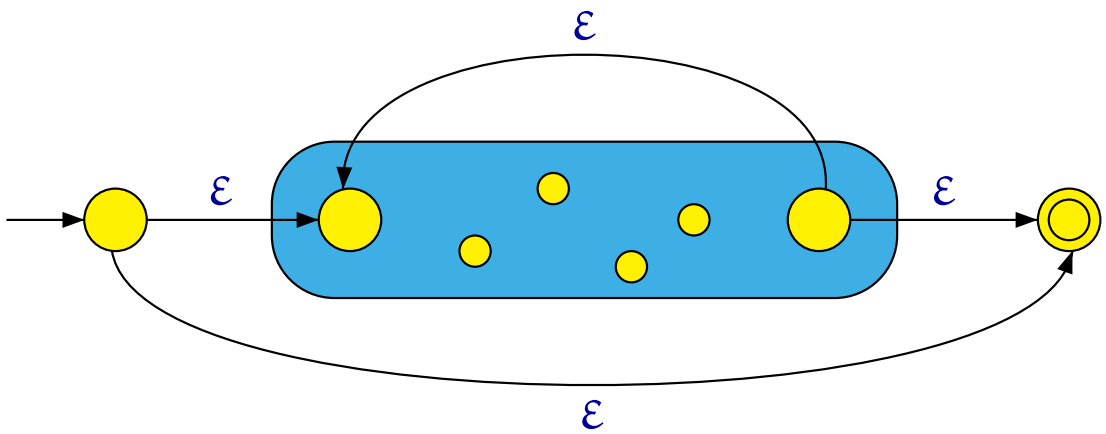


Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:



Konstrukce pro iteraci:

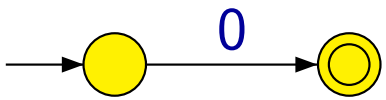


Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:

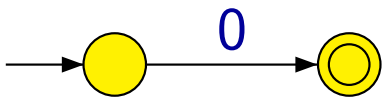
Převod regulárního výrazu na konečný automat

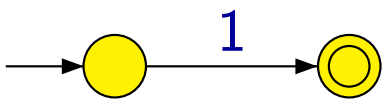
Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:



Převod regulárního výrazu na konečný automat

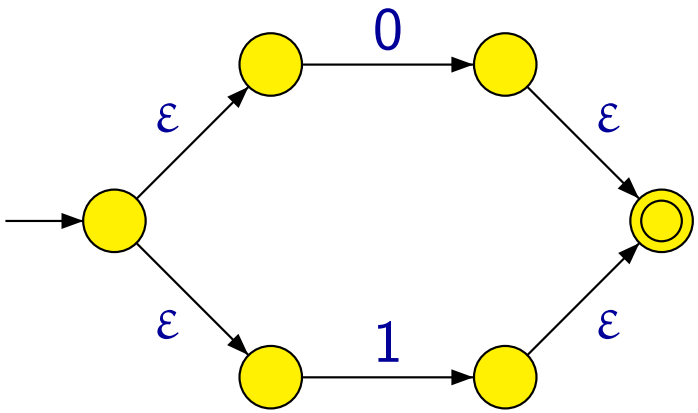
Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:





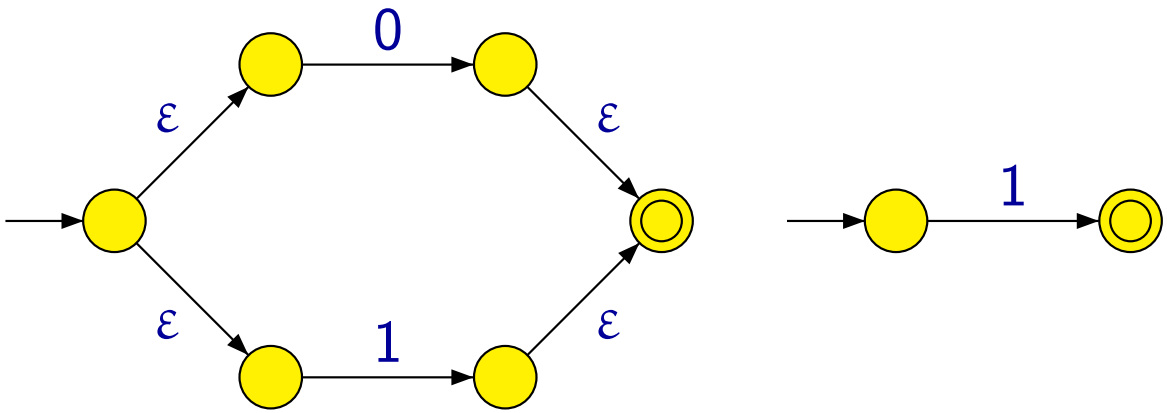
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:



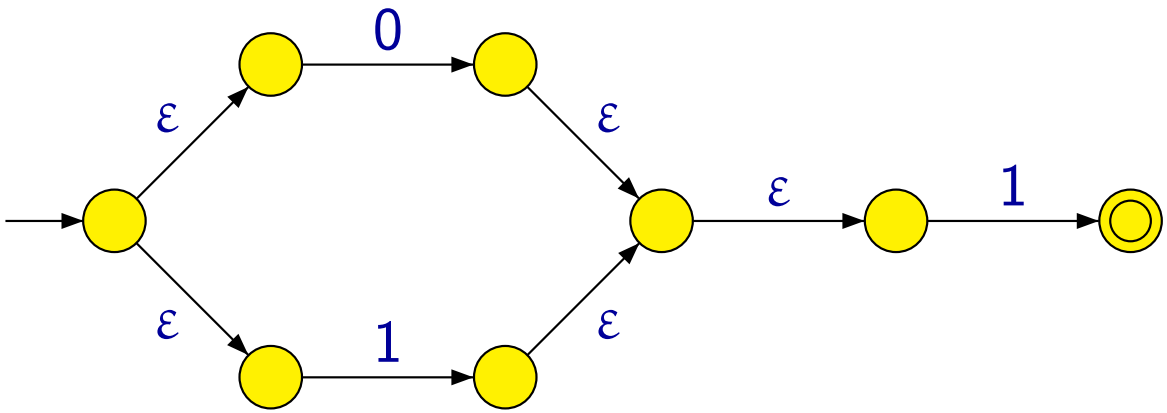
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:



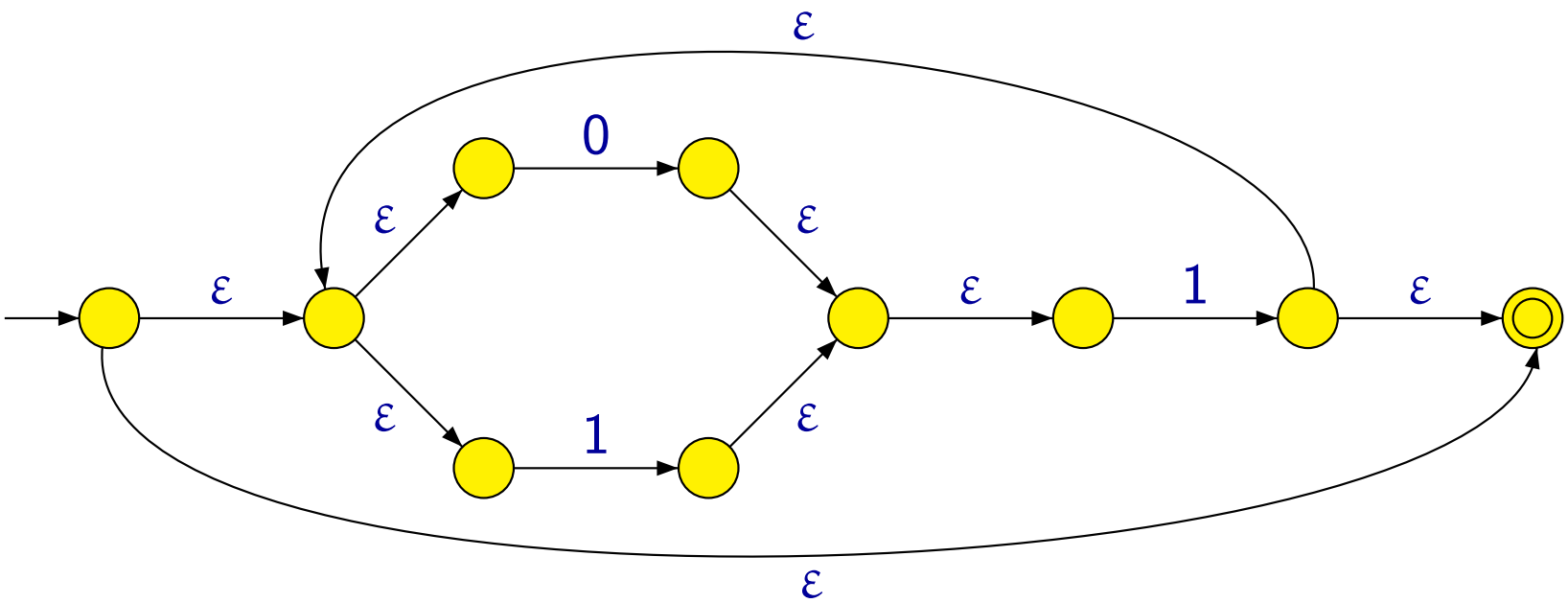
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:



Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0+1)\cdot 1)^*$:



Převod regulárního výrazu na konečný automat

Pokud se výraz α skládá z n znaků (nepočítáme-li závorky), má výsledný automat:

- nejvýše $2n$ stavů,
- nejvýše $4n$ přechodů.

Poznámka: Převodem ze zobecněného nedeterministického automatu na deterministický však může počet stavů vzrůst exponenciálně, tj. výsledný automat pak může mít až $2^{2n} = 4^n$ stavů.

Převod konečného automatu na regulární výraz

Tvrzení

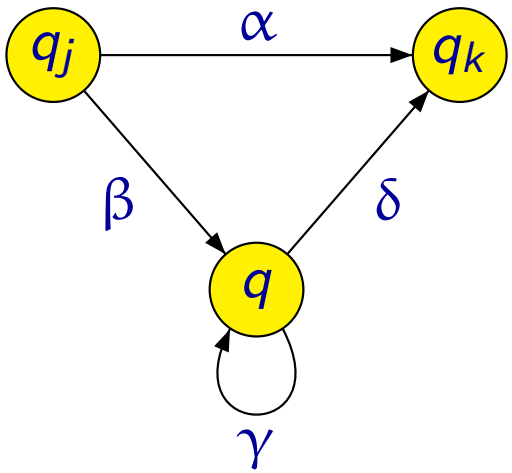
Každý regulární jazyk je možné popsat nějakým regulárním výrazem.

Důkaz: Stačí ukázat, jak pro libovolný konečný automat A zkonstruovat regulární výraz α takový, že $[\alpha] = L(A)$.

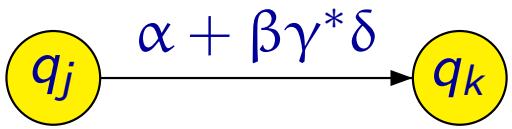
- A upravíme tak, aby měl právě jeden počáteční a právě jeden koncový stav.
- Budeme postupně odebírat jednotlivé stavy.
- Přechody budou označeny regulárními výrazy.
- Zbude automat se dvěma stavy – počátečním a koncovým, a jedním přechodem ohodnoceným výsledným regulárním výrazem.

Převod konečného automatu na regulární výraz

Hlavní myšlenka: Při odstraňování stavu q nahradit pro každou dvojici zbylých stavů q_j, q_k cestu z q_j do q_k vedoucí přes q.

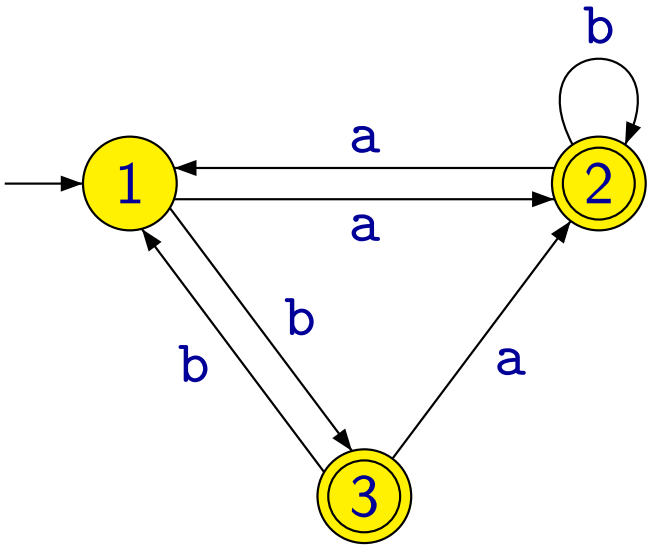


Po odstranění stavu q:



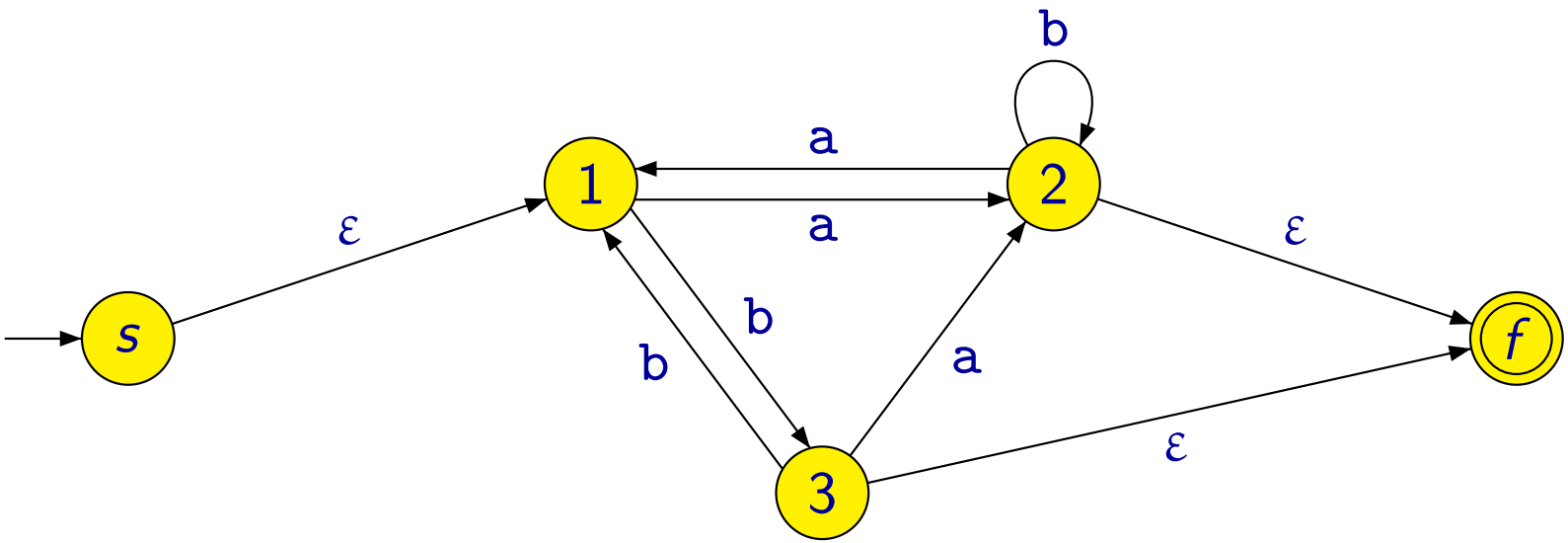
Převod konečného automatu na regulární výraz

Příklad:



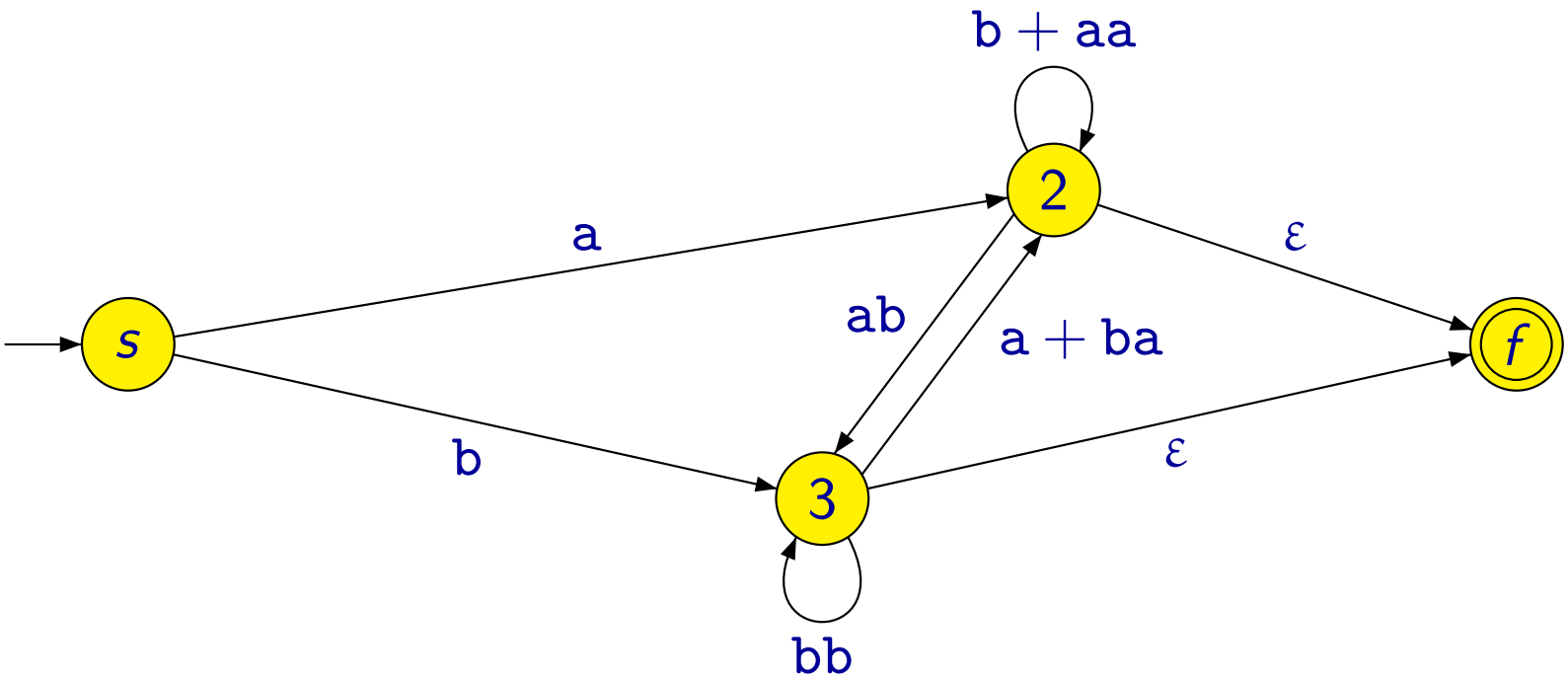
Převod konečného automatu na regulární výraz

Příklad:



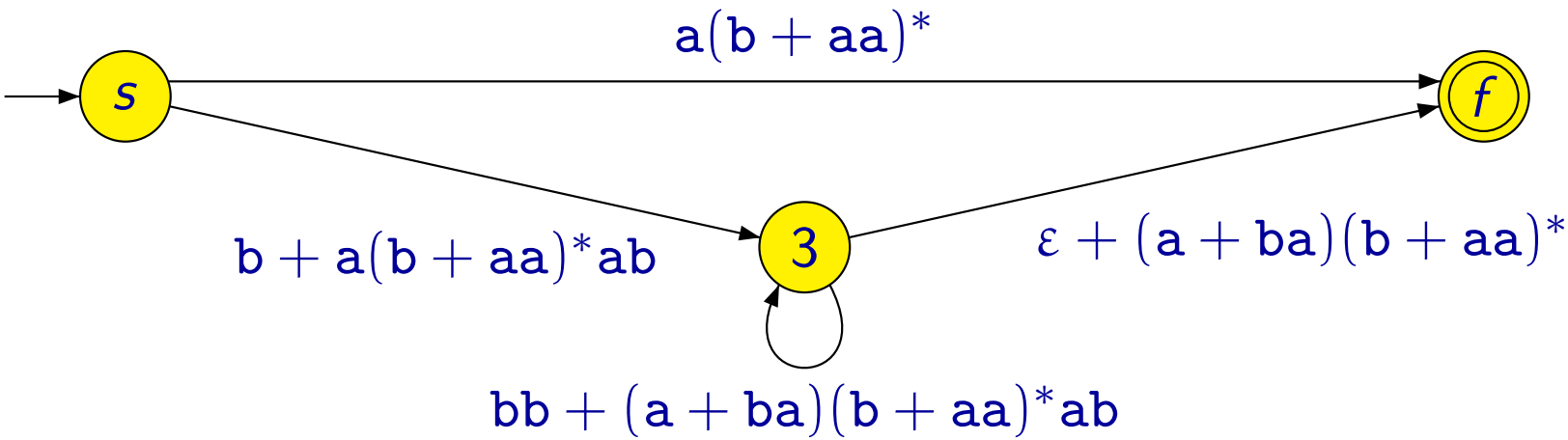
Převod konečného automatu na regulární výraz

Příklad:



Převod konečného automatu na regulární výraz

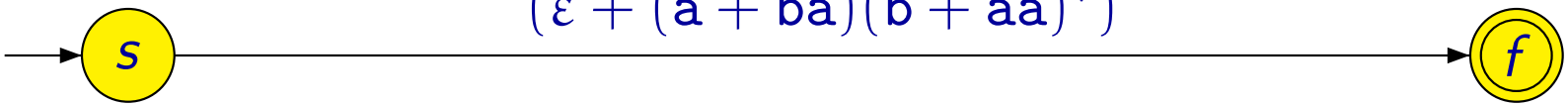
Příklad:



Převod konečného automatu na regulární výraz

Příklad:

$$\mathtt{a}(\mathtt{b}+\mathtt{aa})^*+$$
$$(\mathtt{b}+\mathtt{a}(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})$$
$$(\mathtt{bb}+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*\mathtt{ab})^*$$
$$(\varepsilon+(\mathtt{a}+\mathtt{ba})(\mathtt{b}+\mathtt{aa})^*)$$



Ekvivalence konečných automatů a regulárních výrazů

Věta

Jazyk je regulární právě tehdy, když je ho možné popsat regulárním výrazem.

Bezkontextové gramatiky

Bezkontextové gramatiky

Příklad: Chtěli bychom popsat jazyk aritmetických výrazů obsahující výrazy jako například:

`175` `(9+15)` `(((10-4)*((1+34)+2))/(3+(-37)))`

Pro jednoduchost předpokládejme:

- Výrazy jsou plně uzávorkované.
- Jediné aritmetické operace jsou "`+`", "`-`", "`*`", "`/`" a unární "`-`".
- Hodnoty operandů jsou přirozená čísla zapsaná v desítkové soustavě — zápis čísla je neprázdná posloupnost číslic.

Abeceda jazyka: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\}$

Bezkontextové gramatiky

Příklad (pokr.): Popis pomocí induktivní definice:

- **Číslice** je libovolný ze znaků 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- **Číslo** je neprázdná posloupnost číslic, tj.:
  - Pokud je α číslice, tak α je číslo.
  - Pokud α je číslice a β je číslo, tak i $\alpha\beta$ je číslo.
- **Výraz** je libovolná posloupnost symbolů vytvořená podle následujících pravidel:
  - Pokud je α číslo, tak α je výraz.
  - Pokud α je výraz, tak i $(-\alpha)$ je výraz.
  - Pokud α a β jsou výrazy, tak i $(\alpha+\beta)$ je výraz.
  - Pokud α a β jsou výrazy, tak i $(\alpha-\beta)$ je výraz.
  - Pokud α a β jsou výrazy, tak i $(\alpha*\beta)$ je výraz.
  - Pokud α a β jsou výrazy, tak i (α/β) je výraz.

Bezkontextové gramatiky

Příklad (pokr.): Způsob zápisu téže informace jako v předchozí induktivní definici pomocí **bezkontextové gramatiky**:

Zavedeme následující pomocné symboly — těmto symbolům se říká **neterminály**:

- D — zastupuje libovolnou číslici
- C — zastupuje libovolné číslo
- E — zastupuje libovolný výraz

$$D \to 0 \quad D \to 1 \quad D \to 2 \quad D \to 3 \quad D \to 4$$

$$D \to 5 \quad D \to 6 \quad D \to 7 \quad D \to 8 \quad D \to 9$$

$$C \to D \quad C \to DC$$

$$E \to C \quad E \to (-E) \quad E \to (E{+}E) \quad E \to (E{-}E) \quad E \to (E{*}E) \quad E \to (E/E)$$

Příklad (pokr.): Stručnější způsob zápisu:

$$D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$
$$C \to D \mid DC$$
$$E \to C \mid (-E) \mid (E+E) \mid (E-E) \mid (E*E) \mid (E/E)$$

Bezkontextové gramatiky

Příklad: Jazyk, kde slova jsou (případně i prázdné) posloupnosti výrazů popsaných v předchozím příkladě, kde jednotlivé výrazy jsou odděleny čárkami (abecedu je třeba rozšířit o symbol “,”):

$$
\begin{aligned}
S &\to T \mid \varepsilon \\
T &\to E \mid E\,,T \\
D &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
C &\to D \mid DC \\
E &\to C \mid (-E) \mid (E{+}E) \mid (E{-}E) \mid (E{*}E) \mid (E/E)
\end{aligned}
$$

Bezkontextové gramatiky

Příklad: Příkazy nějakého programovacího jazyka (fragment gramatiky):

$$
\begin{aligned}
S &\to E\texttt{;} \mid T \mid \texttt{if (}E\texttt{)}\ S \mid \texttt{if (}E\texttt{)}\ S\ \texttt{else}\ S \\
&\quad \mid \texttt{while (}E\texttt{)}\ S \mid \texttt{do}\ S\ \texttt{while (}E\texttt{);} \mid \texttt{for (}F\texttt{;}\,F\texttt{;}\,F\texttt{)}\ S \\
&\quad \mid \texttt{return}\ F\texttt{;} \\
T &\to \texttt{\{}\ U\ \texttt{\}} \\
U &\to \varepsilon \mid SU \\
F &\to \varepsilon \mid E \\
E &\to \quad \ldots \\
&\qquad \ldots
\end{aligned}
$$

Poznámka:

- S — příkaz
- T — blok příkazů
- U — sekvence příkazů
- E — výraz
- F — výraz, který je možno vynechat

Bezkontextové gramatiky

Formálně je **bezkontextová gramatika** definována jako čtveřice

$$G = (\Pi, \Sigma, S, P)$$

kde:

- Π je konečná množina **neterminálních symbolů** (**neterminálů**)
- Σ je konečná množina **terminálních symbolů** (**terminálů**), přičemž $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ je **počáteční neterminál**
- $P \subseteq \Pi \times (\Pi \cup \Sigma)^*$ je konečná množina **přepisovacích pravidel**

Bezkontextové gramatiky

Poznámky:

- Pro označení neterminálních symbolů budeme používat velká písmena A, B, C, ...
- Pro označení terminálních symbolů budeme používat malá písmena a, b, c, ... nebo číslice 0, 1, 2, ...
- Pro označení řetězců z $(\Pi \cup \Sigma)^*$ budeme používat malá písmena řecké abecedy α, β, γ, ...
- Místo zápisu (A, α) budeme pro pravidla používat zápis

$$A \to \alpha$$

A – levá strana pravidla
α – pravá strana pravidla

Bezkontextové gramatiky

Příklad: Gramatika $G = (\Pi, \Sigma, S, P)$, kde

- $\Pi = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $S = A$
- P obsahuje pravidla

$$A \to aBBb$$
$$A \to AaA$$
$$B \to \varepsilon$$
$$B \to bCA$$
$$C \to AB$$
$$C \to a$$
$$C \to b$$

Bezkontextové gramatiky

Poznámka: Pokud máme více pravidel se stejnou levou stranou, jako třeba

$$A \to \alpha_1 \qquad A \to \alpha_2 \qquad A \to \alpha_3$$

můžeme je stručněji zapsat jako

$$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$$

Například pravidla dříve uvedené gramatiky můžeme zapsat jako

$$\begin{aligned} A &\to aBBb \mid AaA \\ B &\to \varepsilon \mid bCA \\ C &\to AB \mid a \mid b \end{aligned}$$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

A

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$\underline{A}$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \to \underline{aBBb} \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$\underline{A} \Rightarrow \underline{aBBb}$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow a\underline{B}Bb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \to \underline{aBBb} \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$\underline{C} \to AB \mid a \mid \underline{b}$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$\underline{B} \to \underline{\varepsilon} \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb \Rightarrow abbabb$

Bezkontextové gramatiky

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \to aBBb \mid AaA$$
$$B \to \varepsilon \mid bCA$$
$$C \to AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$

Bezkontextové gramatiky

Na řetězcích z $(\Pi \cup \Sigma)^*$ definujeme relaci $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$ takovou, že

$$\alpha \Rightarrow \alpha'$$

právě když $\alpha = \beta_1 A \beta_2$ a $\alpha' = \beta_1 \gamma \beta_2$ pro nějaká $\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*$ a $A \in \Pi$, kde $(A \to \gamma) \in P$.

Příklad: Jestliže $(B \to bCA) \in P$, pak

$$aCBbA \Rightarrow aCbCAbA$$

Poznámka: Neformálně řečeno zápis $\alpha \Rightarrow \alpha'$ znamená, že z α je možné jedním krokem odvodit α', a to tak, že výskyt nějakého neterminálu A v α nahradíme pravou stranou nějakého pravidla $A \to \gamma$, kde se A vyskytuje na levé straně.

Bezkontextové gramatiky

Na řetězcích z $(\Pi \cup \Sigma)^*$ definujeme relaci $\Rightarrow \subseteq (\Pi \cup \Sigma)^* \times (\Pi \cup \Sigma)^*$ takovou, že

$$\alpha \Rightarrow \alpha'$$

právě když $\alpha = \beta_1 A \beta_2$ a $\alpha' = \beta_1 \gamma \beta_2$ pro nějaká $\beta_1, \beta_2, \gamma \in (\Pi \cup \Sigma)^*$ a $A \in \Pi$, kde $(A \to \gamma) \in P$.

Příklad: Jestliže $(B \to bCA) \in P$, pak

$$aC\underline{B}bA \Rightarrow aC\underline{bCA}bA$$

Poznámka: Neformálně řečeno zápis $\alpha \Rightarrow \alpha'$ znamená, že z α je možné jedním krokem odvodit α', a to tak, že výskyt nějakého neterminálu A v α nahradíme pravou stranou nějakého pravidla $A \to \gamma$, kde se A vyskytuje na levé straně.

Bezkontextové gramatiky

Derivace délky n je posloupnost $\beta_0, \beta_1, \beta_2, \cdots, \beta_n$, kde $\beta_i \in (\Pi \cup \Sigma)^*$ a kde $\beta_{i-1} \Rightarrow \beta_i$ pro všechna $1 \leq i \leq n$, což můžeme stručněji zapsat

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$$

Skutečnost, že pro dané $\alpha, \alpha' \in (\Pi \cup \Sigma)^*$ a $n \in \mathbb{N}$ existuje nějaká derivace $\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \ldots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$, kde $\alpha = \beta_0$ a $\alpha' = \beta_n$, zapisujeme

$$\alpha \Rightarrow^n \alpha'$$

Skutečnost, že $\alpha \Rightarrow^n \alpha'$ pro nějaké $n \geq 0$, zapisujeme

$$\alpha \Rightarrow^* \alpha'$$

Poznámka: Relace $\Rightarrow^*$ je reflexivním a tranzitivním uzávěrem relace $\Rightarrow$ (tj. nejmenší reflexivní a tranzitivní relací obsahující relaci $\Rightarrow$).

Bezkontextové gramatiky

Větné formy jsou ty $\alpha \in (\Pi \cup \Sigma)^*$, pro které platí

$$S \Rightarrow^* \alpha$$

kde S je počáteční neterminál.

Bezkontextové gramatiky

Jazyk $L(G)$ generovaný gramatikou $G = (\Pi, \Sigma, S, P)$ je množina všech slov v abecedě Σ, která lze odvodit nějakou derivací z počátečního neterminálu S pomocí pravidel z P, tj.

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk

$$L = \{a^n b^n \mid n \geq 0\}$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk

$$L = \{a^n b^n \mid n \geq 0\}$$

Gramatika $G = (\Pi, \Sigma, S, P)$, kde $\Pi = \{S\}$, $\Sigma = \{a, b\}$ a P obsahuje

$$S \to aSb \mid \varepsilon$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk

$$L = \{a^n b^n \mid n \geq 0\}$$

Gramatika $G = (\Pi, \Sigma, S, P)$, kde $\Pi = \{S\}$, $\Sigma = \{a, b\}$ a P obsahuje

$$S \to aSb \mid \varepsilon$$

$$S \Rightarrow \varepsilon$$
$$S \Rightarrow aSb \Rightarrow ab$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$
$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$
$$\ldots$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk tvořený všemi palindromy nad abecedou $\{a, b\}$, tj.

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

Poznámka: w^R označuje tzv. **zrcadlový obraz** slova w, tj. slovo w zapsané pozpátku.

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk tvořený všemi palindromy nad abecedou $\{a, b\}$, tj.

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

Poznámka: w^R označuje tzv. **zrcadlový obraz** slova w, tj. slovo w zapsané pozpátku.

Řešení:

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk tvořený všemi palindromy nad abecedou $\{a, b\}$, tj.

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

Poznámka: w^R označuje tzv. **zrcadlový obraz** slova w, tj. slovo w zapsané pozpátku.

Řešení:

$$S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.

Například $(()())(()) \in L$, ale $)()) \notin L$.

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.

Například $(()())(()) \in L$, ale $)()) \notin L$.

Řešení:

$$S \to \varepsilon \mid (S) \mid SS$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.

Například $(()())(()) \in L$, ale $)()) \notin L$.

Řešení:

$$S \to \varepsilon \mid (S) \mid SS$$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow$
$(()S)(S) \Rightarrow (()(S))(S) \Rightarrow (()())(S) \Rightarrow (()())((S)) \Rightarrow$
$(()())(())$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře vytvořenými aritmetickými výrazy, kde operandy jsou vždy tvaru 'a', a kde jako operátory můžeme používat symboly $+$ a $*$.

Například $(a + a) * a + (a * a) \in L$.

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře vytvořenými aritmetickými výrazy, kde operandy jsou vždy tvaru 'a', a kde jako operátory můžeme používat symboly $+$ a $*$.

Například $(a + a) * a + (a * a) \in L$.

Řešení:

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$

Bezkontextové gramatiky

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře vytvořenými aritmetickými výrazy, kde operandy jsou vždy tvaru 'a', a kde jako operátory můžeme používat symboly $+$ a $*$.

Například $(a+a)*a+(a*a) \in L$.

Řešení:

$$E \to a \mid E+E \mid E*E \mid (E)$$

$$E \Rightarrow E+E \Rightarrow E*E+E \Rightarrow (E)*E+E \Rightarrow (E+E)*E+E \Rightarrow$$
$$(a+E)*E+E \Rightarrow (a+a)*E+E \Rightarrow (a+a)*a+E \Rightarrow (a+a)*a+(E) \Rightarrow$$
$$(a+a)*a+(E*E) \Rightarrow (a+a)*a+(a*E) \Rightarrow (a+a)*a+(a*a)$$

Derivační strom

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

Derivační strom

A

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

A

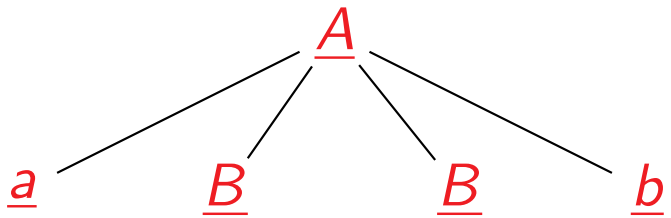
Derivační strom

$\underline{A}$

$\underline{A} \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$\underline{A}$

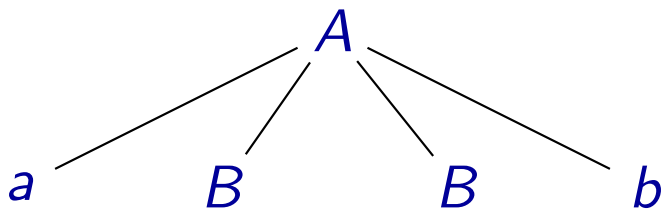
Derivační strom



$\underline{A} \to \underline{aBBb} \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$\underline{A} \Rightarrow \underline{aBBb}$

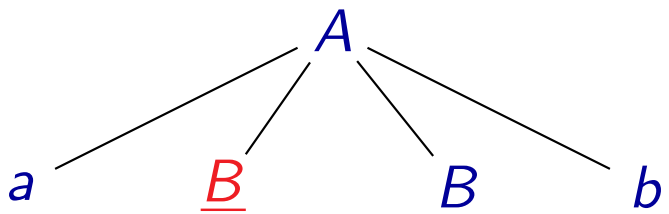
Derivační strom



$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$A \Rightarrow aBBb$

Derivační strom

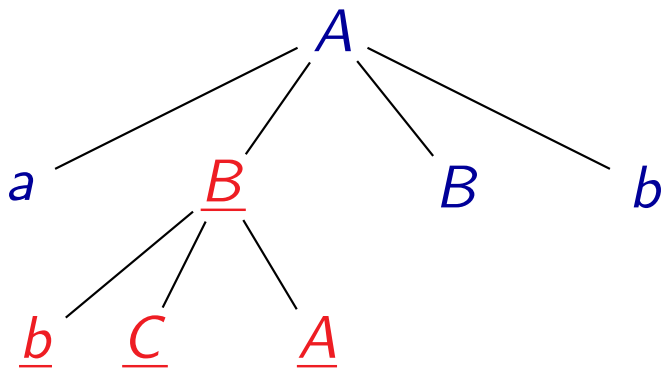


$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$

$A \Rightarrow a\underline{B}Bb$

Derivační strom

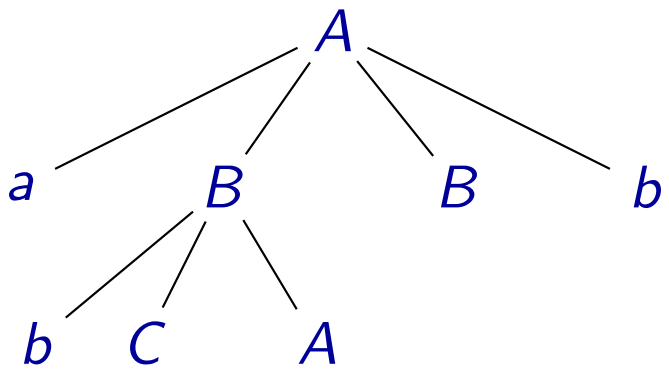
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid \underline{bCA}$
$C \to AB \mid a \mid b$



$A \Rightarrow a\underline{B}Bb \Rightarrow a\underline{bCA}Bb$

Derivační strom

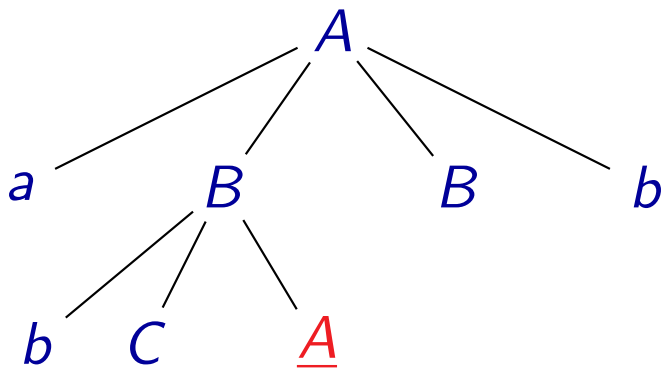
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb$

Derivační strom

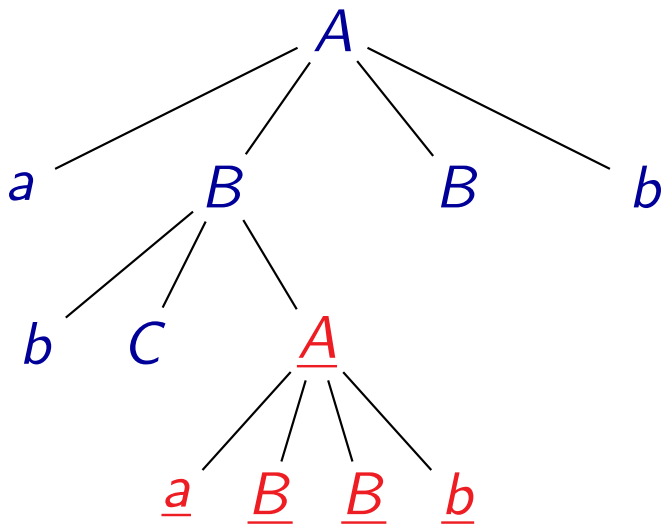
$\underline{A} \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$

Derivační strom

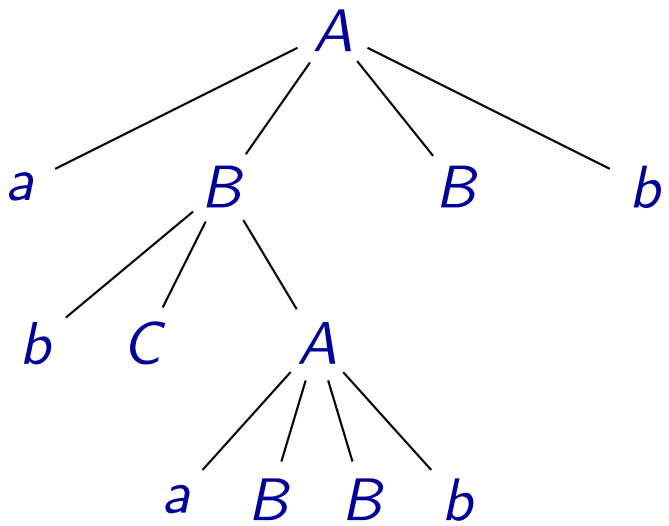
$\underline{A} \to \underline{aBBb} \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$

Derivační strom

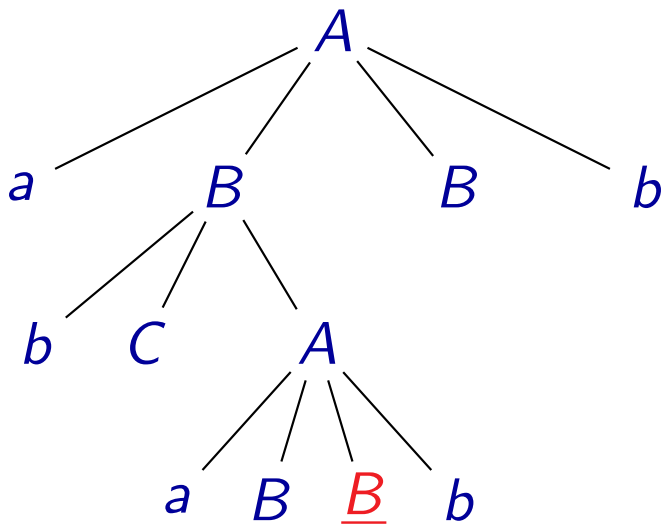
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$

Derivační strom

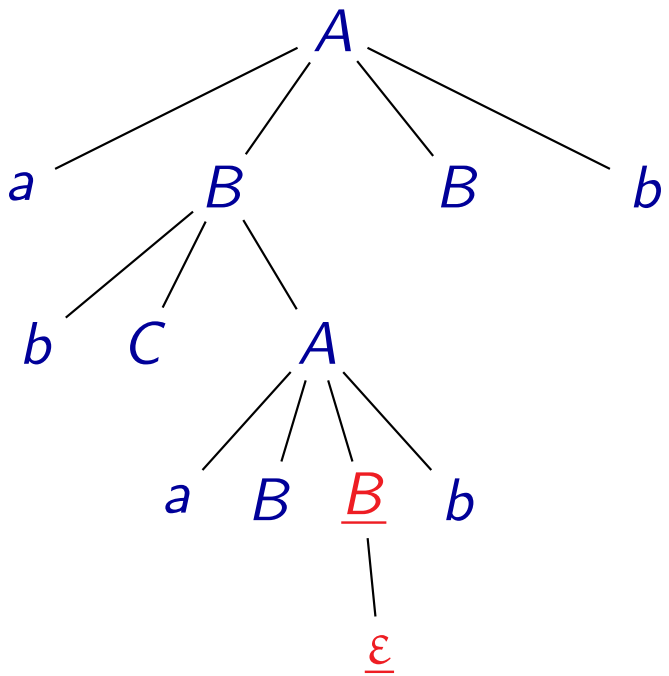
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb$

Derivační strom

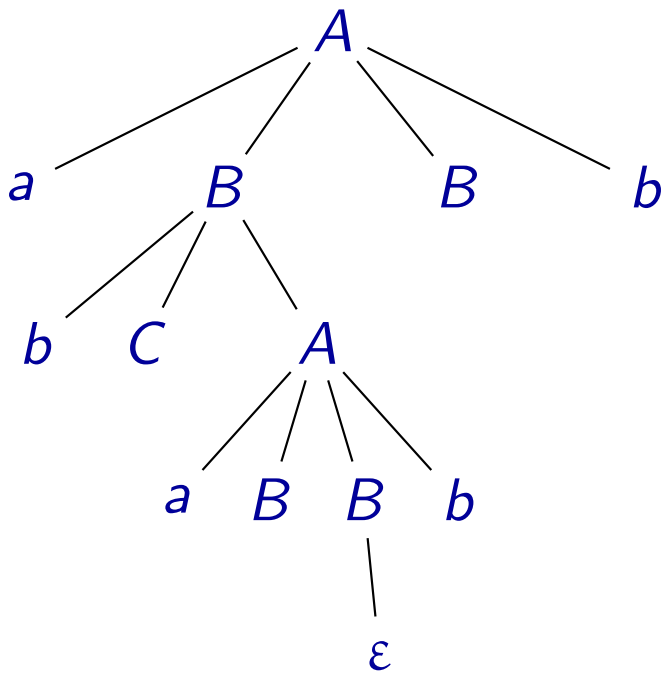
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$

Derivační strom

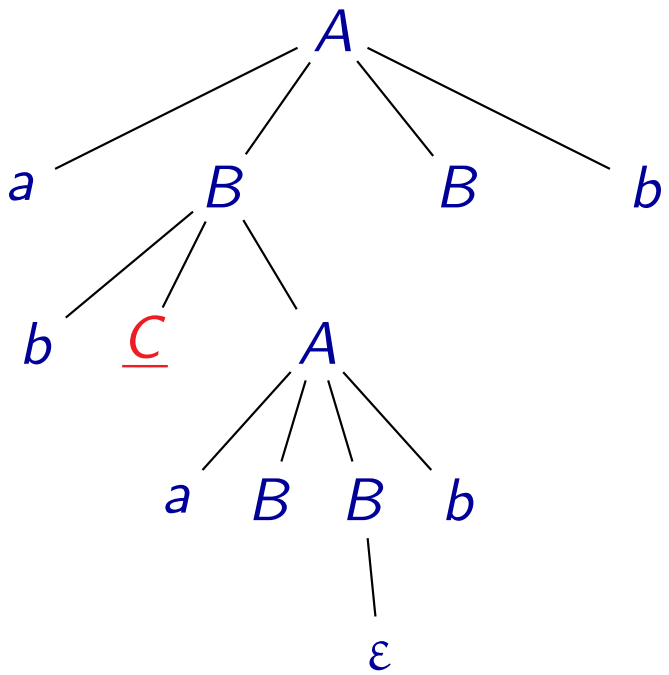
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$

Derivační strom

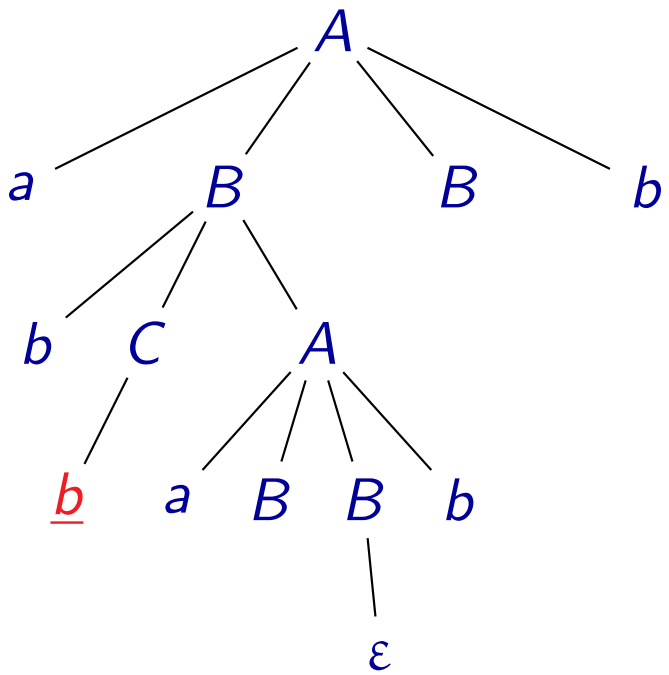
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$\underline{C} \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb$

Derivační strom

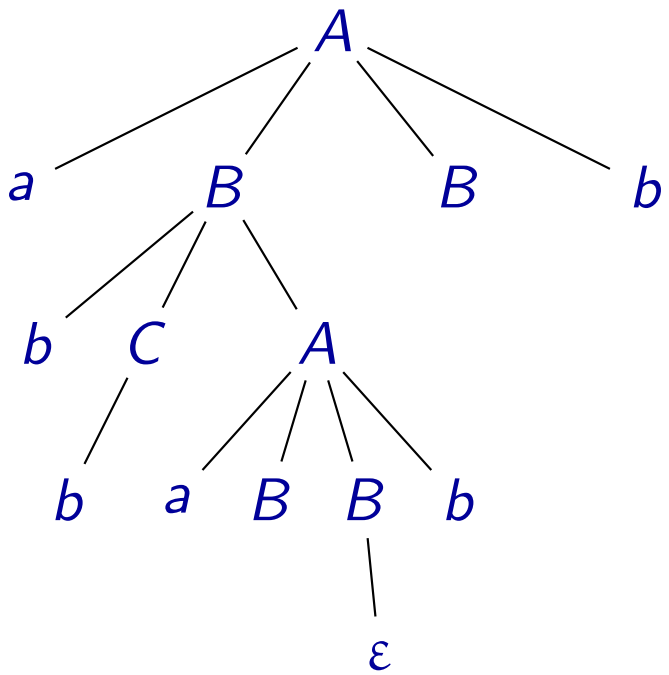
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$\underline{C} \to AB \mid a \mid \underline{b}$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$

Derivační strom

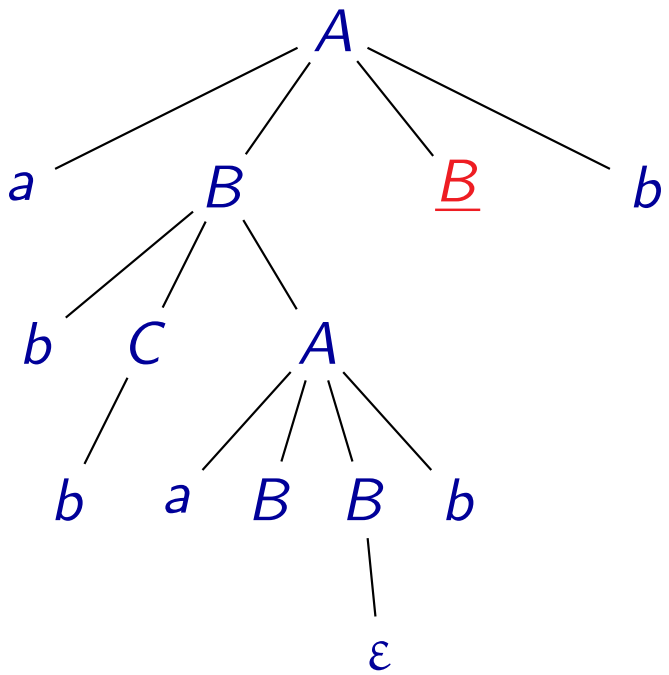
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$

Derivační strom

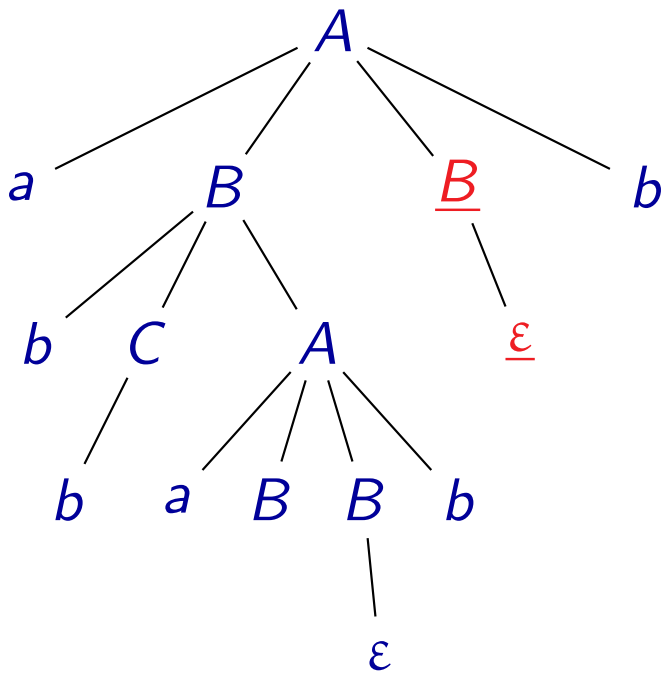
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$

Derivační strom

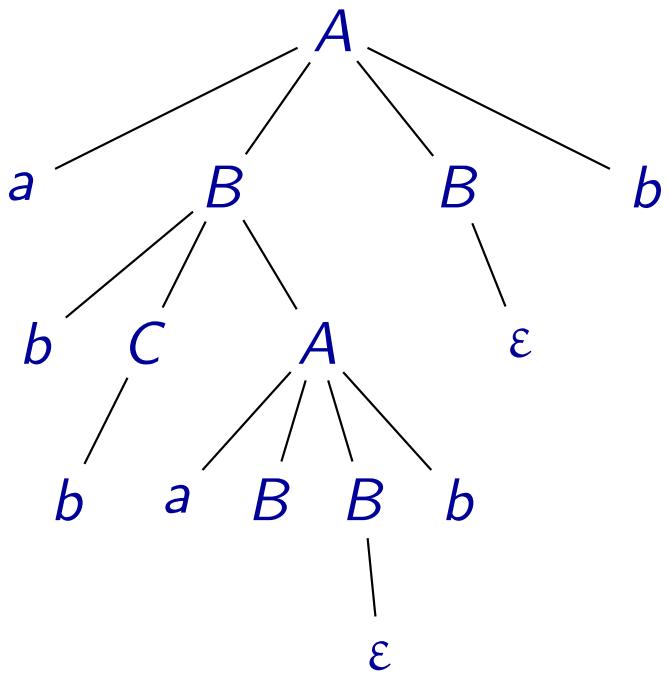
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$

Derivační strom

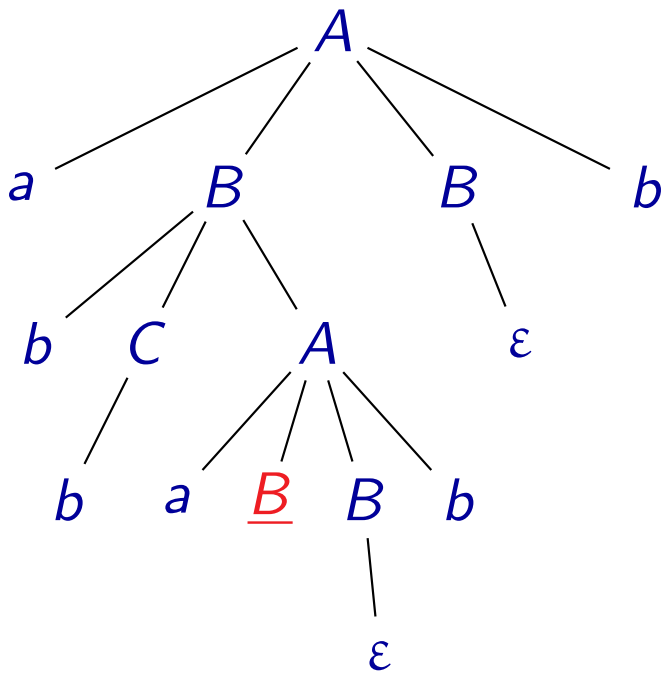
$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Derivační strom

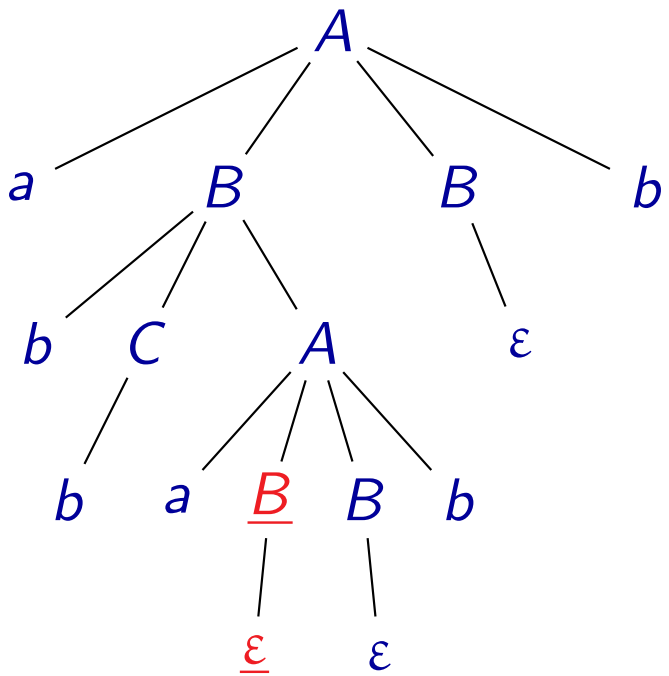
$A \to aBBb \mid AaA$
$\underline{B} \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb$

Derivační strom

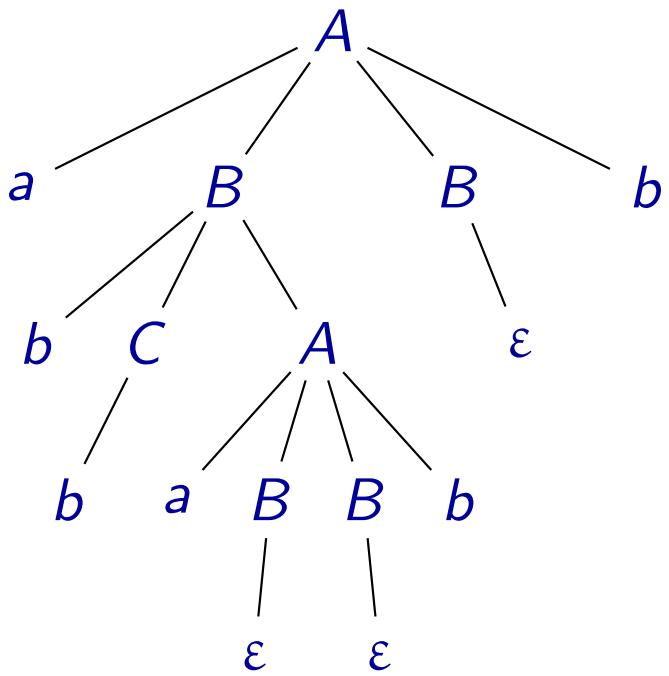
$A \to aBBb \mid AaA$
$\underline{B} \to \underline{\varepsilon} \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb \Rightarrow abbabb$

Derivační strom

$A \to aBBb \mid AaA$
$B \to \varepsilon \mid bCA$
$C \to AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$

Derivační strom

Každé derivaci odpovídá nějaký **derivační strom**:

- Vrcholy stromu jsou ohodnoceny terminály a neterminály.
- Kořen stromu je ohodnocen počátečním neterminálem.
- Listy stromu jsou ohodnoceny terminály nebo symboly ε.
- Ostatní vrcholy stromu jsou ohodnoceny neterminály.
- Pokud je vrchol ohodnocen neterminálem A, pak jeho potomci jsou ohodnoceni symboly pravé strany nějakého přepisovacího pravidla $A \to \alpha$.

Levá a pravá derivace

$$E \to E + E \mid E * E \mid (E) \mid a$$

Levá derivace je derivace, ve které v každém kroku nahrazujeme vždy nejlevější neterminál.

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow \underline{E} * E + E \Rightarrow a * \underline{E} + E \Rightarrow a * a + \underline{E} \Rightarrow a * a + a$$

Pravá derivace je derivace, ve které v každém kroku nahrazujeme vždy nejpravější neterminál.

$$\underline{E} \Rightarrow E + \underline{E} \Rightarrow \underline{E} + a \Rightarrow E * \underline{E} + a \Rightarrow \underline{E} * a + a \Rightarrow a * a + a$$

Derivace však nemusí být ani levá ani pravá:

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow E * \underline{E} + E \Rightarrow E * a + \underline{E} \Rightarrow \underline{E} * a + a \Rightarrow a * a + a$$

Levá a pravá derivace

- Jednomu derivačnímu stromu může odpovídat více různých derivací.
- Každému derivačnímu stromu odpovídá právě jedna levá a právě jedna pravá derivace.

Ekvivalence gramatik

Gramatiky G_1 a G_2 jsou **ekvivalentní**, jestliže generují tentýž jazyk, tj. jestliže $L(G_1) = L(G_2)$.

Poznámka: Problém ekvivalence bezkontextových gramatik je algoritmicky nerozhodnutelný. Dá se dokázat, že není možné vytvořit algoritmus, který by pro libovolné dvě bezkontextové gramatiky rozhodl, zda jsou ekvivalentní či ne.

Dokonce je algoritmicky nerozhodnutelný i problém, zda gramatika generuje jazyk Σ^*.

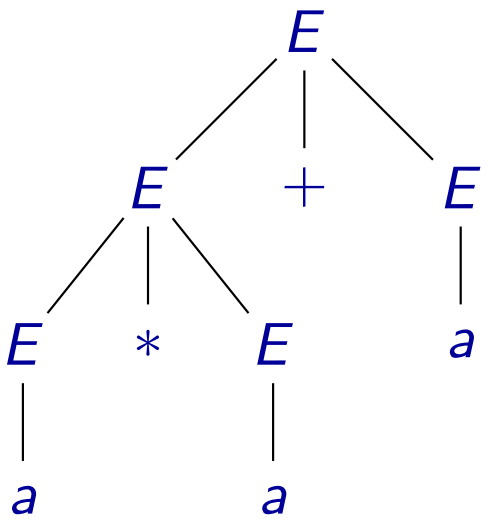
Nejednoznačné gramatiky

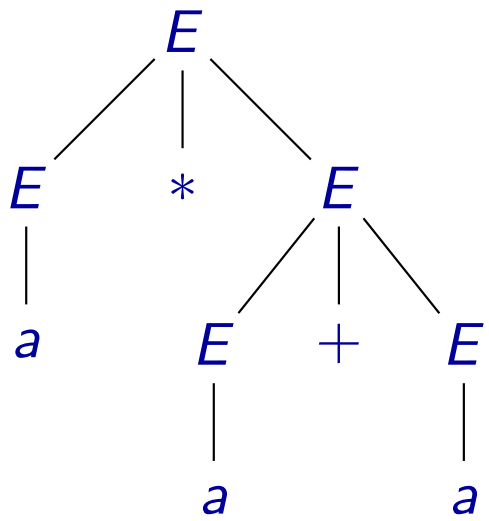
Gramatika G je **nejednoznačná**, jestliže existuje nějaké slovo $w \in L(G)$, kterému přísluší dva různé derivační stromy, resp. dvě různé levé či dvě různé pravé derivace.

Příklad:

$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow a * E + E \Rightarrow a * a + E \Rightarrow a * a + a$

$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow a * E + E \Rightarrow a * a + E \Rightarrow a * a + a$





Nejednoznačné gramatiky

Někdy je možné nejednoznačnou gramatiku nahradit gramatikou, která generuje tentýž jazyk, ale není nejednoznačná.

Příklad: Gramatiku

$$E \to E + E \mid E * E \mid (E) \mid a$$

můžeme nahradit ekvivalentní gramatikou

$$\begin{aligned} E &\to T \mid T + E \\ T &\to F \mid F * T \\ F &\to a \mid (E) \end{aligned}$$

Poznámka: Pokud se nejednoznačná gramatika žádnou ekvivalentní jednoznačnou gramatikou nahradit nedá, říkáme, že je **podstatně nejednoznačná**.

Bezkontextové jazyky

Definice

Jazyk L je **bezkontextový**, jestliže existuje bezkontextová gramatika G taková, že $L = L(G)$.

Třída bezkontextových jazyků je uzavřená vůči:

- zřetězení
- sjednocení
- iteraci

Třída bezkontextových jazyků však není uzavřená vůči:

- doplňku
- průniku

Bezkontextové jazyky

Máme dány gramatiky $G_1 = (\Pi_1, \Sigma, S_1, P_1)$ a $G_2 = (\Pi_2, \Sigma, S_2, P_2)$, přičemž můžeme předpokládat, že $\Pi_1 \cap \Pi_2 = \emptyset$ a $S \notin \Pi_1 \cup \Pi_2$.

- Gramatika G taková, že $L(G) = L(G_1)L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \to S_1 S_2\})$$

- Gramatika G taková, že $L(G) = L(G_1) \cup L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\})$$

- Gramatika G taková, že $L(G) = L(G_1)^*$:

$$G = (\Pi_1 \cup \{S\},\ \Sigma,\ S,\ P_1 \cup \{S \to \varepsilon, S \to S_1 S\})$$

Algoritmy

Příklad: Popis algoritmu pomocí **pseudokódu**:

Algoritmus 1: Algoritmus pro nalezení největšího prvku v poli

```
1  FIND-MAX (A, n):
2  begin
3      k := 0
4      for i := 1 to n − 1 do
5          if A[i] > A[k] then
6              k := i
7          end
8      end
9      return A[k]
10 end
```

Algoritmy

Algoritmus

- zpracovává **vstup**
- generuje **výstup**

Z hlediska analýzy toho, jak daný algoritmus funguje, většinou není příliš podstatný rozdíl v tom, jestli algoritmus:

- čte vstupní data z nějakého vstupního zařízení (např. ze souboru na disku, z klávesnice, apod.)
- zapisuje data na nějaké výstupní zařízení (např. do souboru, na obrazovku, apod.)

nebo

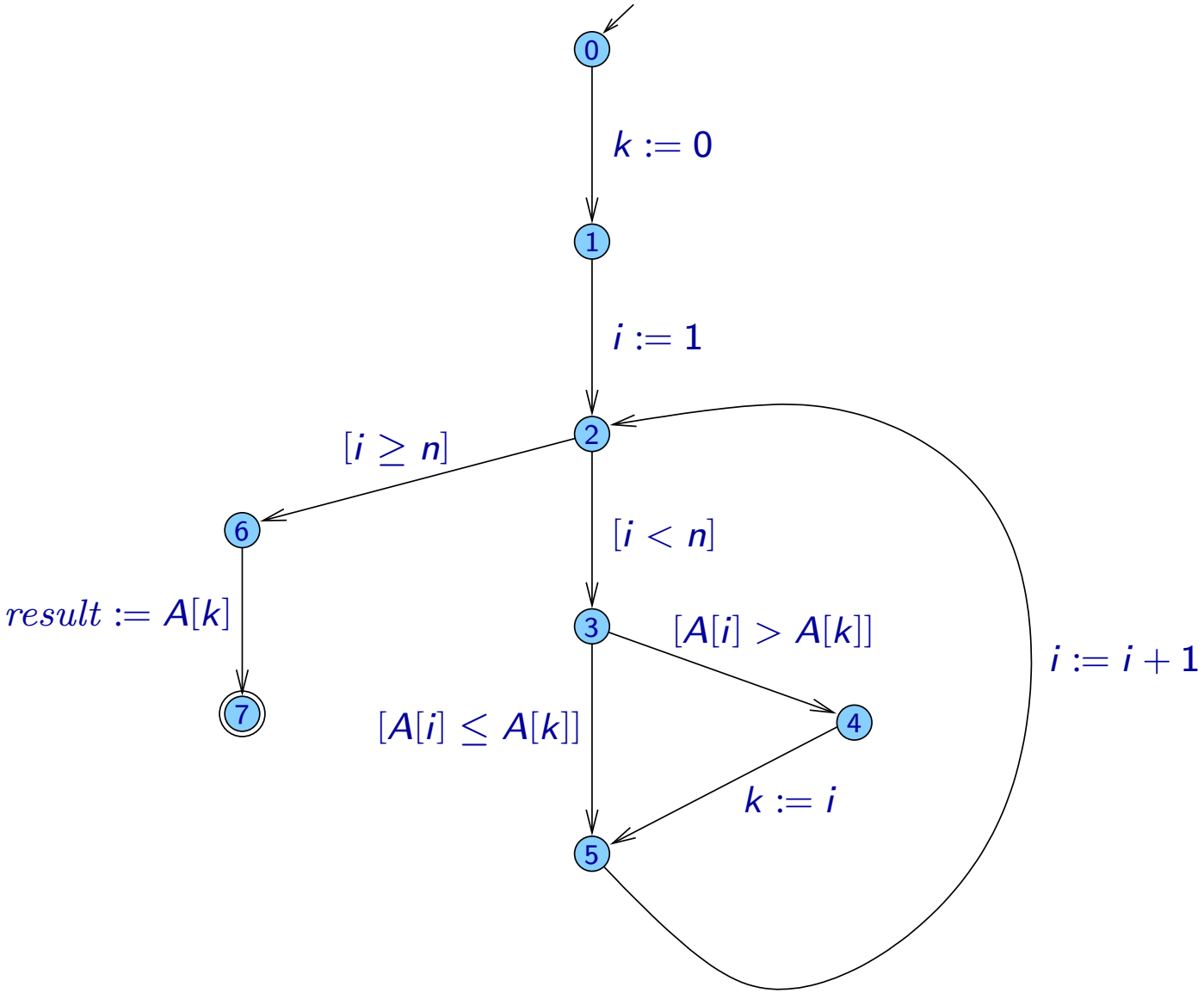
- čte vstupní data z paměti (např. jsou mu předány jako parametry)
- zapisuje data na do paměti (např. je vrátí jako návratovou hodnotu)

Řídící tok

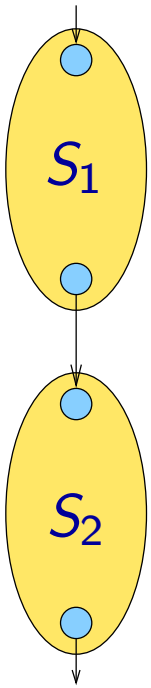
Instrukce lze zhruba rozdělit na dvě skupiny:

- intrukce přímo pracující s daty:
  - přiřazení
  - vyhodnocení hodnot výrazů v podmínkách
  - čtení vstupu, zápis na výstup
  - ...
- instrukce ovlivňující **řídící tok** — určují, které instrukce se budou provádět, v jakém pořadí, apod.:
  - větvení (if, switch, ...)
  - cykly (while, do .. while, for, ...)
  - uspořádání instrukcí do bloků
  - návraty z podpogramů (return, ...)
  - ...

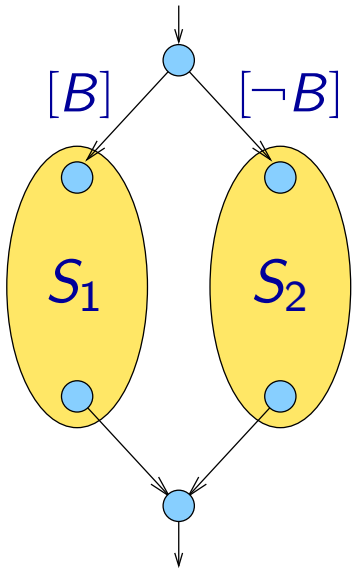
Graf řídícího toku



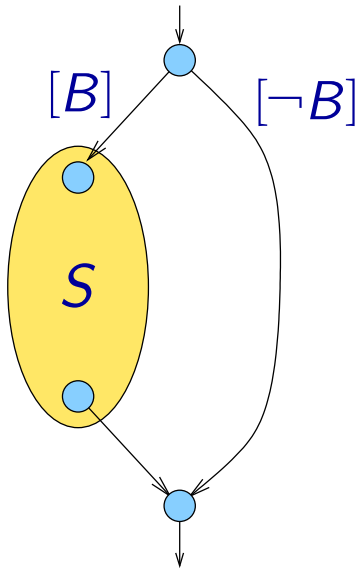
Některé základní konstrukce strukturovaného programání



$S_1; S_2$

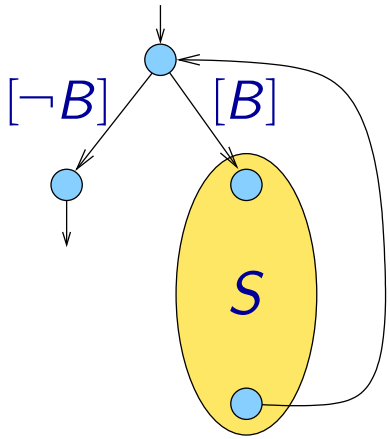


if B **then** S_1 **else** S_2

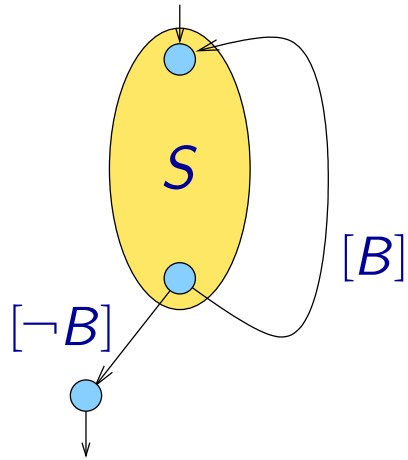


if B **then** S

Některé základní konstrukce strukturovaného programání

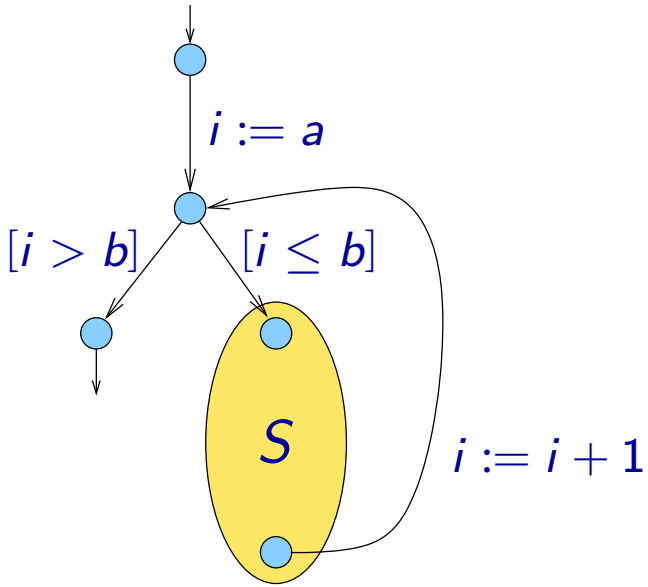


while B **do** S



do S **while** B

Některé základní konstrukce strukturovaného programání



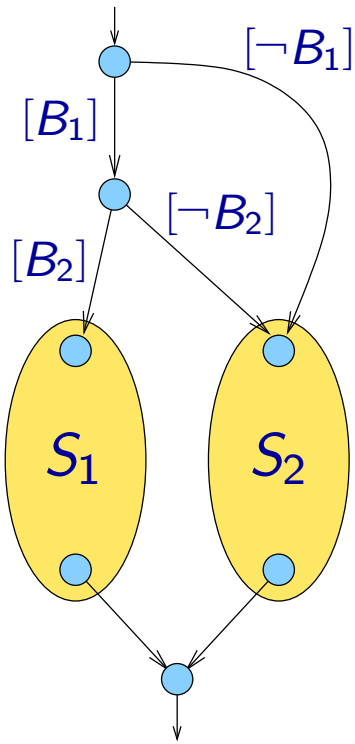
$i := a$
while $i \leq b$ **do**
    S
    $i := i + 1$
end

for $i := a$ **to** b **do** S

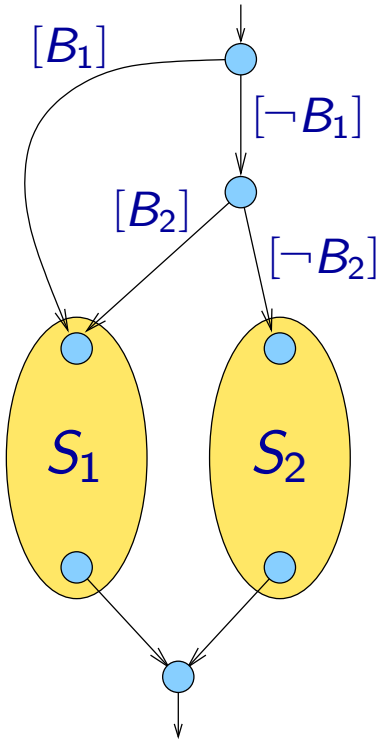
Některé základní konstrukce strukturovaného programání

Zkrácené vyhodnocování složených podmínek, např.:

while $i < n$ **and** $A[i] > x$ **do** ...



if B_1 **and** B_2 **then** S_1 **else** S_2



if B_1 **or** B_2 **then** S_1 **else** S_2

Řídící tok realizovaný pomocí goto

- **goto** ℓ — **nepodmíněný skok**
- **if** B **then goto** ℓ — **podmíněný skok**

Příklad:

0: $k := 0$
1: $i := 1$
2: **goto** *6*
3: **if** $A[i] \leq A[k]$ **then goto** *5*
4: $k := i$
5: $i := i + 1$
6: **if** $i < n$ **then goto** *3*
7: **return** $A[k]$

Řídící tok realizovaný pomocí goto

- **goto** ℓ — **nepodmíněný skok**
- **if** B **then goto** ℓ — **podmíněný skok**

Příklad:

$$
\begin{array}{rl}
\mathit{start}: & k := 0 \\
& i := 1 \\
& \textbf{goto } \mathit{L3} \\
\mathit{L1}: & \textbf{if } A[i] \leq A[k] \textbf{ then goto } \mathit{L2} \\
& k := i \\
\mathit{L2}: & i := i + 1 \\
\mathit{L3}: & \textbf{if } i < n \textbf{ then goto } \mathit{L1} \\
& \textbf{return } A[k]
\end{array}
$$

Vyhodnocení složitých výrazů

Vyhodnocení složitého výrazu, jako třeba

$$A[i+s] := (B[3*j+1]+x)*y+8$$

může být na nižší úrovni nahrazeno posloupností jednodušších příkazů, jako třeba

$$\begin{aligned}
t_1 &:= i+s \\
t_2 &:= 3*j \\
t_2 &:= t_2+1 \\
t_3 &:= B[t_2] \\
t_3 &:= t_3+x \\
t_3 &:= t_3*y \\
t_3 &:= t_3+8 \\
A[t_1] &:= t_3
\end{aligned}$$

Výpočet algoritmu

Algoritmus je vykonáván strojem — může to být například:

- skutečný počítač — vykonává instrukce strojového kódu
- virtuální stroj — vykonává instrukce bytekódu
- nějaký idealizovaný matematický model počítače
- ...

Stroj může být:

- jednoúčelový — vykonává jen jeden algoritmus
- obecnější — algoritmus dostává ve formě **programu**

Stroj pracuje po **krocích**.

Algoritmus během výpočtu zpracovává konkrétní **vstup**.

Výpočet algoritmu

Během výpočtu si stroj musí pamatovat:

- která intrukce se právě provádí
- obsah pracovní paměti

Podle typu stroje je určeno:

- s jakým typem dat stroj pracuje
- jak jsou tato data v paměti organizována

Podle typu algoritmu a typu analýzy, kterou chceme provádět, se můžeme rozhodnout, zda má smysl mezi obsah paměti zahranout i místa

- odkud se čtou vstupní data
- kam se zapisují výstupní data

Výpočet algoritmu

Konfigurace — popis celkového stavu stroje v nějakém okamžiku během výpočtu

Příklad: Konfigurace tvaru

$$(q, mem)$$

kde

- q — aktuální řídící stav
- mem — představuje aktuální obsah paměti stroje — jaké hodnoty jsou momentálně přiřazeny jednotlivým proměnným.

Příklad obsahu paměti mem:

$$\langle A\colon [3,8,1,3,6],\quad n\colon 5,\quad i\colon 1,\quad k\colon 0,\quad result\colon ?\rangle$$

Výpočet algoritmu

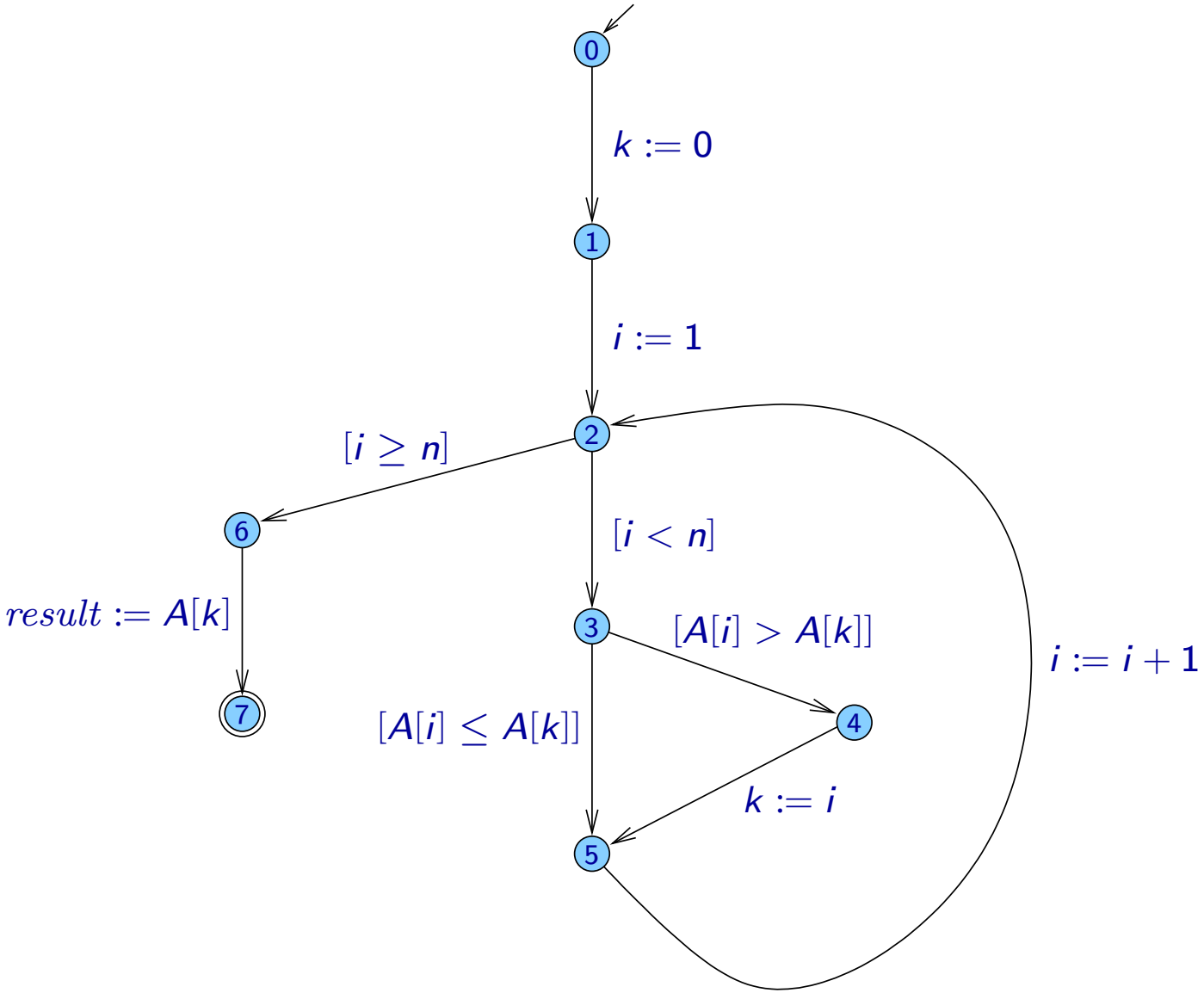
Příklad konfigurace:

$$(2,\ \langle A\colon [3,8,1,3,6],\ \ n\colon 5,\ \ i\colon 1,\ \ k\colon 0,\ \ \mathit{result}\colon ?\rangle)$$

Výpočet stroje $\mathcal{M}$ provádějícího algoritmus Alg, kde zpracovává vstup w, je posloupnost konfigurací.

- Začíná se v **počáteční konfiguraci**.
- Každým krokem se přejde z jedné konfigurace do druhé.
- Výpočet končí v **koncové konfiguraci**.

Výpočet algoritmu



Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3, 8, 1, 3, 6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0,\ \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1,\ \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } ?, \quad result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } ?, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 0, \quad result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 1, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 2, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 3, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \quad n\text{: } 5, \quad i\text{: } 3, \quad k\text{: } 1, \quad result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } ?,\ result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } ?,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 0,\ result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 1,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 2,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6],\ n\text{: } 5,\ i\text{: } 3,\ k\text{: } 1,\ result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

$$
\begin{array}{rl}
\alpha_0\colon & (0,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon ?,\ result\colon ?\rangle) \\
\alpha_1\colon & (1,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon ?,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_2\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_3\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_4\colon & (4,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 0,\ result\colon ?\rangle) \\
\alpha_5\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 1,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_6\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_7\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_8\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 2,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_9\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{10}\colon & (3,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{11}\colon & (5,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 3,\ k\colon 1,\ result\colon ?\rangle) \\
\alpha_{12}\colon & (2,\ \langle A\colon [3,8,1,3,6],\ n\colon 5,\ i\colon 4,\ k\colon 1,\ result\colon ?\rangle)
\end{array}
$$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{16}: $(6, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle)$

Výpočet algoritmu

Příklad: Výpočet, kde algoritmus FIND-MAX zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0: $(0, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } ?, \; result\text{: } ?\rangle)$
α_1: $(1, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } ?, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_2: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_3: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_4: $(4, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 0, \; result\text{: } ?\rangle)$
α_5: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 1, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_6: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_7: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_8: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 2, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_9: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{10}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{11}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 3, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{12}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{13}: $(3, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{14}: $(5, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 4, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{15}: $(2, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{16}: $(6, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } ?\rangle)$
α_{17}: $(7, \langle A\text{: } [3,8,1,3,6], \; n\text{: } 5, \; i\text{: } 5, \; k\text{: } 1, \; result\text{: } 8\rangle)$

Výpočet algoritmu

Provedením instrukce I se přejde z konfigurace α do konfigurace α':

$$\alpha \xrightarrow{I} \alpha'$$

Výpočet může být:

- **Konečný**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \quad \cdots \quad \xrightarrow{I_{t-2}} \alpha_{t-1} \xrightarrow{I_{t-1}} \alpha_t$$

  kde α_t je koncová konfigurace

- **Nekonečný**:

$$\alpha_0 \xrightarrow{I_0} \alpha_1 \xrightarrow{I_1} \alpha_2 \xrightarrow{I_2} \alpha_3 \xrightarrow{I_3} \alpha_4 \xrightarrow{I_4} \quad \cdots$$

Výpočet algoritmu

Výpočet je možné popsat dvěma různými způsoby:

- jako posloupnost konfigurací $\alpha_0, \alpha_1, \alpha_2, \ldots$
- jako posloupnost provedených instrukcí $I_0, I_1, I_2, \ldots$

Korektnost algoritmu

Algoritmy slouží k řešení **problémů**.

- **Problém** — specifikace toho, **co** má algoritmus dělat:
  - Popis vstupu
  - Popis výstupu
  - Vztah mezi vstupy a výstupy
- **Algoritmus** — konkrétní postup, **jak** při výpočtu postupovat

Korektnost algoritmu

Příklad: Problém nalezení maximálního prvku v poli:

Vstup: Pole A indexované od nuly a číslo n udávající počet prvků v tomto poli, přičemž se předpokládá, že $n \geq 1$.

Výstup: Hodnota $result$, která je hodnotou maximálního prvku v poli A, tj. hodnota $result$, pro kterou platí:

- $A[j] \leq result$ pro všechna $j \in \mathbb{N}$, kde $0 \leq j < n$, a
- existuje $j \in \mathbb{N}$ takové, že $0 \leq j < n$ a $A[j] = result$.

Instance problému — konkrétní vstup, např.

$$A = [3, 8, 1, 3, 6], \; n = 5.$$

Pro tuto instanci je výstupem hodnota 8.

Korektnost algoritmu

Definice

Algoritmus Alg **řeší** problém P, jestliže pro **každou** instanci w problému P jsou splněny následující dvě podmínky:

(a) Výpočet algoritmu Alg nad vstupem w se po konečném počtu kroků (korektně) zastaví.

(b) Algoritmus Alg vygeneruje pro vstup w výstup, který odpovídá podmínkám kladeným na výstup ve specifikaci problému P.

Algoritmus, který řeší problém P, je korektním řešením tohoto problému.

Korektnost algoritmu

Algoritmus Alg **není** korektním řešení problému P, jestliže existuje vstup w takový, že při výpočtu nad tímto vstupem nastane některá z následujících chyb:

- provedení nějaké chybné nepovolené operace (přístup k prvku pole mimo povolený rozsah indexů, dělení nulou, ...),
- vygenerovaný výstup neodpovídá podmínkám specifikovaným v zadání problému P,
- výpočet se nikdy nezastaví.

Testování — spustění algoritmu nad různými vstupy a zkontrolování, zda se algoritmus pro tyto vstupy chová „správně“.

Testování může prokázat přítomnost chyb, ale ne to, že se algoritmus chová korektně pro **všechny** vstupy.

Korektnost algoritmu

Důkaz korektnosti algoritmu je obecně vhodné rozdělit na dvě části:

- Zdůvodnění toho, že algoritmus pro žádný vstup nikdy neudělá nic „špatně“:
  - během výpočtu nedojde k žádné chybné operaci
  - pokud program skončí, výstup bude „správně“
- Zdůvodnění toho, že se algoritmus pro každý vstup po konečném počtu kroků zastaví.

Korektnost algoritmu

Invariant — podmínka, která musí být v určitém místě kódu algoritmu vždy (tj. ve všech možných výpočtech pro všechny možné vstupy) splněna v okamžiku, kdy algoritmus tímto místem prochází.

Řekneme, že konfigurace α je **dosažitelná**, jestliže existuje vstup w takový, že je α jednou z konfigurací, kterými algoritmus Alg projde při výpočtu nad vstupem w.

Pokud je algoritmus reprezentován ve formě grafu řídícího toku, můžeme pro **řídící stav** q (tj. vrchol grafu) specifikovat invarianty, které platí v každé dosažitelné konfiguraci, kde je řídícím stavem q.

Invarianty

Invarianty můžeme zapisovat formulemi predikátové logiky:

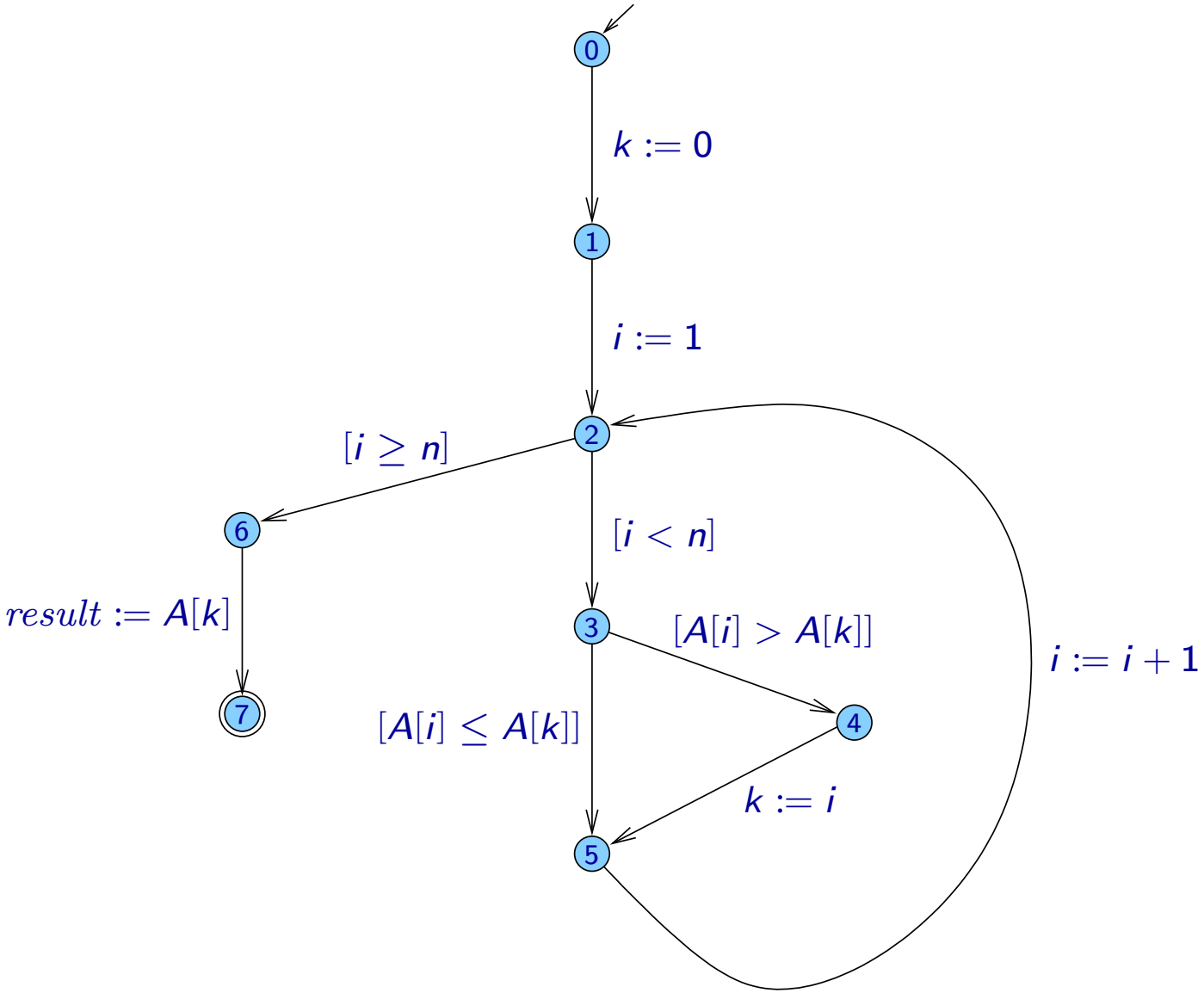
- **volné** proměnné odpovídají proměnným programu
- **valuace** je dána hodnotami proměnných programu v dané konfiguraci

Příklad: Formule

$$(1 \leq i) \wedge (i \leq n)$$

bude platit například v konfiguraci, kde proměnná i má hodnotu 5 a proměnná n má hodnotu 14.

Invarianty



Invarianty

Příklady invariantů:

- invariant v řídícím stavu q zapíšeme formulí φ_q

Invarianty v jednotlivých řídících stavech (zatím jen hypotézy):

- φ_0: $(n \geq 1)$
- φ_1: $(n \geq 1) \wedge (k = 0)$
- φ_2: $(n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i)$
- φ_3: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- φ_4: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- φ_5: $(n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k \leq i)$
- φ_6: $(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$
- φ_7: $(n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$

Invarianty

Příklady invariantů:

- invariant v řídícím stavu q zapíšeme formulí φ_q

Invarianty v jednotlivých řídících stavech (zatím jen hypotézy):

- φ_0: $n \geq 1$
- φ_1: $n \geq 1, \quad k = 0$
- φ_2: $n \geq 1, \quad 1 \leq i \leq n, \quad 0 \leq k < i$
- φ_3: $n \geq 1, \quad 1 \leq i < n, \quad 0 \leq k < i$
- φ_4: $n \geq 1, \quad 1 \leq i < n, \quad 0 \leq k < i$
- φ_5: $n \geq 1, \quad 1 \leq i < n, \quad 0 \leq k \leq i$
- φ_6: $n \geq 1, \quad i = n, \quad 0 \leq k < n$
- φ_7: $n \geq 1, \quad i = n, \quad 0 \leq k < n$

Invarianty

Zkontrolování toho, že invarianty opravdu platí:

- Pro každou instrukci algoritmu je třeba zkontrolovat, zda za předpokladu, že bude platit příslušný invariant před provedením této instrukce, bude platit i příslušný invariant po provedení této instrukce.

Předpokládejme algoritmus ve formě grafu řídícího toku:

- hrany odpovídají instrukcím
- vezměme si hranu ze stavu q do stavu q' označenou instrukcí I
- řekněme, že (zatím neověřené) invarianty pro stavy q a q' jsou vyjádřeny formulemi φ a φ'
- pro tuto hranu musíme zkontrolovat, že pro všechny konfigurace $\alpha = (q, mem)$ a $\alpha' = (q', mem')$ takové, že $\alpha \xrightarrow{I} \alpha'$, platí, že pokud
  - v konfiguraci α platí φ,

  pak
  - v konfiguraci α' platí φ'

Invarianty

Zkotrolování instrukcí, které jsou testy podmínek:

- hrana označená testem podmínky $[B]$

Obsah paměti se nemění.
Stačí ověřit, že platí implikace

$$(\varphi \wedge B) \rightarrow \varphi'$$

Poznámka: Příslušná implikace musí platit pro všechny možné hodnoty proměnných.

Příklad: Předpokládáme, že se formulích objevují jen proměnné n, i, k, a že hodnotami těchto proměnných mohou být jen celá čísla:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})\,(\varphi \wedge B \rightarrow \varphi')$$

Invarianty

Zkotrolování instrukcí, které přiřazují hodnoty proměnným (mění obsah paměti):

- hrana označená přiřazením $x := E$

φ'' — formule, kterou dostaneme z formule φ' přejmenováním všech volných výskytů proměnné x na proměnnou x'

Je třeba ověřit platnost implikace

$$(\varphi \wedge (x' = E)) \rightarrow \varphi''$$

Příklad: Přiřazení $k := 3 * k + i + 1$:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})(\forall k' \in \mathbb{Z})\,(\varphi \wedge (k' = 3 * k + i + 1) \rightarrow \varphi'')$$

Invarianty

Dokončení ověření toho, že algoritmus pro nalezení maximálního prvku v poli vrací správný výsledek (za předpokladu, že skončí):

- ψ_0: φ_0
- ψ_1: $\varphi_1 \wedge (\forall j \in \mathbb{N})(0 \leq j < 1 \rightarrow A[j] \leq A[k])$
- ψ_2: $\varphi_2 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- ψ_3: $\varphi_3 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- ψ_4: $\varphi_4 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k]) \wedge (A[i] > A[k])$
- ψ_5: $\varphi_5 \wedge (\forall j \in \mathbb{N})(0 \leq j \leq i \rightarrow A[j] \leq A[k])$
- ψ_6: $\varphi_6 \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq A[k])$
- ψ_7: $\varphi_7 \wedge (result = A[k]) \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq result) \wedge (\exists j \in \mathbb{N})(0 \leq j < n \wedge A[j] = result)$

Invarianty

Často není třeba specifikovat invarianty ve všech řídících stavech, ale je v některých „důležitých“ — zejména stavy, kde se vstupuje do nebo vystupuje z cyklů:

Je pak třeba ověřit:

- Že invariant platí před vstupem do cyklu.
- Že pokud invariant platí před provedením cyklu, tak bude platit i po jeho provedení.
- Že invariant platí při opuštění cyklu.

Invarianty

Příklad: V algoritmu FIND-MAX je takovým „důležitým“ stavem stav 2.

Ve stavu 2 platí:

- $n \geq 1$
- $1 \leq i \leq n$
- $0 \leq k < i$
- Pro všechna j taková, že $0 \leq j < i$, platí $A[j] \leq A[k]$.

Konečnost výpočtu

Dva možné případy, jak může vypadat nekonečný výpočet:

- nějaká konfigurace se zopakuje — následující konfigurace se opakují stále dokola
- objevují se stále nové a nové konfigurace

Konečnost výpočtu

Jeden z běžných způsobů dokazování toho, že se algoritmus zaručeně pro každý vstup po konečném počtu kroků zastaví:

- každé (dosažitelné) konfiguraci přiřadit hodnotu z nějaké vhodně zvolené množiny W
- na množině W definovat uspořádání $\leq$ takové, že ve W neexistují nekonečné (ostře) klesající posloupnosti
- ukázat, že s provedením každé instrukce se hodnota přiřazená konfiguraci zmenšuje, tj. pro $\alpha \xrightarrow{I} \alpha'$ je

$$f(\alpha) > f(\alpha')$$

($f(\alpha)$, $f(\alpha')$ jsou hodnoty z množiny W přiřazené konfiguracím α a α')

Konečnost výpočtu

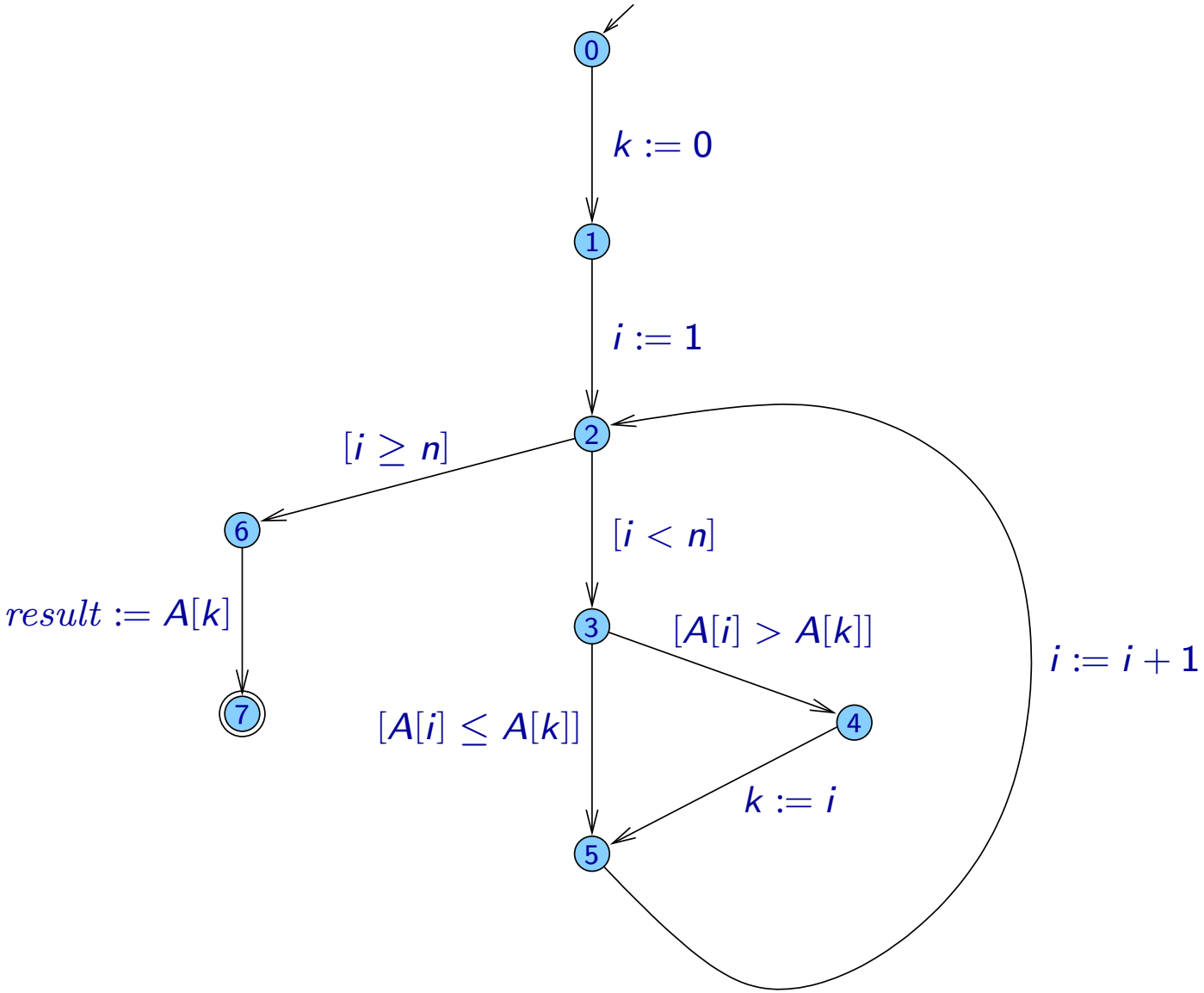
Jako množinu W je možno použít například:

- Množinu přirozených čísel $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ s uspořádáním $\leq$.
- Množinu vektorů přirozených číslel s lexikografickým uspořádáním, tj. s uspořádáním, kde vektor $(a_1, a_2, \ldots, a_m)$ je menší než vektor $(b_1, b_2, \ldots, b_n)$, jestliže
  - existuje i takové, že $1 \leq i \leq m$ a $i \leq n$, kde $a_i < b_i$ a pro všechna j taková, že $1 \leq j < i$, platí $a_j = b_j$, nebo
  - $m < n$ a pro všechna j taková, že $1 \leq j \leq m$, je $a_j = b_j$.

  Například $(5, 1, 3, 6, 4) < (5, 1, 4, 1)$ a $(4, 1, 1) < (4, 1, 1, 3)$.

  Poznámka: Počet prvků vektorů musí být omezen nějakou konstantou.

Konečnost výpočtu



Konečnost výpočtu

Příklad: Vektory přiřazené jednotlivým konfiguracím:

- Stav 0: $f(\alpha) = (4)$
- Stav 1: $f(\alpha) = (3)$
- Stav 2: $f(\alpha) = (2, n - i, 3)$
- Stav 3: $f(\alpha) = (2, n - i, 2)$
- Stav 4: $f(\alpha) = (2, n - i, 1)$
- Stav 5: $f(\alpha) = (2, n - i, 0)$
- Stav 6: $f(\alpha) = (1)$
- Stav 7: $f(\alpha) = (0)$

Výpočetní složitost algoritmů

Složitost algoritmu

- Počítače pracují rychle, ale ne nekonečně rychle. Provedení každé instrukce trvá nějakou (i když velmi krátkou) dobu.
- Stejný problém může řešit více různých algoritmů a doba výpočtu může být různá — chtěli bychom mít možnost algoritmy vzájemně porovnat.
- Algoritmy můžeme naprogramovat a změřit čas výpočtu. Tím zjistíme jak dlouho trvá výpočet na konkrétních datech, na kterých algoritmus testujeme.
- Chtěli bychom mít i nějakou přesnější představu o tom, jak dlouho bude trvat výpočet na všech možných vstupních datech.

Složitost algoritmu

- Doba výpočtu je ovlivněna mnoha faktory, např.:
  - použitý algoritmus
  - množství vstupních dat
  - použitý hardware (důležitá může být např. taktovací frekvence procesoru)
  - použitý programovací jazyk — a jeho konkrétní implementace (překladač/interpreter)
  - ...
- Pokud potřebujeme řešit problém pro „malá“ vstupní data, doba výpočtu je většinou zanedbatelná.
- S narůstajícím množstvím vstupních dat (velikosti vstupu) může doba výpočtu růst, někdy velmi výrazně.

Složitost algoritmu

- **Časová složitost algoritmu** — jak závisí doba výpočtu na množství vstupních dat
- **Paměťová** (resp. **prostorová**) **složitost algoritmu** — jak závisí množství použité paměti na množství vstupních dat

Poznámka: Přesné definice budou uvedeny později.

Poznámka:

- Existují i další typy výpočetní složitosti, kterými se nebudeme zabývat (např. komunikační složitost).

Složitost algoritmu

Přesné určení doby výpočtu nebo množství použité paměti může být extrémně komplikované.

Většinou se při analýze výpočetní složitosti algoritmu používá celá řada zjednodušení:

- Většinou se neanalyzuje, jak závisí doba výpočtu nebo množství použité paměti na konkrétních vstupních datech, ale pouze, jak závistí na **velikosti vstupu**, tj. na množství těchto dat.

- Funkce vyjadřující, jak roste doba výpočtu nebo množství použité paměti v závislosti na velikosti vstupu, se nepočítají přesně — počítají se **odhady** těchto funkcí.

- Odhady těchto funkcí se vyjadřují pomocí tzv. **asymptotické notace** — např. se řekne, že časová složitost algoritmu MergeSort je $O(n \log n)$, zatímco časová složitost algoritmu BubbleSort je $O(n^2)$.

Velikost vstupu

Velikost vstupu — hodnota udávající, jak je daná vstupní instance „velká“

- Nejčastěji je velikost vyjádřena jako jediné číslo — obvykle se označuje toto číslo n nebo N.
- Někdy je vhodnější vyjádřit velikost dvojicí (občas i trojicí, čtveřicí, atd.) parametrů — v tom případě se často označují n a m (nebo N a M).
- Co přesně bude považováno za velikost vstupu, si můžeme zvolit.

Velikost vstupu

Příklady toho, co například může být velikostí vstupu:

- Vstupem je sekvence nějakých hodnot, pole prvků apod. (např. u problému třídění, vyhledávání v poli, hledání maximálního prvku, apod.):

  n — počet prvků v této sekvenci nebo poli

- Vstupem je řetězec znaků (slovo z nějaké abecedy):

  n — počet znaků v tomto řetězci

- Vstupem jsou dva řetězce, např. (dlouhý) text, který se bude prohledávat, a (kratší) hledaný řetězec:

  n — počet znaků v prohledávaném textu
  m — počet znaků hledaného řetězce

Velikost vstupu

- Vstupem je množina řetězců:

  Jedna možnost:

  n — součet délek všech řetězců

  Jiná varianta:

  n — součet délek všech řetězců, m — počet řetězců

- Vstupem je graf:

  n — počet vrcholů, m — počet hran

Velikost vstupu

- Vstupem je jedno číslo (např. u testování prvočíselnosti):

  Jedna možnost:

  n — počet bitů daného čísla — např. velikost vstupu 962261 je 20

  Jiná varianta:

  n — hodnota daného čísla — velikost vstupu 962261 je 962261

- Vstupem je posloupnost čísel, přičemž hodnoty těchto čísel ovlivňují dobu výpočtu (např. u problému, kde je cílem spočítat největšího společného dělitele všech čísel v dané posloupnosti):

  n — součet počtu bitů všech čísel v dané posloupnosti

Doba výpočtu

Řekněme, že máme:

- algoritmus Alg řešící problém P (resp. konkrétní implementaci algoritmu Alg),
- stroj $\mathcal{M}$ vykonávající algoritmus Alg,
- vstup w z množiny In, což je množina všech vstupů pro problém P

Příklad:

- konkrétní implementace algoritmu Quicksort v jazyce C++ řešící problém třídění,
- počítač s daným konkrétním typem procesoru, s určitou konkrétní frekvencí, na které pracuje procesor, s daným konkrétním množstvím paměti, operačním systémem, atd.
- vstup: pole $[6, 13, 1, 8, 4, 5, 8]$
  (pozn.: realističtější by byl příklad pole s milionem prvků)

Doba výpočtu

$t(w)$ — doba výpočtu algoritmu Alg nad vstupem w na stroji $\mathcal{M}$

V jakých jednotkách dobu výpočtu udávat? (Uvidíme, že ve skutečnosti to při použití asymptické notace není podstatné.)

- **v sekundách** — závisí na příliš mnoha detailech implementace, těžké určit jinak než měřením

  (i na tomtéž počítači s těmi samými daty může doba výpočtů různě kolísat)

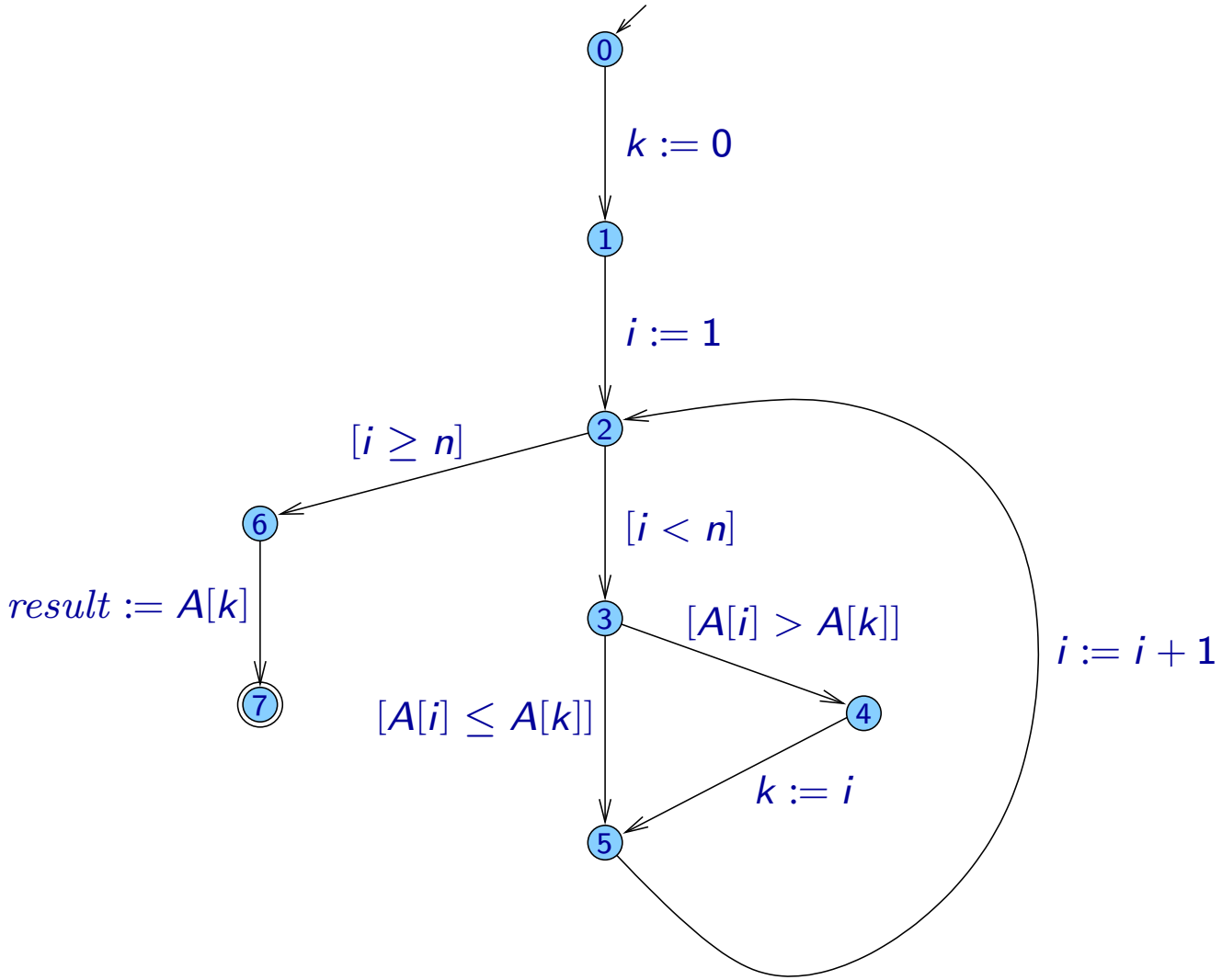
- **v počtu provedených kroků** — je třeba specifikovat, co považovat za jedek krok, například:
  - jeden příkaz vyššího programovacího jazyka
  - jedna instrukce strojového kódu nebo bytekódu
  - jeden takt procesoru
  - jedna operace určitého typu — např. porovnání, aritmetická operace, apod. (přičemž ostatní operace jsou ignorovány)
  - ...

Doba výpočtu

Řekněme, že máme algoritmus reprezentován ve formě grafu řídícího toku:

- Každé instrukci (tj. každé hraně) přiřadíme hodnotu udávající, jak dlouho trvá provedení této instrukce.
- Provedení různých instrukcí může trvat různou dobu.
- Pro jednoduchost předpokládejme, že provedení té samé instrukce trvá pokaždé stejnou dobu — hodnota přiřazená dané instrukci je číslo z množiny $\mathbb{R}^+$ (množina nezáporných reálných čísel).

Doba výpočtu



Instr.	doba
$k := 0$	c_0
$i := 1$	c_1
$[i < n]$	c_2
$[i \geq n]$	c_3
$[A[i] \leq A[k]]$	c_4
$[A[i] > A[k]]$	c_5
$k := i$	c_6
$i := i + 1$	c_7
$result := A[k]$	c_8

Doba výpočtu

Příklad: Doby provedení jednotlivých instrukcí by mohly být třeba:

Instr.	označení	doba
$k := 0$	c_0	4
$i := 1$	c_1	4
$[i < n]$	c_2	10
$[i \geq n]$	c_3	12
$[A[i] \leq A[k]]$	c_4	14
$[A[i] > A[k]]$	c_5	12
$k := i$	c_6	5
$i := i + 1$	c_7	6
$result := A[k]$	c_8	5

Pro konkrétní vstup w, např. pro $w = ([3, 8, 4, 5, 2], 5)$, bychom mohli výpočet odsimulovat a určit konkrétní dobu výpočtu $t(w)$.

Časová složitost algoritmu

Řekněme, že:

- Pro daný algoritmus Alg a stroj $\mathcal{M}$ a každý vstup w z množiny všech vstupů In je přesně definována doba výpočtu $t(w)$.
- Každému vstupu w z množiny In je přiřazeno číslo $size(w)$ udávající velikost vstupu w.

  (Formálně se jedná o funkci $size : In \to \mathbb{N}$.)

Definice

Časová složitost algoritmu Alg v nejhorším případě je funkce $T : \mathbb{N} \to \mathbb{R}^+$, která každému přirozenému číslu n přiřazuje maximální dobu výpočtu algoritmu Alg nad vstupem velikosti n.
Pro každé $n \in \mathbb{N}$ tedy platí:

- Pro každý vstup $w \in In$ takový, že $size(w) = n$, je $t(w) \leq T(n)$.
- Existuje vstup $w \in In$ takový, že $size(w) = n$ a $t(w) = T(n)$.

Časová složitost algoritmu

Z definice vidíme, že časová složitost algoritmu je funkce, jejíž přesné hodnoty závistí nejen na daném algoritmu Alg, ale také na následujících věcech:

- na stroji $\mathcal{M}$, na kterém algoritmus Alg běží,
- na definici doby výpočtu $t(w)$ algoritmu Alg na stroji $\mathcal{M}$ pro vstup $w \in In$,
- na definici velikosti vstupu (tj. definici funkce $size$).

Časová složitost algoritmu

Někdy se také určuje časová složitost algoritmu v **průměrném případě**:

- Musí se předpokládat určité **pravděpodobností rozdělení** na dané množině vstupů.
- Místo maximální doby výpočtu nad vstupy velikosti n se uvažuje střední hodnota doby těchto výpočtů.
- Většinou je analýza v průměrném případě o dost komplikovanější než analýza nejhoršího případu.
- Často se tyto dvě funkce příliš neliší, někdy je ale rozdíl významný.

Poznámka: Zkoumat složitost v **nejlepším případě** většinou moc smysl nemá.

Časová složitost algoritmu

Příklad analýzy časové složitosti algoritmu FIND-MAX **bez** použití asymptotické notace:

- Takto podrobně se analýza výpočetní složitosti algoritmu téměř nikdy **nedělá** — je to příliš pracné a komplikované.
- Uvidíme tak ale, co vše je při použití asymptotické notace zanedbáno a o kolik je analýza s použitím asymptotické notace jednodušší.
- Budeme počítat s konstantami $c_0, c_1, \ldots, c_8$, které udávají dobu trvání jednotlivých instrukcí — nebudeme počítat s konkrétními čísly.

Časová složitost algoritmu

Předpokládáme vstupy tvaru (A, n), kde A je pole a n počet prvků tohoto pole (přičemž $n \geq 1$).

Jako velikost vstupu (A, n) zvolme n.

Uvažujme nyní o nějaké jednom vstupu $w = (A, n)$ velikosti n:

- Dobu výpočtu $t(w)$ nad vstupem w můžeme vyjádřit jako

$$t(w) = c_0 m_0 + c_1 m_1 + \cdots + c_8 m_8,$$

kde $m_0, m_1, \ldots, m_8$ jsou čísla udávající, kolikrát je daná instukce při výpočtu nad vstupem w provedena.

Časová složitost algoritmu

Instr.	doba	počet provedení	hodnota m_i
$k := 0$a	c_0	m_0	1
$i := 1$	c_1	m_1	1
$[i < n]$	c_2	m_2	$n-1$
$[i \geq n]$	c_3	m_3	1
$[A[i] \leq A[k]]$	c_4	m_4	$n-1-\ell$
$[A[i] > A[k]]$	c_5	m_5	ℓ
$k := i$	c_6	m_6	ℓ
$i := i+1$	c_7	m_7	$n-1$
$result := A[k]$	c_8	m_8	1

ℓ — počet průchodů cyklem, kdy platí $A[i] > A[k]$ (zjevně je $0 \leq \ell < n$)

Časová složitost algoritmu

Dosazením do

$$t(w) = c_0 m_0 + c_1 m_1 + \cdots + c_8 m_8,$$

dostaneme

$$t(w) = d_1 + d_2 \cdot (n-1) + d_3 \cdot (n-1-\ell) + d_4 \cdot \ell,$$

kde

$$d_1 = c_0 + c_1 + c_3 + c_8 \qquad d_3 = c_4$$
$$d_2 = c_2 + c_7 \qquad d_4 = c_5 + c_6$$

Po úpravě je

$$t(w) = (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot \ell + (d_1 - d_2 - d_3)$$

Poznámka: $t(w)$ není časová složitost, ale doba výpočtu pro konkrétní vstup w

Časová složitost algoritmu

Například pokud budou doby provedení jednotlivých instrukcí následující:

Instr.	označení	doba
$k := 0$	c_0	4
$i := 1$	c_1	4
$[i < n]$	c_2	10
$[i \geq n]$	c_3	12
$[A[i] \leq A[k]]$	c_4	14
$[A[i] > A[k]]$	c_5	12
$k := i$	c_6	5
$i := i + 1$	c_7	6
$result := A[k]$	c_8	5

bude $d_1 = 25$, $d_2 = 16$, $d_3 = 14$ a $d_4 = 17$.

V takovém případě je $t(w) = 30n + 3\ell - 5$.

Pro konkrétní vstup $w = ([3, 8, 4, 5, 2], 5)$ je $n = 5$ a $\ell = 1$, takže $t(w) = 30 \cdot 5 + 3 \cdot 1 - 5 = 148$.

Časová složitost algoritmu

Pro které vstupy velikosti n bude výpočet trvat nejdéle (tj. které vstupy představují nejhorší případ), může záviset na detailech implementace a přesných hodnotách konstant:

Doba výpočtu algoritmu FIND-MAX pro vstup $w = (A, n)$ velikosti n:

$$t(w) \;=\; (d_2 + d_3) \cdot n \;+\; (d_4 - d_3) \cdot \ell \;+\; (d_1 - d_2 - d_3)$$

- Pokud $d_3 \geq d_4$ — nejhorší jsou případy, kdy má ℓ co nejmenší hodnotu

  $\ell = 0$ — například vstupy tvaru $[0, 0, \ldots, 0]$ nebo třeba $[n,\ n-1,\ n-2,\ \ldots,\ 2,\ 1]$

- Pokud $d_3 \leq d_4$ — nejhorší jsou případy, kdy má ℓ co největší hodnotu

  $\ell = n - 1$ — například vstupy tvaru $[0, 1, \ldots, n-1]$

Časová složitost algoritmu

Časová složitost $T(n)$ algoritmu FIND-MAX v nejhorším případě je tedy dána následovně:

- Pokud $d_3 \geq d_4$:

$$T(n) = (d_2 + d_3) \cdot n + (d_1 - d_2 - d_3)$$

- Pokud $d_3 \leq d_4$:

$$\begin{aligned} T(n) &= (d_2 + d_3) \cdot n + (d_4 - d_3) \cdot (n-1) + (d_1 - d_2 - d_3) \\ &= (d_2 + d_4) \cdot n + (d_1 - d_2 - d_4) \end{aligned}$$

Příklad: Pro $d_1 = 25$, $d_2 = 16$, $d_3 = 14$ a $d_4 = 17$ bude

$$\begin{aligned} T(n) &= (16 + 17) \cdot n + (25 - 16 - 17) \\ &= 33n - 8 \end{aligned}$$

Časová složitost algoritmu

V obou případech (ať už $d_3 \geq d_4$ nebo $d_3 \leq d_4$) bude časová složitost algoritmu FIND-MAX funkce tvaru

$$T(n) = an + b$$

kde a a b jsou nějaké konstanty, jejichž přesné hodnoty závisí na délce trvání jednotlivých instrukcí.

Poznámka: Konkrétně bychom tyto konstanty mohli vyjádřit jako

$$a = d_2 + \max\{d_3, d_4\} \qquad b = d_1 - d_2 - \max\{d_3, d_4\}$$

Například

$$T(n) = 33n - 8$$

Časová složitost algoritmu

Pokud bychom se spokojili s tím, že časová složitost algoritmu FIND-MAX je nějaká funkce tvaru

$$T(n) = an + b,$$

kde by nás ale nezajímaly konkrétní hodnoty konstant a a b, celá analýza mohla být výrazně jednodušší.

- Ve skutečnosti ani většinou nechceme vědět, jak přesně funkce $T(n)$ vypadá (obecně to může být nějaká velmi komplikovaná funkce), a stačilo by nám, že víme, že hodnoty funkce $T(n)$ „zhruba“ odpovídají hodnotám nějaké funkce $S(n) = an + b$, kde a a b jsou nějaké konstanty.

Časová složitost algoritmu

U dané funkce $T(n)$ vyjadřující časovou nebo paměťovou složitost se tak většinou spokojíme s jejím přibližným vyjádřením — **odhadem**, kde

- zanedbáme méně významné členy

  (např. ve funkci $T(n) = 15n^2 + 40n - 5$ zanedbáme členy $40n$ a -5 a místo původní funkce budeme uvažovat jen o funkci $T(n) = 15n^2$),

- zanedbáme konstanty, kterými se násobí

  (např. místo funkce $T(n) = 15n^2$ budeme uvažovat o funkci $T(n) = n^2$)

- konstanty v exponentech ignorovat nebudeme — například je podstatný rozdíl mezi funkcemi $T_1(n) = n^2$ a $T_2(n) = n^3$.

- bude nás zajímat, jak se funkce $T(n)$ chová pro „velké“ hodnoty n, chování na malých hodnotách budeme ignorovat

Rychlost růstu funkcí

Program zpracovává vstup velikosti n.
Předpokládejme, že pro vstup velikosti n provede $T(n)$ operací, a že provedení jedné operace trvá $1\,\mu\mathrm{s}$ ($10^{-6}\,\mathrm{s}$).

	n							
$T(n)$	20	40	60	80	100	200	500	1000
n	20 µs	40 µs	60 µs	80 µs	0.1 ms	0.2 ms	0.5 ms	1 ms
$n \log n$	86 µs	0.213 ms	0.354 ms	0.506 ms	0.664 ms	1.528 ms	4.48 ms	9.96 ms
n^2	0.4 ms	1.6 ms	3.6 ms	6.4 ms	10 ms	40 ms	0.25 s	1 s
n^3	8 ms	64 ms	0.216 s	0.512 s	1 s	8 s	125 s	16.7 min.
n^4	0.16 s	2.56 s	12.96 s	42 s	100 s	26.6 min.	17.36 hod.	11.57 dní
2^n	1.05 s	12.75 dní	36560 let	$38.3 \cdot 10^9$ let	$40.1 \cdot 10^{15}$ let	$50 \cdot 10^{45}$ let	$10.4 \cdot 10^{136}$ let	–
$n!$	77147 let	$2.59 \cdot 10^{34}$ let	$2.64 \cdot 10^{68}$ let	$2.27 \cdot 10^{105}$ let	$2.96 \cdot 10^{144}$ let	–	–	–

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $T_1(n) = n$, $T_2(n) = n^3$, $T_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

Složitost	Velikost vstupu
$T_1(n) = n$	10^{12}
$T_2(n) = n^3$	10^4
$T_3(n) = 2^n$	40

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $T_1(n) = n$, $T_2(n) = n^3$, $T_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

Složitost	Velikost vstupu
$T_1(n) = n$	10^{12}
$T_2(n) = n^3$	10^4
$T_3(n) = 2^n$	40

Nyní počítač 1000 násobně zrychlíme. Zvládne tedy 10^{15} kroků.

Složitost	Velikost vstupu	Nárust
$T_1(n) = n$	10^{15}	$1000\times$
$T_2(n) = n^3$	10^5	$10\times$
$T_3(n) = 2^n$	50	$+10$

Asymptotická notace

V následujícím se zaměříme na funkce typu $f : \mathbb{N} \to \mathbb{R}$, kde:

- Hodnota $f(n)$ nemusí být definovaná pro všechny hodnoty $n \in \mathbb{N}$, ale musí existovat nějaká konstanta n_0 taková, že hodnota $f(n)$ je definovaná pro všechna $n \in \mathbb{N}$ taková, že $n \geq n_0$.

  Příklad: Funkce $f(n) = \log_2(n)$ není definovaná pro $n = 0$, ale pro všechna $n \geq 1$ už definovaná je.

- Musí existovat taková konstanta n_0, že pro všechny hodnoty $n \in \mathbb{N}$, kde $n \geq n_0$, platí $f(n) \geq 0$.

  Příklad: Pro funkci $f(n) = n^2 - 25$ platí $f(n) \geq 0$ pro všechna $n \geq 5$.

Asymptotická notace

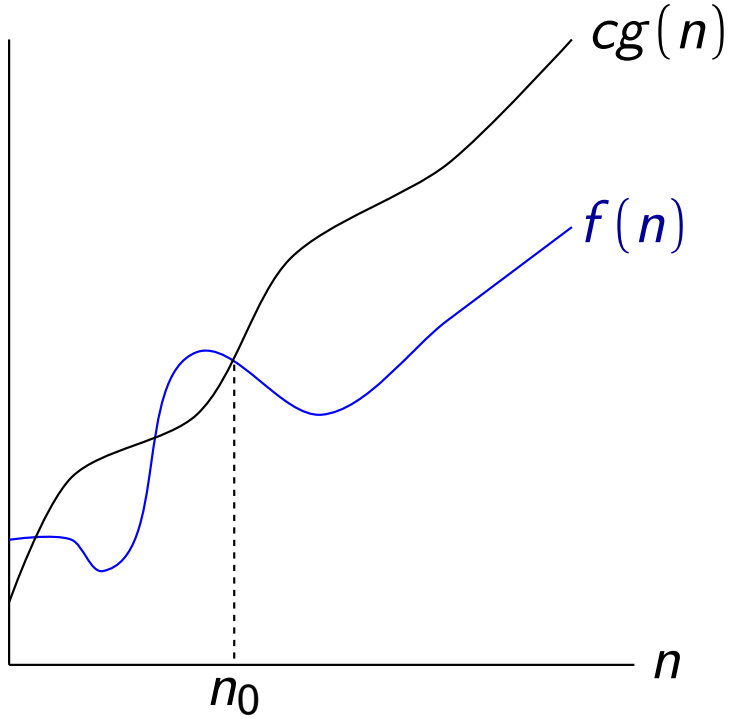
Vezměme si libovolnou funkci $f : \mathbb{N} \to \mathbb{R}$. Zápisy $O(f)$, $\Omega(f)$ a $\Theta(f)$ označují **množiny funkcí** typu $\mathbb{N} \to \mathbb{R}$, kde:

- $O(f)$ – množina všech funkcí, které rostou nejvýše tak rychle jako f
- $\Omega(f)$ – množina všech funkcí, které rostou alespoň tak rychle jako f
- $\Theta(f)$ – množina všech funkcí, které rostou stejně rychle jako f

Poznámka: Toto nejsou definice! Ty následují na následujících slidech.

- O – velké „O“
- Ω – velké řecké písmeno „omega“
- Θ – velké řecké písmeno „theta“

Asymptotická notace – symbol O



Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{R}$. Pro funkci $f : \mathbb{N} \to \mathbb{R}$ platí $f \in O(g)$ právě tehdy, když

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(f(n) \leq c\, g(n)\big).$$

Asymptotická notace – symbol O

Poznámky:

- c je kladné reálné číslo (tj. $c \in \mathbb{R}$ a $c > 0$)
- n_0 a n jsou přirozená čísla (tj. $n_0 \in \mathbb{N}$ a $n \in \mathbb{N}$)

Asymptotická notace – symbol O

Příklad: Vezměme si funkce $f(n) = 2n^2 + 3n + 7$ a $g(n) = n^2$.

Chceme ukázat $f \in O(g)$, tj. $f \in O(n^2)$:

- Postup 1:

  Zvolme například $c = 3$.

  $$cg(n) = 3n^2 = 2n^2 + \tfrac{1}{2}n^2 + \tfrac{1}{2}n^2$$

  Potřebujeme najít takové n_0, aby pro každé $n \geq n_0$ platilo současně

  $$2n^2 \geq 2n^2 \qquad \tfrac{1}{2}n^2 \geq 3n \qquad \tfrac{1}{2}n^2 \geq 7$$

  Snadno ověříme, že například $n_0 = 6$ vyhovuje těmto požadavkům.

  Pak pro každé $n \geq 6$ platí $cg(n) \geq f(n)$:

  $$cg(n) = 3n^2 = 2n^2 + \tfrac{1}{2}n^2 + \tfrac{1}{2}n^2 \geq 2n^2 + 3n + 7 = f(n)$$

Asymptotická notace – symbol O

Příklad, kde $f(n) = 2n^2 + 3n + 7$ a $g(n) = n^2$:

- Postup 2:

  Zvolme $c = 12$.

  $$cg(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2$$

  Potřebujeme najít takové n_0, aby pro každé $n \geq n_0$ platilo současně

  $$2n^2 \geq 2n^2 \qquad 3n^2 \geq 3n \qquad 7n^2 \geq 7$$

  Uvedené vztahy zjevně platí pro $n_0 = 1$, takže pro každé $n \geq 1$ platí $cg(n) \geq f(n)$:

  $$cg(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2 \geq 2n^2 + 3n + 7 = f(n)$$

Asymptotická notace – symbol O

Tvrzení

Předpokládejme, že a a b jsou nějaké konstanty takové, že $a > 0$ a $b > 0$, a k a ℓ jsou nějaké libovolné konstanty, kde $k \geq 0$, $\ell \geq 0$ a $k < \ell$.

Uvažujme funkce

$$f(n) = a \cdot n^k \qquad\qquad g(n) = b \cdot n^\ell$$

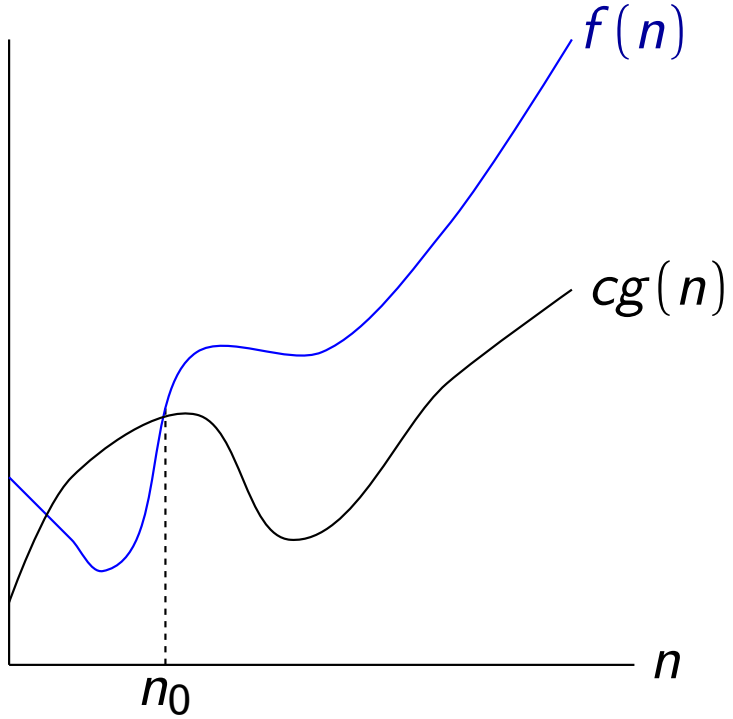
Pro každé takové funkce f a g platí $f \in O(g)$:

Důkaz: Zvolme $c = \frac{a}{b}$.

Vzhledem k tomu, že pro $n \geq 1$ zjevně platí $n^k \leq n^\ell$ (protože $k \leq l$), tak pro $n \geq 1$ platí

$$c \cdot g(n) = \frac{a}{b} \cdot g(n) = \frac{a}{b} \cdot b \cdot n^\ell = a \cdot n^\ell \geq a \cdot n^k = f(n)$$

Asymptotická notace – symbol Ω



Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{R}$. Pro funkci $f : \mathbb{N} \to \mathbb{R}$ platí $f \in \Omega(g)$ právě tehdy, když

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(c\, g(n) \leq f(n)\big).$$

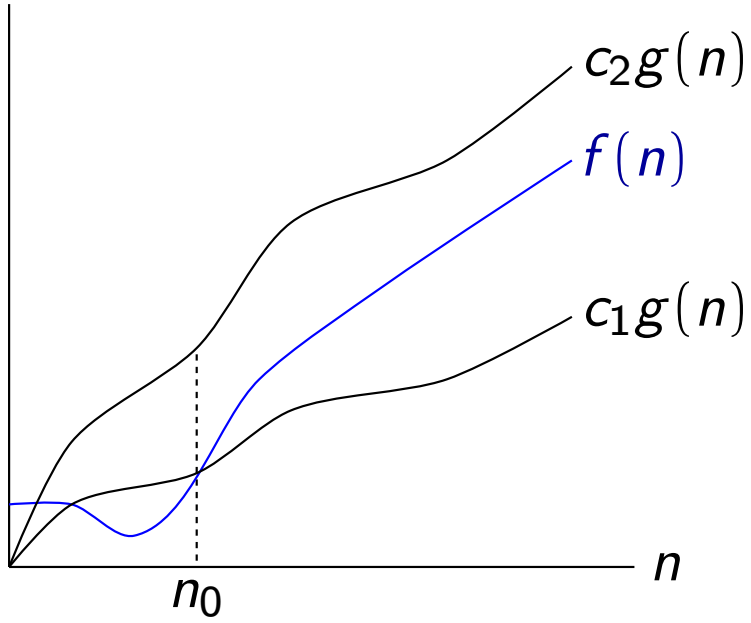
Asymptotická notace – symbol Ω

Není těžké zdůvodnit, že platí následující tvrzení:

Pro libovolné funkce f a g platí:

$$f \in O(g) \qquad \text{právě tehdy, když} \qquad g \in \Omega(f)$$

Asymptotická notace – symbol Θ



Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{R}$. Pro funkci $f : \mathbb{N} \to \mathbb{R}$ platí $f \in \Theta(g)$ právě tehdy, když

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)\big(c_1\, g(n) \leq f(n) \leq c_2\, g(n)\big).$$

Asymptotická notace – symbol Θ

Pro libovolné funkce f a g platí:

$$f \in \Theta(g) \quad \text{právě tehdy, když} \quad f \in O(g) \text{ a } f \in \Omega(g)$$

$$f \in \Theta(g) \quad \text{právě tehdy, když} \quad f \in O(g) \text{ a } g \in O(f)$$

$$f \in \Theta(g) \quad \text{právě tehdy, když} \quad g \in \Theta(f)$$

Asymptotická notace

Pro libovolné tři funkce f, g a h platí:

- jestliže $f \in O(g)$ a $g \in O(h)$, pak $f \in O(h)$
- jestliže $f \in \Omega(g)$ a $g \in \Omega(h)$, pak $f \in \Omega(h)$
- jestliže $f \in \Theta(g)$ a $g \in \Theta(h)$, pak $f \in \Theta(h)$

Asymptotická notace

Příklady:

$n \in O(n^2)$
$1000n \in O(n)$
$2^{\log_2 n} \in \Theta(n)$
$n^3 \notin O(n^2)$
$n^2 \notin O(n)$
$n^3 + 2^n \notin O(n^2)$

$n^3 \in O(n^4)$
$0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2)$
$n^3 - n^2 \log_2^3 n + 1000n - 10^{100} \in \Theta(n^3)$
$n^3 + 1000n - 10^{100} \in O(n^3)$
$n^3 + n^2 \notin \Theta(n^2)$
$n! \notin O(2^n)$

Asymptotická notace

- Existují dvojice funkcí f a g takové, že

$$f \notin O(g) \qquad \text{a} \qquad g \notin O(f),$$

  například

$$f(n) = n \qquad\qquad g(n) = n^{1+\sin(n)}.$$

- $O(1)$ označuje množinu všech **omezených** funkcí, tj. funkcí jejichž funkční hodnoty jsou shora omezeny nějakou konstantou.

Asymptotická notace

- Pro libovolné dvě funkce f, g platí:
  - $\max(f, g) \in \Theta(f + g)$
  - pokud $f \in O(g)$, pak $f + g \in \Theta(g)$
- Pro libovolné čtyři funkce f_1, f_2, g_1, g_2 platí:
  - pokud $f_1 \in O(f_2)$ a $g_1 \in O(g_2)$, pak $f_1 + g_1 \in O(f_2 + g_2)$ a $f_1 \cdot g_1 \in O(f_2 \cdot g_2)$
  - pokud $f_1 \in \Theta(f_2)$ a $g_1 \in \Theta(g_2)$, pak $f_1 + g_1 \in \Theta(f_2 + g_2)$ a $f_1 \cdot g_1 \in \Theta(f_2 \cdot g_2)$

Asymptotická notace

- O funkci f řekneme, že je:

  logaritmická, pokud $f(n) \in \Theta(\log n)$
  lineární, pokud $f(n) \in \Theta(n)$
  kvadratická, pokud $f(n) \in \Theta(n^2)$
  kubická, pokud $f(n) \in \Theta(n^3)$
  polynomiální, pokud $f(n) \in O(n^k)$ pro nějaké $k > 0$
  exponenciální, pokud $f(n) \in O(c^{n^k})$ pro nějaké $c > 1$ a $k > 0$

- Exponenciální funkce se v asymptotické notaci často uvádí ve tvaru $2^{O(n^k)}$, protože potom již nemusíme uvažovat různé základy mocniny.

Asymptotická notace

Jak bylo uvedeno, výrazy $O(g)$, $\Omega(g)$ a $\Theta(g)$ označují určité množiny funkcí.

V odborných textech se však někdy používají tyto výrazy i v poněkud odlišném významu:

- zápis $O(g)$, $\Omega(g)$ nebo $\Theta(g)$ nereprezentuje danou množinu funkcí, ale **nějakou** funkci z dané množiny.

Tato konvence se používá zejména v zápisu rovnic nebo nerovnic.

Příklad: $3n^3 + 5n^2 - 11n + 2 = 3n^3 + O(n^2)$

Při použití této konvence je tedy možné například psát $f = O(g)$ místo $f \in O(g)$.

Složitost algoritmů

Řekněme, že bychom chtěli analyzovat časovou složitost $T(n)$ nějakého algoritmu, který se skládá z instrukcí $I_1, I_2, \ldots, I_k$:

- Pokud $m_1, m_2, \ldots, m_k$ jsou počty provedení jednotlivých instrukcí pro nějaký vstup w (tj. pro vstup x se instrukce I_i provede m_i krát), tak celkový počet instrukcí provedených pro vstup w je
$$T(n) = c_1 m_1 + c_2 m_2 + \cdots + c_k m_k.$$
- Vezměme si funkce $f_1, f_2, \ldots, f_k$, kde $f_i : \mathbb{N} \to \mathbb{R}$, přičemž $f_i(n)$ je maximum z počtu provedení instrukce I_i pro všechny vstupy velikosti n.
- Zjevně platí, že pro libovolnou funkci f_i je $T \in \Omega(f_i)$.
- Zjevně také platí $T \in O(f_1 + f_2 + \cdots + f_k)$.

Složitost algoritmů

- Připomeňme si, že pokud $f \in O(g)$, pak $f + g \in O(g)$.
- Pokud tedy pro některou funkci f_i platí, že pro všechny f_j, kde $j \neq i$, je $f_j \in O(f_i)$, pak

$$T \in O(f_i).$$

- Často se tedy při analýze celkové časové složitosti $T(n)$ můžeme omezit pouze na analýzu počtu provedení nejčastěji prováděné instrukce (pro vstup velikosti n je provedena maximálně $f_i(n)$ krát), protože platí

$$T \in \Theta(f_i).$$

Složitost algoritmů

Příklad: Při analýze složitosti algoritmu FIND-MAX jsme zjistili, že časová složitost daného algoritmu v nejhorším případě je

$$f(n) = an + b\,.$$

Kdybychom to nechtěli takto podrobně zjišťovat a spokojili se s hrubším odhadem, mohli jsme určit, že časová složitost tohoto algoritmu je $\Theta(n)$, protože:

- Algoritmus obsahuje jediný cyklus, který se pro vstup velikosti n provede vždy právě $(n-1)$ krát, tj. počet průchodů cyklem je v $\Theta(n)$.
- V rámci jednoho průchodu cyklem se provede několik instrukcí, jejichž počet je shora i zdola omezen nějakými konstantami nezávislými na velikosti vstupu.
- Ostatní instrukce se provedou jednou. K celkové složitosti tak přispívají přičtením nějaké konstanty.

Složitost algoritmů

Pokusme se analyzovat časovou složitost následujícího algoritmu:

Algoritmus 2: Třídění přímým vkládáním

```
1  INSERTION-SORT (A, n):
2  begin
3      for j := 1 to n − 1 do
4          x := A[j]
5          i := j − 1
6          while i ≥ 0 and A[i] > x do
7              A[i + 1] := A[i]
8              i := i − 1
9          end
10         A[i + 1] := x
11     end
12 end
```

Tj. chceme najít funkci $T(n)$ takovou, že časová složitost algoritmu INSERTION-SORT v nejhorším případě je v $\Theta(T(n))$.

Složitost algoritmů

Uvažujme vstupy velikosti n:

- Vnější cyklus **for** se provede $n-1$ krát.
- Vnitřní cyklus **while** se pro danou hodnotu j provede maximálně $(j-1)$ krát.
- Existují vstupy, pro které platí že pro každou hodnotu j od 2 do n se vnitřní cyklus **while** provede právě $(j-1)$ krát.
- V nejhorším případě se tedy cyklus **while** provede celkem m krát, kde

$$m = 1 + 2 + \cdots + (n-1) = (1 + (n-1)) \cdot \tfrac{n-1}{2} = \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

- Celková časová složitost algoritmu INSERTION-SORT v nejhorším případě je tedy $\Theta(n^2)$.

Složitost algoritmů

V předchozím případě jsme přesně spočítali celkový počet průchodů cyklem **while**.

Obecně to není vždy možné spočítat takto přesně nebo to může být hodně komplikované. Pokud nás zajímá jen asymptotický odhad, tak to často ani není nutné.

Složitost algoritmů

Pokud bychom například neuměli spočítat součet aritmetické posloupnosti, mohli bychom provést analýzu následovně:

- Vnější cyklus **for** se neprovede více než n krát, vnitřní cyklus **while** se při každé iteraci vnějšího cyklu provede maximálně n krát. Celkově se tedy vnitřní cyklus provede maximálně n^2 krát.

  Platí tedy $T \in O(n^2)$.

- Pro některé vstupy se při posledních $\lfloor n/2 \rfloor$ průchodech cyklem **for** provede cyklus **while** alespoň $\lceil n/2 \rceil$ krát.

  Pro některé vstupy se tedy cyklus **while** provede alespoň $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$ krát.

  $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \geq (n/2 - 1) \cdot (n/2) = \frac{1}{4}n^2 - \frac{1}{2}n$

  Platí tedy $T \in \Omega(n^2)$.

Prostorová (paměťová) složitost algoritmů

- Zatím jsme se zajímali o čas, který potřebujeme k výpočtu
- Někdy bývá kritickou velikost paměti potřebné k provedení výpočtu.

Množství paměti použité strojem $\mathcal{M}$ při provádění výpočtu nad vstupem w může být např.:

- maximální počet bitů nutných pro uložení všech dat v každé jednotlivé konfiguraci
- maximální počet paměťových buněk použitých během výpočtu

Definice

Prostorová složitost algoritmu Alg běžícího na stroji $\mathcal{M}$ je funkce $S : \mathbb{N} \to \mathbb{N}$, kde $S(n)$ udává maximální množství paměti použité strojem $\mathcal{M}$ při výpočtech nad vstupy velikosti n.

Prostorová (paměťová) složitost algoritmů

- Pro konkrétní problém můžeme mít dva algoritmy takové, že jeden má menší prostorovou složitost a druhý zase časovou složitost.
- Je-li časová složitost algoritmu v $O(f(n))$ je i prostorová v $O(f(n))$ (počet použitých paměťových buněk nemůže být větší než počet provedených operací, protože v každém kroku se použije jen nějaký omezený počet buněk (omezený nějakou konstantou nezávislou na velikosti vstupu).
- Prostorová složitost může být mnohdy o dost menší než časová složitost — například paměťová složitost algoritmu Insertion-Sort je $\Theta(n)$, zatímco časová $\Theta(n^2)$.

Složitost algoritmů

Orientační typické hodnoty velikosti vstupu n, pro které algoritmus s danou časovou složitostí ještě většinou zvládne na „běžném PC“ spočítat výsledek ve zlomku sekundy nebo maximálně v řádu sekund.

(Závisí to samozřejmě výrazně na konkrétních detailech. Navíc se zde předpokládá, že v asymptotické notaci nejsou skryty nějaké velké konstanty.)

$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$
$1\,000\,000 - 100\,000\,000$	$100\,000 - 1\,000\,000$	$1000 - 10\,000$	$100 - 1000$

$2^{O(n)}$	$O(n!)$
$20 - 30$	$10 - 15$

Složitost algoritmů

Při používání asymptotických odhadů časové složitosti algoritmů bychom si měli být vědomi některých úskalí:

- Asyptotické odhady se týkají pouze toho, jak roste čas s rostoucí velikostí vstupu.
- Neříkají nic o konkrétní době výpočtu. V asymptotické notaci mohou být skryty velké konstanty.
- Algoritmus, který má lepší asymptotickou časovou složitost než nějaký jiný algoritmus, může být ve skutečnosti rychlejší až pro nějaké hodně velké vstupy.
- Většinou analyzujeme složitost v nejhorším případě. Pro některé algoritmy může být doba výpočtu v nejhorším případě mnohem větší než doba výpočtu na „typických“ instancích.

Složitost algoritmů

- Můžeme si to ilustrovat na algoritmech pro třídění.

Algoritmus	Nejhorší případ	Průměrný případ
Bubblesort	$\Theta(n^2)$	$\Theta(n^2)$
Heapsort	$\Theta(n \log n)$	$\Theta(n \log n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \log n)$

- Quicksort má horší asymptotickou složitost v nejhorším případě než Heapsort, stejnou asymptotickou složitost v průměrném případě a přesto je v praxi nejrychlejší.

Složitost algoritmů

Polynom — funkce tvaru

$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n + a_0$$

kde $a_0, a_1, \ldots, a_k$ jsou konstanty.

Příklady polynomů:

$$4n^3 - 2n^2 + 8n + 13 \qquad 2n + 1 \qquad n^{100}$$

Funkce f je **polynomiální**, jestliže je shora omezena nějakým polynomem, tj. jestliže existuje nějaká konstanta k taková, že $f \in O(n^k)$.

Polynomiální jsou například funkce, které patří do následujících tříd:

$$O(n) \qquad O(n \log n) \qquad O(n^2) \qquad O(n^5) \qquad O(\sqrt{n}) \qquad O(n^{100})$$

Složitost algoritmů

Funkce jako 2^n nebo $n!$ polynomiální nejsou — pro libovolně velkou konstantu k platí

$$2^n \in \Omega(n^k) \qquad\qquad n! \in \Omega(n^k)$$

Polynomiální algoritmus — algoritmus, jehož časová složitost je polynomiální (tj. shora omezená nějakým polynomem)

Zhruba se dá říct, že:

- polynomiální algoritmy jsou efektivní algoritmy, které se dají prakticky použít i pro relativně velké vstupy
- algoritmy, které polynomiální nejsou, se dají použít jen pro poměrně malé vstupy

Složitost algoritmů

Rozdělení na polynomiální a nepolynomiální algoritmy je velmi hrubé — nelze kategoricky tvrdit, že polynomiální algoritmy jsou vždy prakticky použitelné a nepolynomiální naopak nikdy nejsou:

- algoritmus se složitostí $\Theta(n^{100})$ pravděpodobně příliš prakticky použitelný nebude,
- některé algoritmy, které nejsou polynomiální, mohou fungovat efektivně pro velkou část vstupů, a složitost větší než polynomiální mají jen kvůli některým problematickým vstupům, na kterých může výpočet trvat velmi dlouhou dobu.

Poznámka: Polynomiální algoritmy, kde by konstanta v exponentu bylo nějaké velké číslo (např. algoritmy se složitostí $\Theta(n^{100})$), se při řešení běžných algoritmických problémů prakticky nevyskytují.

Složitost algoritmů

Pro většinu běžných algoritmických problémů nastává jedna ze tří možností:

- Je znám polynomiální algoritmus se složitostí $O(n^k)$, kde k je nějaké velmi malé číslo (např. 5 a častěji třeba 3 a méně).

- Není znám žádný polynomiální algoritmus a nejlepší známé algoritmy mají složitosti jako třeba $2^{\Theta(n)}$, $\Theta(n!)$ nebo nějaké ještě větší.

  V některých případech může být znám i důkaz, že pro daný problém žádný polynomiální algoritmus neexistuje (tj. nedá se vytvořit).

- Není znám žádný algoritmus, který řeší daný problém (a případně je i dokázáno, že žádný takový algoritmus neexistuje).

Složitost algoritmů

Typický příklad polynomiálního algoritmu — násobení matic s časovou složitostí $\Theta(n^3)$ a paměťovou složitostí $\Theta(n^2)$:

Algoritmus 3: Násobení matic

```
1  MATRIX-MULT (A, B, C, n):
2  begin
3      for i := 1 to n do
4          for j := 1 to n do
5              x := 0
6              for k := 1 to n do
7                  x := x + A[i][k] * B[k][j]
8              end
9              C[i][j] := x
10         end
11     end
12 end
```

Složitost algoritmů

- Při hrubé analýze složitosti často stačí spočítat počet do sebe vnořených smyček — a tento počet pak udává stupeň polynomu

  Příklad: Tři vnořené cykly při násobení matic — časová složitost algoritmu je $O(n^3)$.

- Pokud neprobíhají všechny smyčky např. od 0 do n, ale počet průchodů vnitřními smyčkami se při různých iteracích vnější smyčky mění, podrobnější analýza může být komplikovanější.

  Většinou to pak vede na počítání součtů různých typů číselných řad (např. aritmetické, geometrické, apod.).

  Často dá taková podrobnější analýza podobný výsledek jako hrubá analýza, mnohdy však může být složitost zjištěná touto podrobnější analýzou podstatně nižší než by vyplývalo z hrubého odhadu.

Aritmetická posloupnost — číselná řada $a_0, a_1, \ldots, a_{n-1}$, kde

$$a_i = a_0 + i \cdot d,$$

kde d je nějaká konstanta nezávistlá na i.

Poznámka: V aritmetické posloupnosti tedy pro všechna i platí $a_{i+1} = a_i + d$.

Součet aritmetické posloupnosti:

$$\sum_{i=0}^{n-1} a_i = a_0 + a_1 + \cdots + a_{n-1} = \frac{1}{2} n \left(a_{n-1} + a_0\right)$$

Příklad:

$$1 + 2 + \cdots + n \;=\; \frac{1}{2}n(n+1) \;=\; \frac{1}{2}n^2 + \frac{1}{2}n \;=\; \Theta(n^2)$$

Konkrétně například pro $n = 100$ je

$$1 + 2 + \cdots + 100 \;=\; 50 \cdot 101 \;=\; 5050.$$

Poznámka: Stačí si umědomit, že

$$1 + 2 + \cdots + 100 \;=\; (1 + 100) + (2 + 99) + \cdots + (50 + 51),$$

kde sčítáme celkem 50 dvojic, kde součet každé dvojice je 101.

Geometrická posloupnost — číselná řada $a_0, a_1, \ldots, a_n$, kde

$$a_i = a_0 \cdot q^i,$$

kde q je nějaká konstanta nezávistlá na i.

Poznámka: V geometrické posloupnosti tedy pro všechna i platí $a_{i+1} = a_i \cdot q$ for each i

Součet geometrické posloupnosti (kde $q \neq 1$):

$$\sum_{i=0}^{n} a_i = a_0 + a_1 + \cdots + a_n = a_0 \frac{q^{n+1} - 1}{q - 1}$$

Příklad:

$$1 + q + q^2 + \cdots + q^n = \frac{q^{n+1} - 1}{q - 1}$$

Speciálně pro $q = 2$:

$$1 + 2^1 + 2^2 + 2^3 + \cdots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2 \cdot 2^n - 1 = \Theta(2^n)$$

Složitost algoritmů

Exponenciální funkce: funkce tvaru c^n, kde c je konstanta — např. funkce 2^n

Logaritmus — inverzní funkce k exponenciální funkci: pro dané n je

$$\log_c n$$

taková hodnota x, že $c^x = n$.

Složitost algoritmů

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072
18	262144
19	524288
20	1048576

n	$\lceil \log_2 n \rceil$
0	—
1	0
2	1
3	2
4	2
5	3
6	3
7	3
8	3
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	4
17	5
18	5
19	5
20	5

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16
131072	17
262144	18
524288	19
1048576	20

Tvrzení

Pro libovolná $a, b > 1$ a libovolné $n > 0$ platí

$$\log_a n = \frac{\log_b n}{\log_b a}$$

Důkaz: Z $n = a^{\log_a n}$ plyne $\log_b n = \log_b(a^{\log_a n})$.
Protože $\log_b(a^{\log_a n}) = \log_a n \cdot \log_b a$, dostáváme $\log_b n = \log_a n \cdot \log_b a$,
z čehož plyne výše uvedený závěr. □

Z toho důvodu se při použití asymptotické notace základ logaritmu obvykle vynechává: například místo $\Theta(n \log_2 n)$ můžeme napsat $\Theta(n \log n)$.

Složitost algoritmů

Příklady toho, kde se při analýze algoritmů objevují exponenciální funkce a logaritmy:

- Nějaká hodnota se opakovaně zmenšuje na polovinu nebo naopak zdvojnásobuje.

  Například u **binárního vyhledávání** (metodou půlení intervalu) se s každou iterací cyklu zmenšuje velikost intervalu na polovinu.

  Předpokládejme, že pole má velikost n.

  Jaká je minimální velikost pole n, při které se provede alespoň k iterací?

  Odpověď: 2^k

  Platí tedy $k = \log_2(n)$. Časová složitost algoritmu je pak $\Theta(\log n)$.

Složitost algoritmů

- Pomocí n bitů je možno reprezentovat čísla od 0 do $2^n - 1$.
- Minimální počet bitů potřebných pro uložení přirozeného čísla x reprezentovaného binárně je

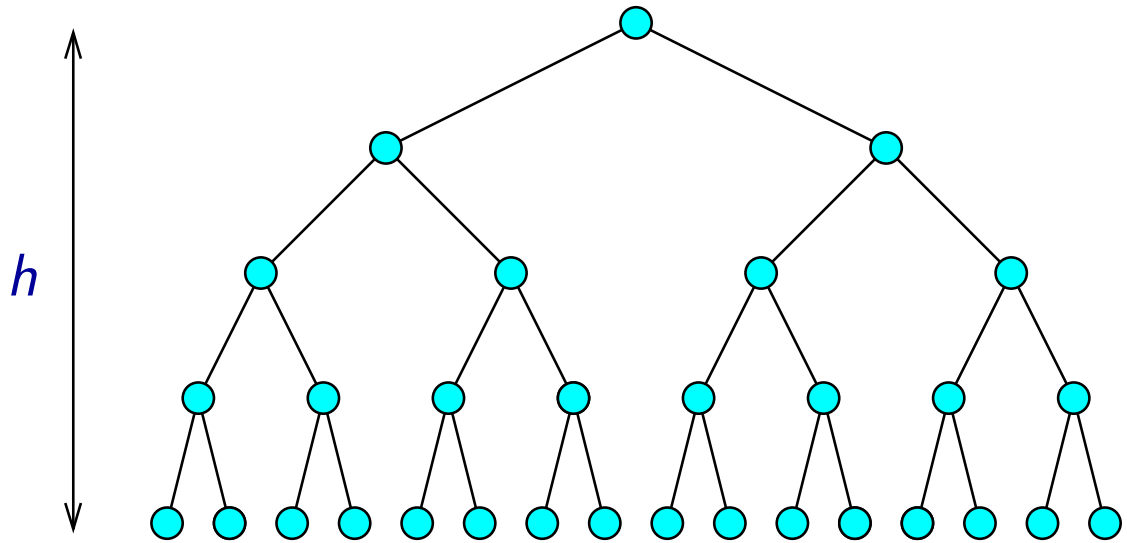
$$\lceil \log_2(x+1) \rceil.$$

- Dokonale vyvážený binární strom o výšce h má $2^{h+1} - 1$ vrcholů, z čehož 2^h jsou listy.
- Dokonale vyvážený binární strom o n vrcholech má výšku zhruba $\log_2 n$.

  Ilustrační příklad: Kdybychom nakreslili vyvážený strom o $n = 1\,000\,000$ vrcholech tak, aby sousední vrcholy byly vzdáleny o 1 cm a výška každé vrstvy byla také 1 cm, měl by tento strom na šířku 10 km a na výšku zhruba 20 cm.

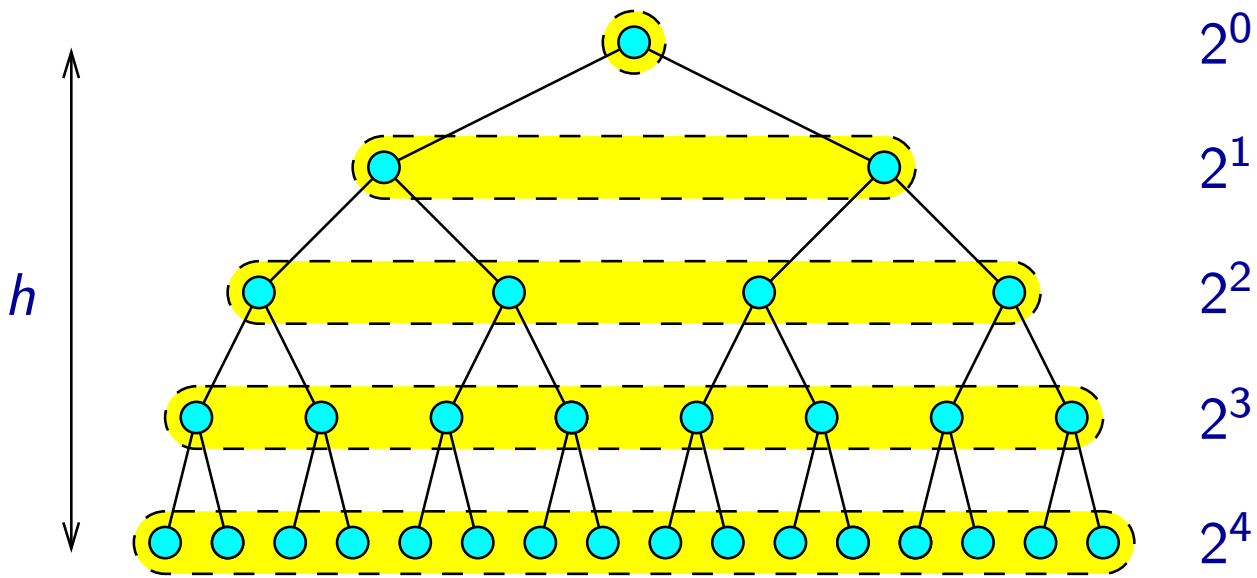
Složitost algoritmů

Dokonale vyvážený binární strom výšky h:



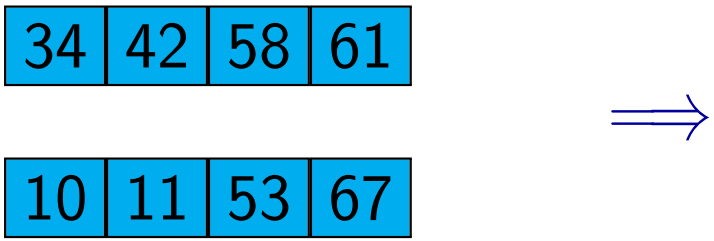
Složitost algoritmů

Dokonale vyvážený binární strom výšky h:



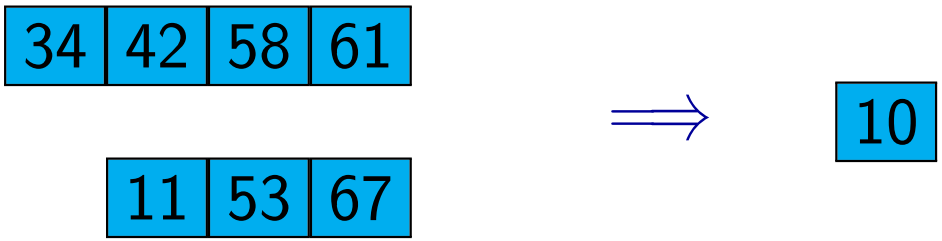
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



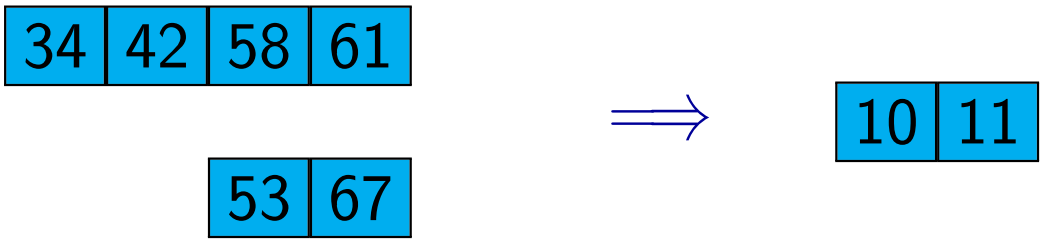
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



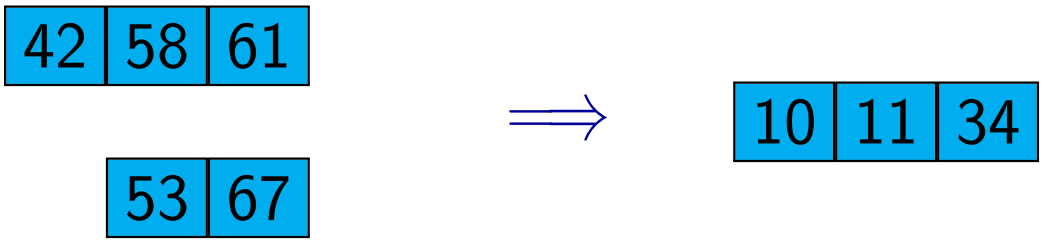
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



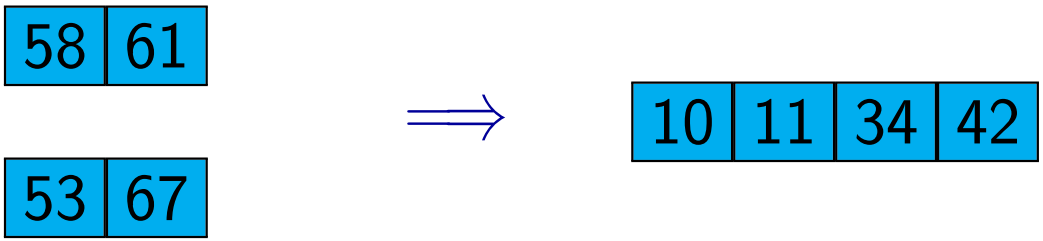
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



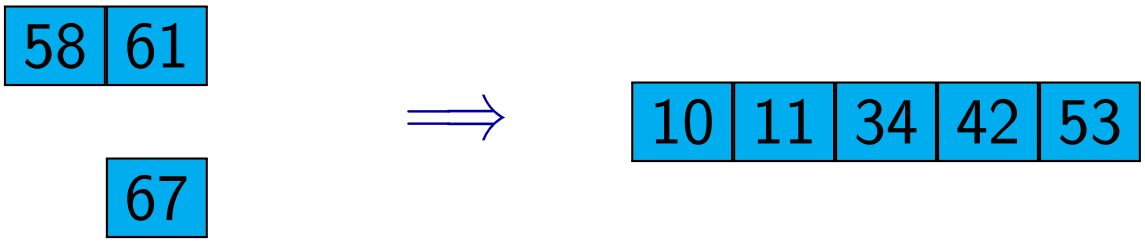
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



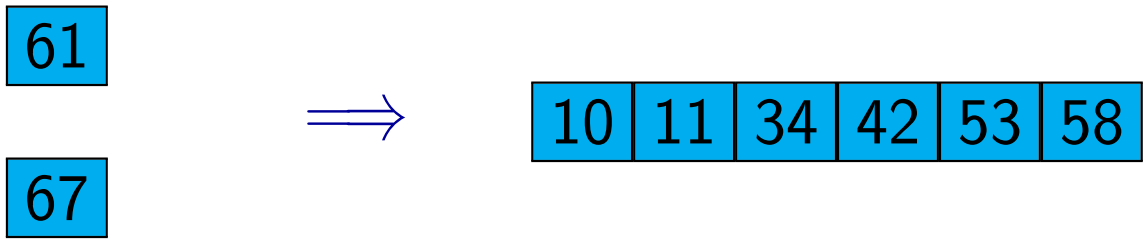
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



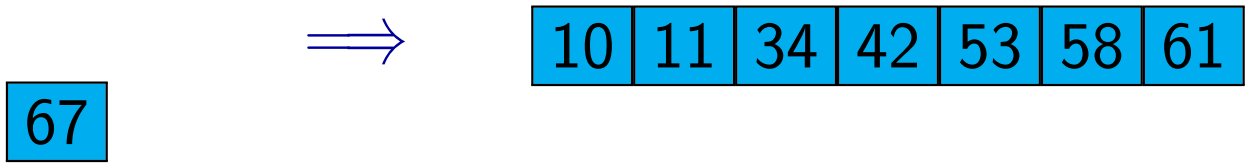
Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

Složitost algoritmů

Algoritmus 4: Merge sort

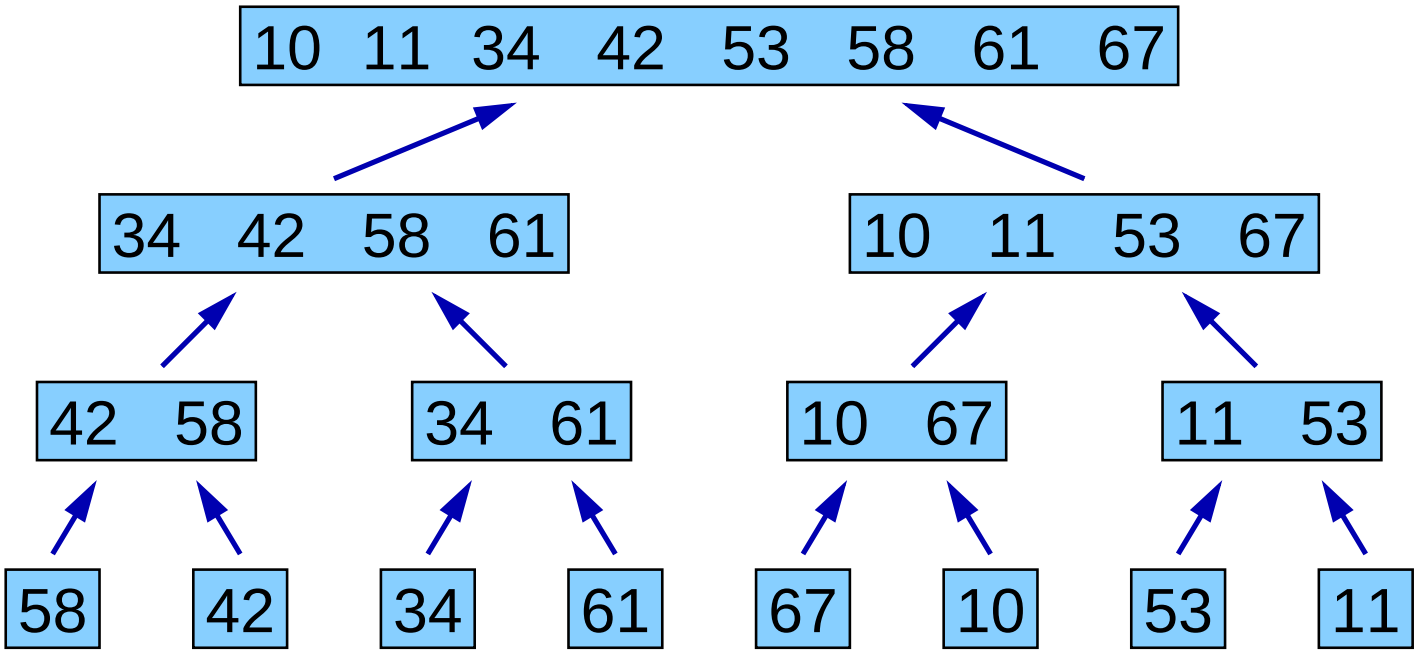
```
1 MERGE-SORT (A, p, r):
2 begin
3     if r − p > 1 then
4         q := ⌊(p + r) / 2⌋
5         MERGE-SORT(A, p, q)
6         MERGE-SORT(A, q, r)
7         MERGE(A, p, q, r)
8     end
9 end
```

Pro setřídění pole A, které obsahuje prvky $A[0], A[1], \cdots, A[n-1]$, zavoláme MERGE-SORT$(A, 0, n)$.

Poznámka: Procedura MERGE(A, p, q, r) spojí setříděné posloupnosti uložené v $A[p \,..\, q-1]$ a $A[q \,..\, r-1]$ do jedné posloupnosti uložené v $A[p \,..\, r-1]$.

Složitost algoritmů

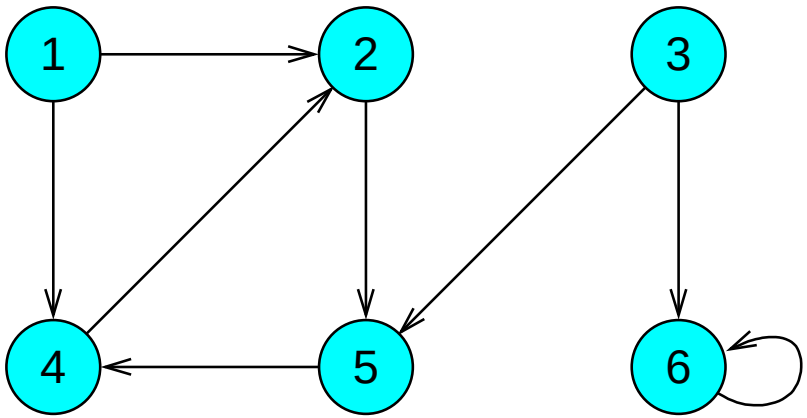
Vstup: 58, 42, 34, 61, 67, 10, 53, 11

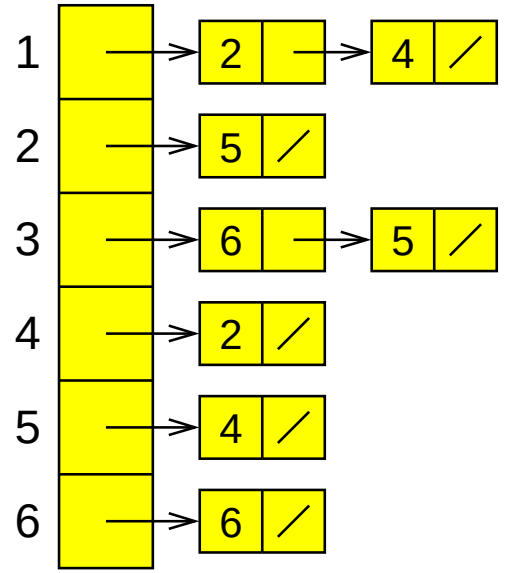


Strom rekurzivních volání má $\Theta(\log n)$ úrovní. Na každé úrovni se provede $\Theta(n)$ operací. Časová složitost algoritmu MERGE-SORT je $\Theta(n \log n)$.

Složitost algoritmů

Reprezentace grafu:

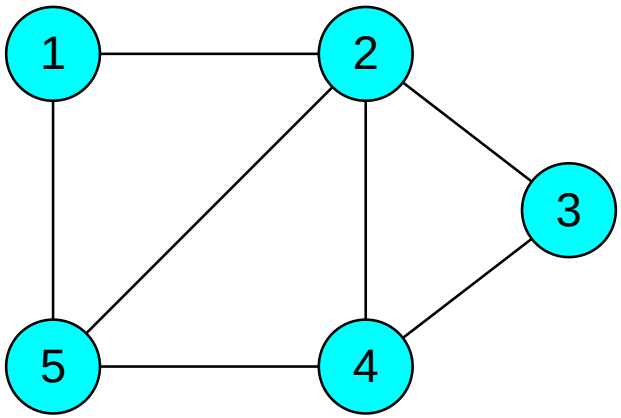


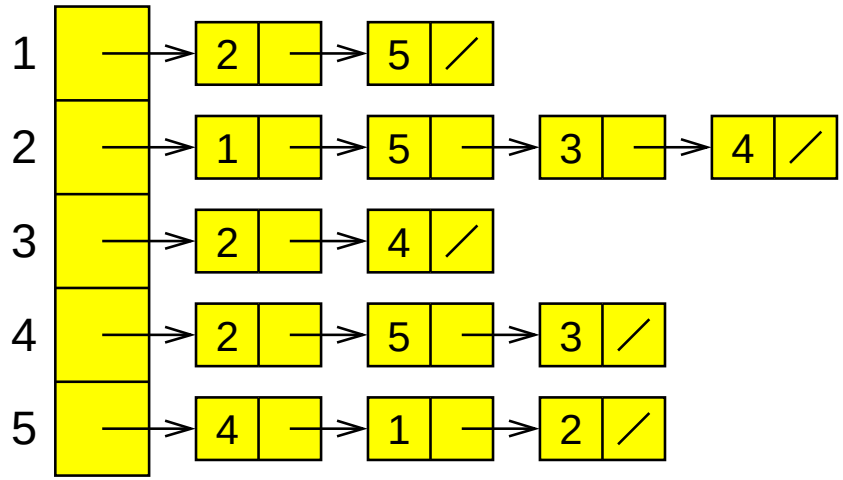


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Složitost algoritmů

Reprezentace grafu:





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Složitost algoritmů

Nalezení nejkratší cesty grafu, kde hrany nejsou ohodnoceny:

- Algoritmus pro prohledávání grafu do šířky
- Vstupem je graf G (s množinou vrcholů V) a počáteční vrchol s.
- Algoritmus pro všechny vrcholy najde nejkratší cestu z vrcholu s.
- Pro graf, který má n vrcholů a m hran je doba výpočtu tohoto algoritmu $\Theta(n+m)$.

Složitost algoritmů

Algoritmus 5: Prohledávání do šířky

```
1  BFS (G, s):
2  begin
3      BFS-INIT(G, s)
4      ENQUEUE(Q, s)
5      while Q ≠ ∅ do
6          u := DEQUEUE(Q)
7          for each v ∈ edges[u] do
8              if color[v] = WHITE then
9                  color[v] := GRAY
10                 d[v] := d[u] + 1
11                 pred[v] := u
12                 ENQUEUE(Q, v)
13             end
14         end
15         color[u] := BLACK
16     end
17 end
```

Složitost algoritmů

Algoritmus 6: Prohledávání do šířky — inicializace — initialization

```
1  BFS-INIT (G, s):
2  begin
3      for each u ∈ V − {s} do
4          color[u] := WHITE
5          d[u] := ∞
6          pred[u] := NIL
7      end
8      color[s] := GRAY
9      d[u] := 0
10     pred[u] := NIL
11     Q := ∅
12 end
```

Složitost algoritmů

- Množina obsahující n prvků má 2^n podmnožin.

  Algoritmus řešící daný problém **hrubou silou**, kdy testuje nějakou podmínku pro všechny podmnožiny dané množiny.

- Pokud by stačilo omezit se na podmnožiny určité konkrétní velikosti k, těchto podmnožin je

$$\binom{n}{k}$$

  Pro některé hodnoty k celkový počet těchto podmnožin není o moc menší než 2^n:

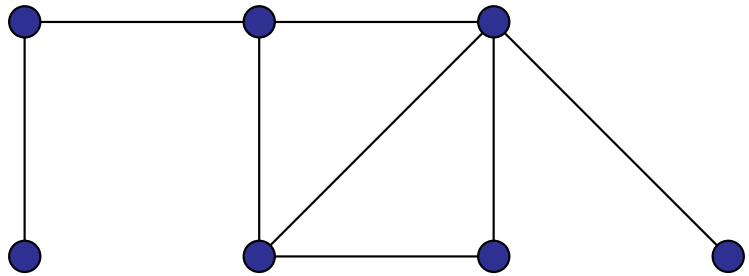
  Dá se například ukázat, že

$$\binom{n}{\lfloor n/2 \rfloor} \geq \frac{2^n}{n}.$$

Problém nezávislé množiny (IS — independent set)

Vstup: Neorientovaný graf G, číslo k.

Otázka: Existuje v grafu G nezávislá množina velikosti k?



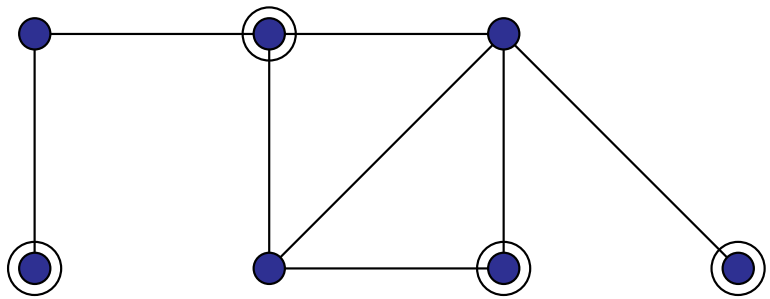
$k = 4$

Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

Problém nezávislé množiny (IS — independent set)

Vstup: Neorientovaný graf G, číslo k.

Otázka: Existuje v grafu G nezávislá množina velikosti k?

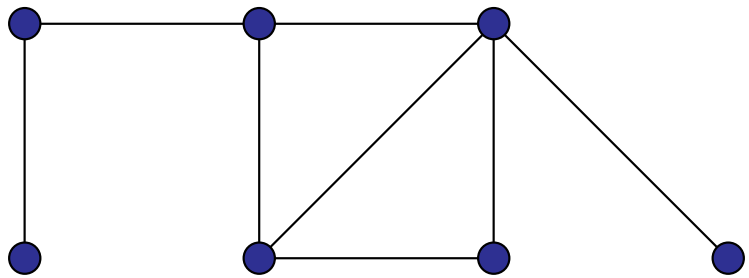


$k = 4$

Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

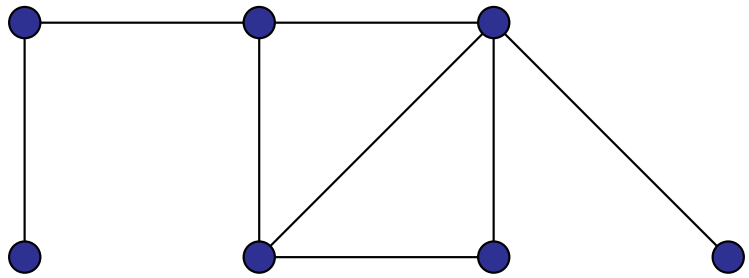
Složitost algoritmů

Příklad instance, kde je odpověď ANO:



$k = 4$

Příklad instance, kde je odpověď NE:



$k = 5$

Složitost algoritmů

Algoritmus řešící problém nezávislé množiny hrubou silou tím způsobem, že bude postupně testovat pro všechny k-prvkové podmnožiny n vrcholů, zda tvoří daná podmnožina nezávislou množinu, bude mít časovou složitost $2^{\Theta(n)}$.

Příklady algoritmických problémů

Na následujích slidech jsou uvedeny příklady několika typických algoritmických problémů, které:

- jsou algoritmicky řešitelné — většinou není těžké vymyslet algoritmus s exponenciální časovou složitostí, řešící daný problém
- není pro ně znám žádný algoritmus s polynomiální časovou složitostí
- na druhou stranu není ani dokázáno, že daný pro daný problém nemůže algoritmus s polynomiální časovou složitostí existovat
- vše jsou to příklady tzv. **NP-úplných** problémů

Problém SAT

SAT (splnitelnost booleovských formulí)

Vstup: Formule výrokové logiky φ.

Otázka: Je φ splnitelná?

Příklad:

Formule $\varphi_1 = p \wedge (\neg q \vee r)$ je splnitelná:
např. při ohodnocení v, kde $v(p) = 1$, $v(q) = 0$, $v(r) = 1$, platí $v \models \varphi_1$.

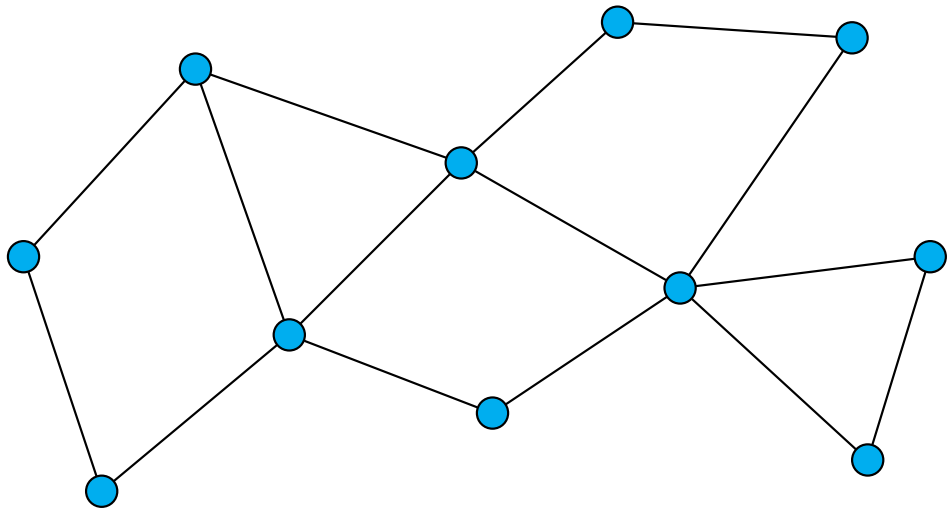
Formule $\varphi_2 = (p \wedge \neg p) \vee (\neg q \wedge r \wedge q)$ není splnitelná:
pro libovolné ohodnocení v platí $v \not\models \varphi_2$.

Barvení grafu

Vstup: Neorientovaný graf G, přirozené číslo k.

Otázka: Lze vrcholy grafu G obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

Příklad: $k = 3$



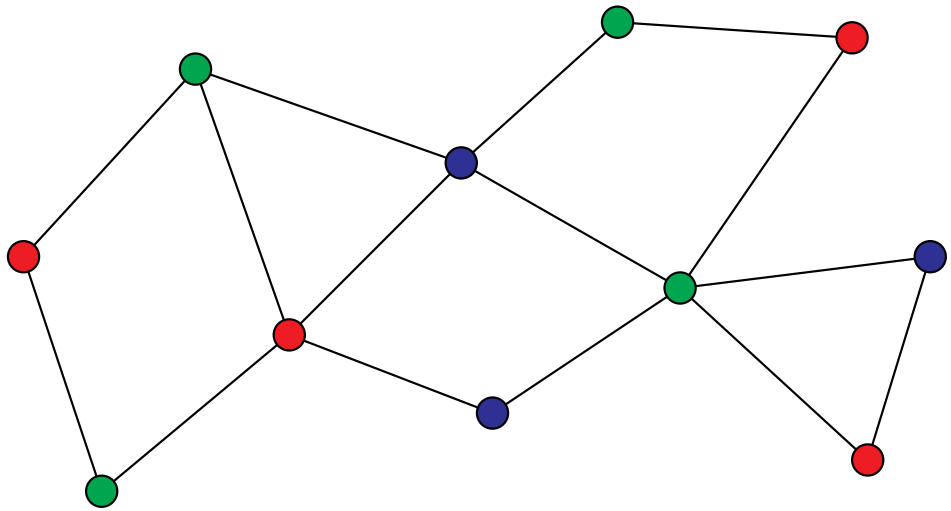
Barvení grafu

Barvení grafu

Vstup: Neorientovaný graf G, přirozené číslo k.

Otázka: Lze vrcholy grafu G obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

Příklad: $k = 3$



Odpověď: Ano

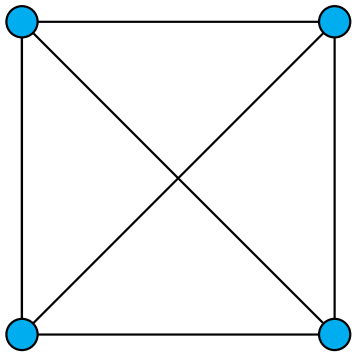
Barvení grafu

Barvení grafu

Vstup: Neorientovaný graf G, přirozené číslo k.

Otázka: Lze vrcholy grafu G obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

Příklad: $k = 3$



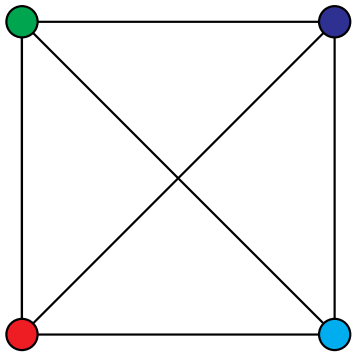
Barvení grafu

Barvení grafu

Vstup: Neorientovaný graf G, přirozené číslo k.

Otázka: Lze vrcholy grafu G obarvit k barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

Příklad: $k = 3$



Odpověď: NE

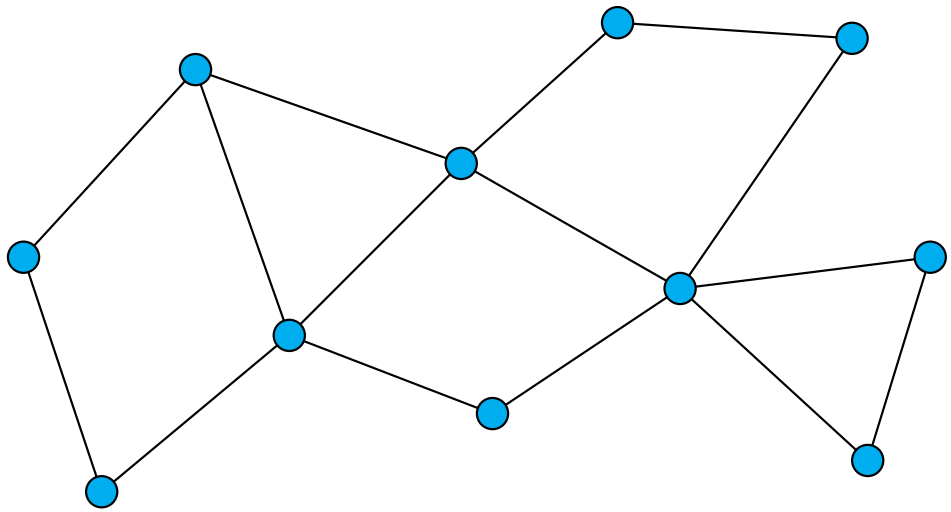
VC – Vrcholové pokrytí

VC – vrcholové pokrytí (vertex cover)

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Existuje v grafu G množina vrcholů velikosti k taková, že každá hrana má alespoň jeden svůj vrchol v této množině?

Příklad: $k = 6$



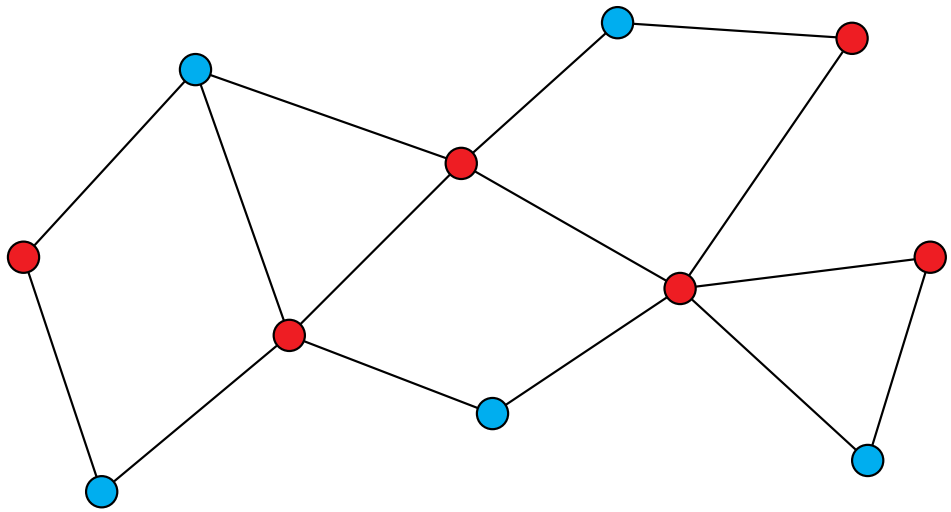
VC – Vrcholové pokrytí

VC – vrcholové pokrytí (vertex cover)

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Existuje v grafu G množina vrcholů velikosti k taková, že každá hrana má alespoň jeden svůj vrchol v této množině?

Příklad: $k = 6$



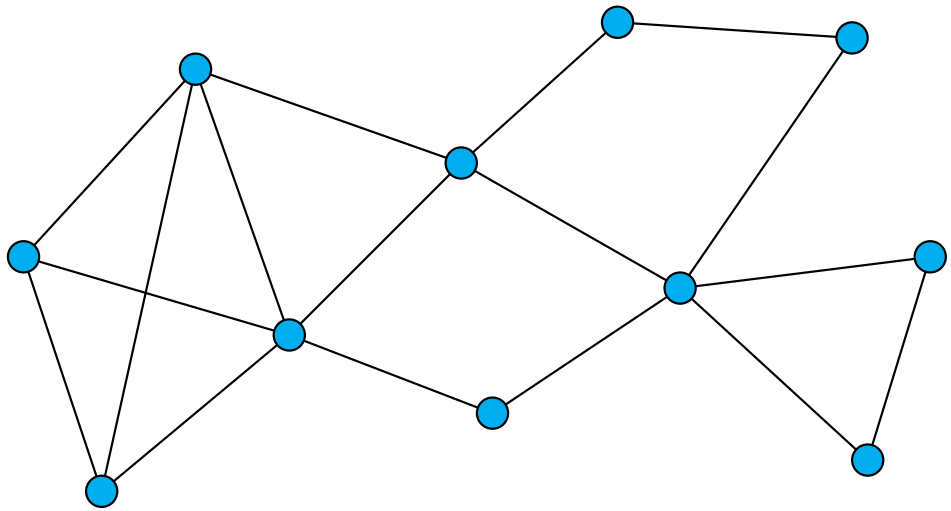
Odpověď: Ano

CLIQUE – problém kliky

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Existuje v grafu G množina vrcholů velikosti k taková, že každé dva vrcholy této množiny jsou spojeny hranou?

Příklad: $k = 4$

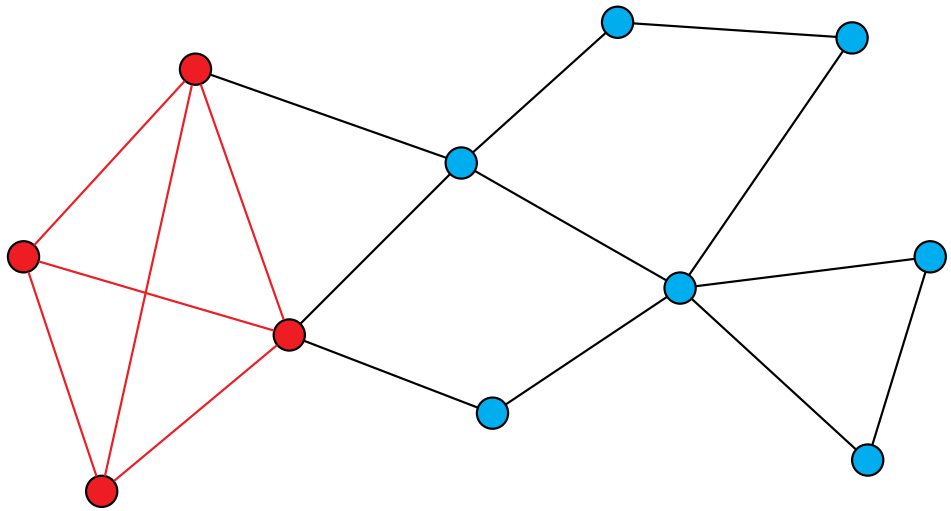


CLIQUE – problém kliky

Vstup: Neorientovaný graf G a přirozené číslo k.

Otázka: Existuje v grafu G množina vrcholů velikosti k taková, že každé dva vrcholy této množiny jsou spojeny hranou?

Příklad: $k = 4$



Odpověď: ANO

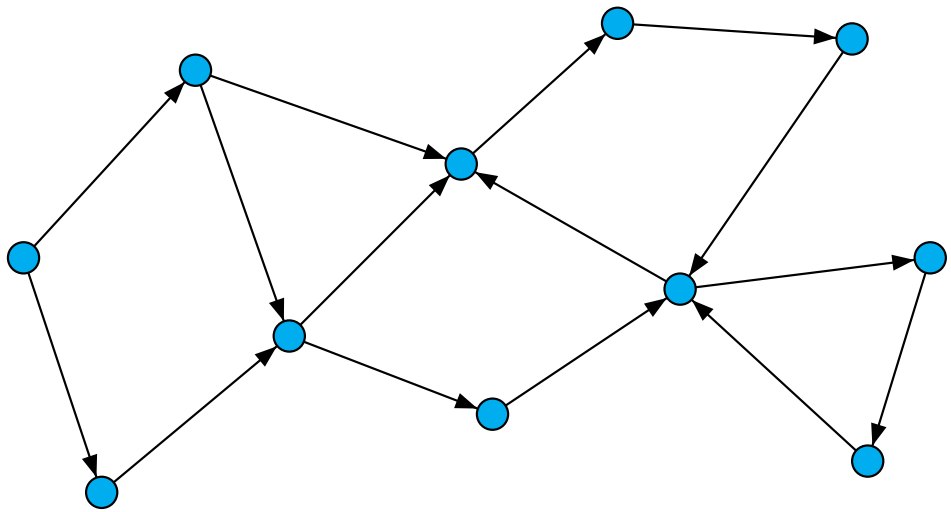
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



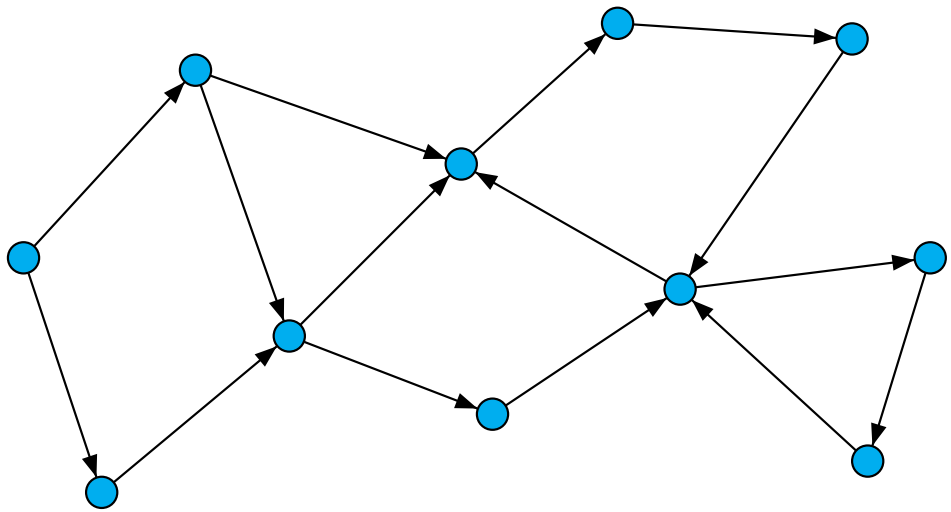
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



Odpověď: NE

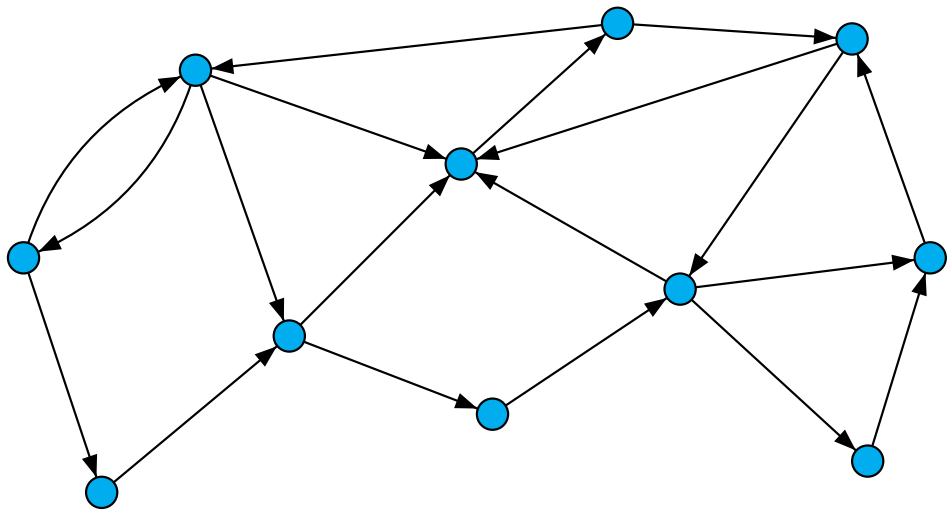
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



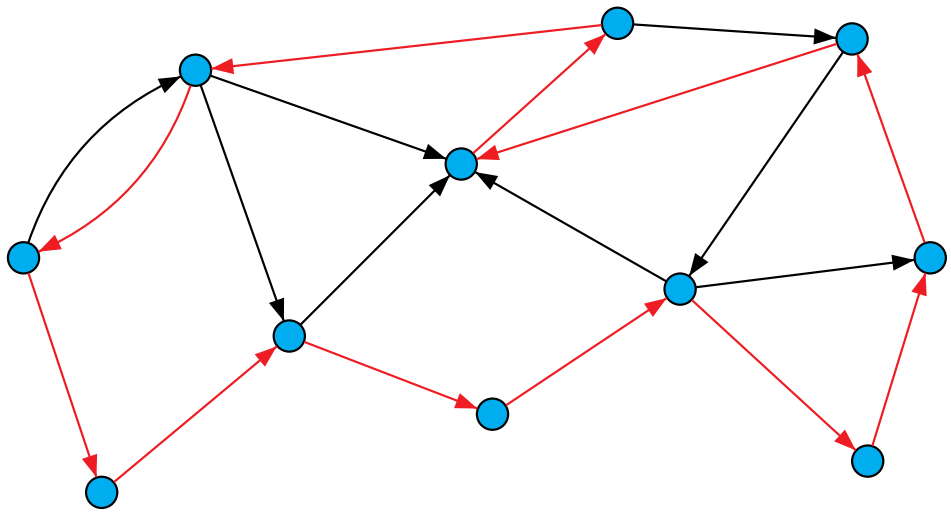
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



Odpověď: Ano

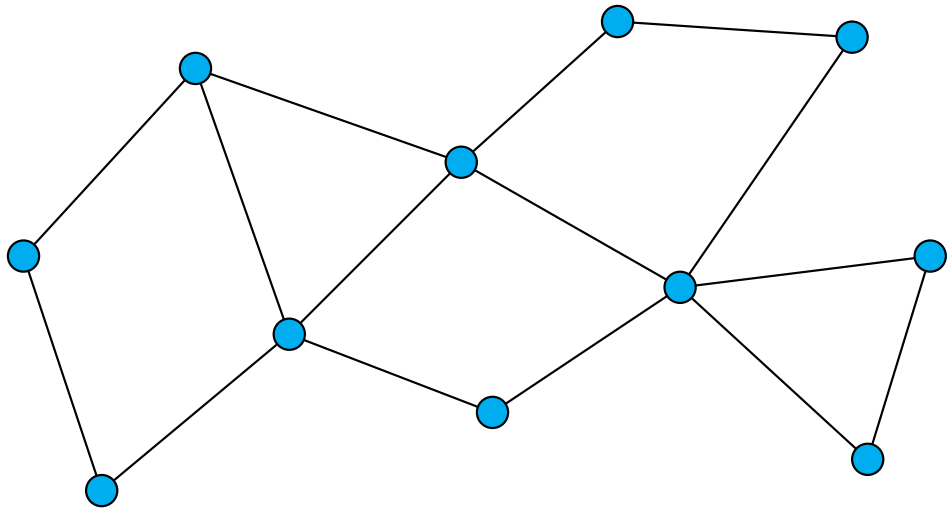
Hamiltonovská kružnice

HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



Odpověď: NE

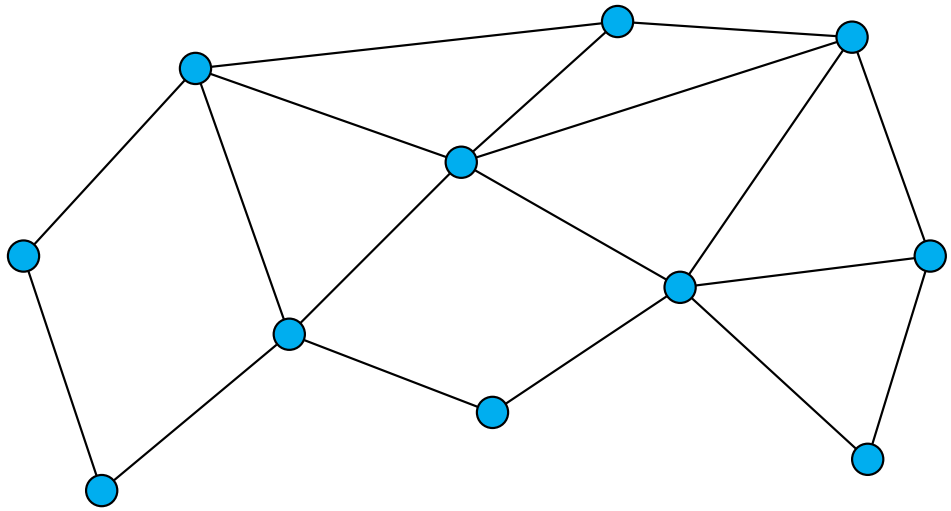
Hamiltonovská kružnice

HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



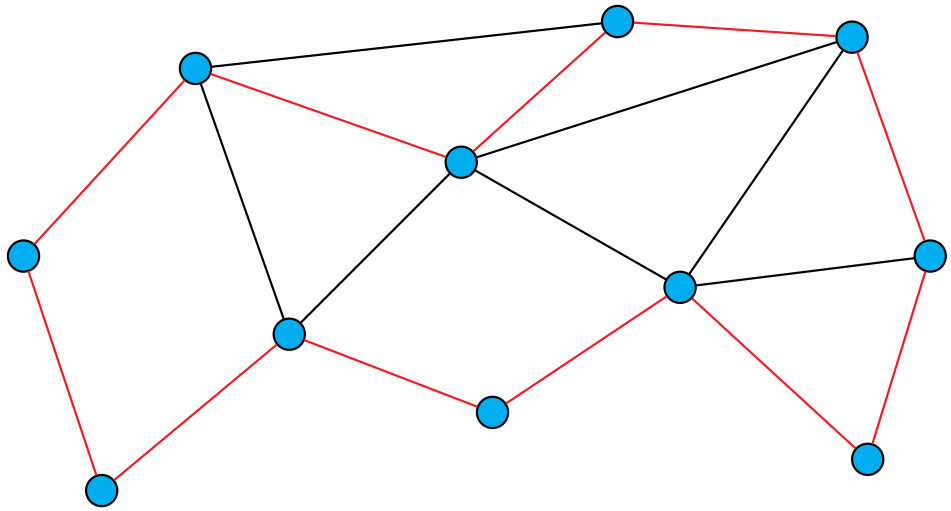
Hamiltonovská kružnice

HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf G.

Otázka: Existuje v grafu G Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



Odpověď: Ano

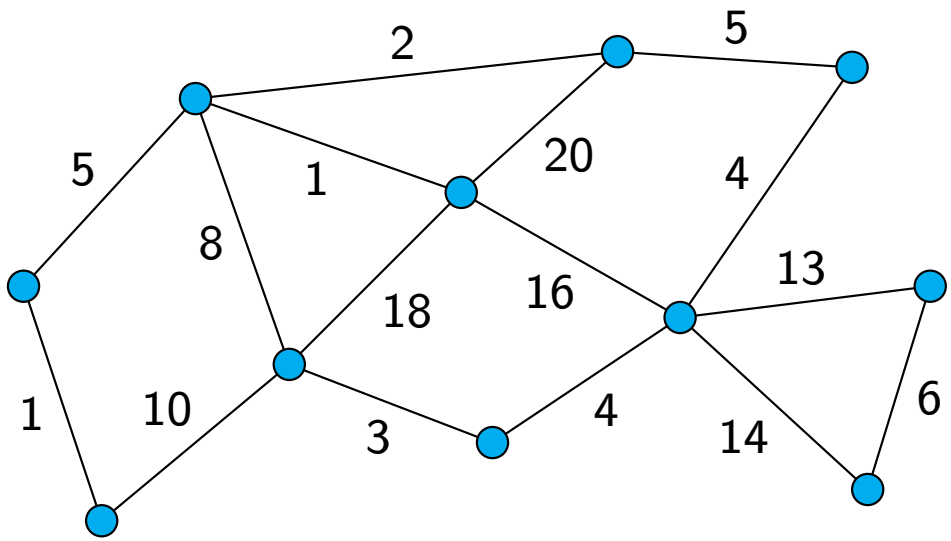
Problém obchodního cestujícího

TSP - Problém „obchodního cestujícího“

Vstup: Neorientovaný graf G s hranami ohodnocenými přirozenými čísly a číslo k.

Otázka: Existuje v grafu G uzavřená cesta procházející všemi vrcholy takový, že součet délek hran na této cestě (včetně opakovaných) je maximálně k?

Příklad: $k = 70$



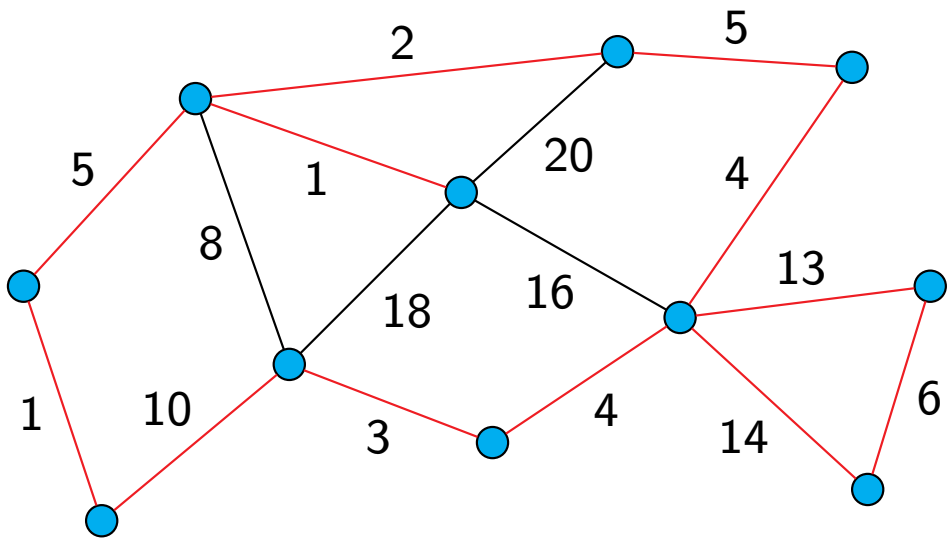
Problém obchodního cestujícího

TSP - Problém „obchodního cestujícího“

Vstup: Neorientovaný graf G s hranami ohodnocenými přirozenými čísly a číslo k.

Otázka: Existuje v grafu G uzavřená cesta procházející všemi vrcholy takový, že součet délek hran na této cestě (včetně opakovaných) je maximálně k?

Příklad: $k = 70$



Odpověď: Ano, protože byl nalezen sled se součtem 69.

SUBSET-SUM

Problém SUBSET-SUM

Vstup: Sekvence přirozených čísel $a_1, a_2, \ldots, a_n$ a přirozené číslo s.

Otázka: Existuje množina $I \subseteq \{1, 2, \ldots, n\}$ taková, že $\sum_{i \in I} a_i = s$?

Jinak řečeno, ptáme se zda z dané (multi)množiny čísel je možné vybrat podmnožinu, jejíž součet je s.

Příklad: Pro vstup tvořený čísly $3, 5, 2, 3, 7$ a číslem $s = 15$ je odpověď Ano, neboť $3 + 5 + 7 = 15$.

Pro vstup tvořený čísly $3, 5, 2, 3, 7$ a číslem $s = 16$ je odpověď Ne, neboť žádná podmnožina těchto čísel nedává součet 16.

SUBSET-SUM

Poznámka:
Pořadí čísel $a_1, a_2, \ldots, a_n$ na vstupu není důležité.

Všimněte si však určitého rozdílu oproti tomu, kdybychom problém formulovali tak, že vstupem je množina $\{a_1, a_2, \ldots, a_n\}$ a číslo s — v množině se čísla neopakují, zatímco v sekvenci se může totéž číslo vyskytnout vícekrát.

SUBSET-SUM

Problém SUBSET-SUM je speciálním případem **problému batohu** (knapsack problem):

Knapsack problem

Vstup: Sekvence dvojic přirozených čísel $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ a dvě přirozená čísla s a t.

Otázka: Existuje množina $I \subseteq \{1, 2, \ldots, n\}$ taková, že $\sum_{i \in I} a_i \leq s$ a $\sum_{i \in I} b_i \geq t$?

SUBSET-SUM

Neformálně můžeme problém batohu formulovat takto:

Máme n předmětů, kde i-tý předmět váží a_i gramů a má cenu b_i Kč. Do batohu se vejdou předměty o maximální celkové váze s gramů.

Otázka zní, zda můžeme z předmětů vybrat podmnožinu, která by vážila maximálně s gramů a měla celkovou cenu alespoň t Kč.

Poznámka:
Zde jsme problém batohu formulovali jako rozhodovací problém.

Běžnější je formulovat tento problém jako optimalizační problém, kde je cílem najít takovou množinu $I \subseteq \{1, 2, \ldots, n\}$, kde hodnota $\sum_{i \in I} b_i$ je maximální, přičemž ovšem musí být dodržena podmínka $\sum_{i \in I} a_i \leq s$, tj. vybrat předměty s maximální celkovou cenou tak, aby nebyla překročena kapacita batohu.

SUBSET-SUM

To, že SUBSET-SUM je speciálním případem problému batohu, vidíme z následující jednoduché konstrukce:

Řekněme, že $a_1, a_2, \ldots, a_n$, s_1 je instance problému SUBSET-SUM. Je očividné, že pro instanci problému batohu, kde máme sekvenci $(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)$, $s = s_1$ a $t = s_1$, je odpověď stejná jako pro původní instanci SUBSET-SUM.

SUBSET-SUM

Pokud chceme studovat složitost problémů jako jsou SUBSET-SUM nebo problém batohu, je dobré si nejprve ujasnit, co považujeme za velikost vstupu.

Asi nejpřirozenější je definovat velikost vstupu jako celkový počet bitů, který potřebujeme k zápisu instance.

Musíme však určit, jakým způsobem jsou na vstupu zadána přirozená čísla – zda binárně (případně v jiné číselné soustavě o základu alespoň 2, např. desítkové nebo šestnáctkové) nebo unárně.

- Pokud počítáme velikost vstupu jako celkový počet bitů při použití **binárního** zápisu čísel, tak pro problém SUBSET-SUM není znám polynomiální algoritmus.
- Pokud počítáme velikost vstupu jako celkový počet bitů při použití **unárního** zápisu, tak existuje pro problém SUBSET-SUM algoritmus s polynomiální časovou složitostí.

ILP – celočíselné lineární programování

Problém ILP (celočíselné lineární programování)

Vstup: Celočíselná matice A a celočíselný vektor b.

Otázka: Existuje celočíselný vektor x, takový že $Ax \leq b$?

Příklad instance problému:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

Ptáme se tedy, zda existuje celočíselné řešení následující soustavy nerovnic:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

ILP – celočíselné lineární programování

Jedním z řešení soustavy

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

je například $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, tj.

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

neboť

$$\begin{aligned} 3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\ -4 + 1 &= -3 \leq -3 \\ 2 \cdot (-4) + 1 &= -7 \leq 5 \end{aligned}$$

Pro tuto instanci je tedy odpověď ANO.

ILP – celočíselné lineární programování

Poznámka: Analogický problém, kdy se pro danou soustavu lineálních nerovnic ptáme, zda existuje její řešení v oboru **reálných** čísel, je možné řešit v polynomiálním čase.

Příklady algoritmických problémů

Na následujích slidech jsou uvedeny příklady několika typických algoritmických problémů, které:

- **nejsou algoritmicky řešitelné**
- pro tyto problémy se dá dokázat, že pro pro ně nemohou existovat algoritmy, které by je řešily

Poznámka: Připomeňme, že algoritmus řeší daný problém, jestliže se pro každý vstup po konečném počtu kroků zastaví a vydá správný výstup.

Pro následující problémy tedy platí, že pro každý algoritmus, který by se je pokoušel řešit, se dá najít příklad vstupu, pro který:

- se algoritmus nikdy nezastaví, nebo
- vydá chybný výstup

Poznámka: V případě, že se jedná o rozhodovací problémy, se problémům, které nejsou algoritmicky řešitelné, říká **nerozhodnutelné problémy**.

Příklady nerozhodnutelných problémů

S jedním příkladem nerozhodnutelného problému už jsme se setkali:

Problém

Vstup: Bezkontextové gramatiky G_1 a G_2.

Otázka: Je $L(G_1) = L(G_2)$?

případně

Problém

Vstup: Bezkontextová gramatika G generující jazyk nad abecedou Σ.

Otázka: Je $L(G) = \Sigma^*$?

Další nerozhodnutelné problémy

Problém

Vstup: Bezkontextové gramatiky G_1 a G_2.

Otázka: Je $L(G_1) \cap L(G_2) = \emptyset$?

Problém

Vstup: Bezkontextová gramatika G.

Otázka: Je G nejednoznačná?

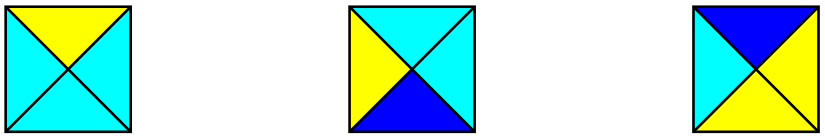
Další nerozhodnutelné problémy

Nerozhodnutelná je celá řada 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?
- ...

Další nerozhodnutelné problémy

Vstupem je množina typů kachliček, jako třeba:

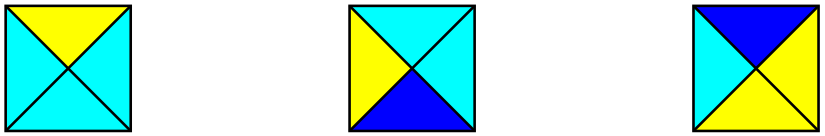


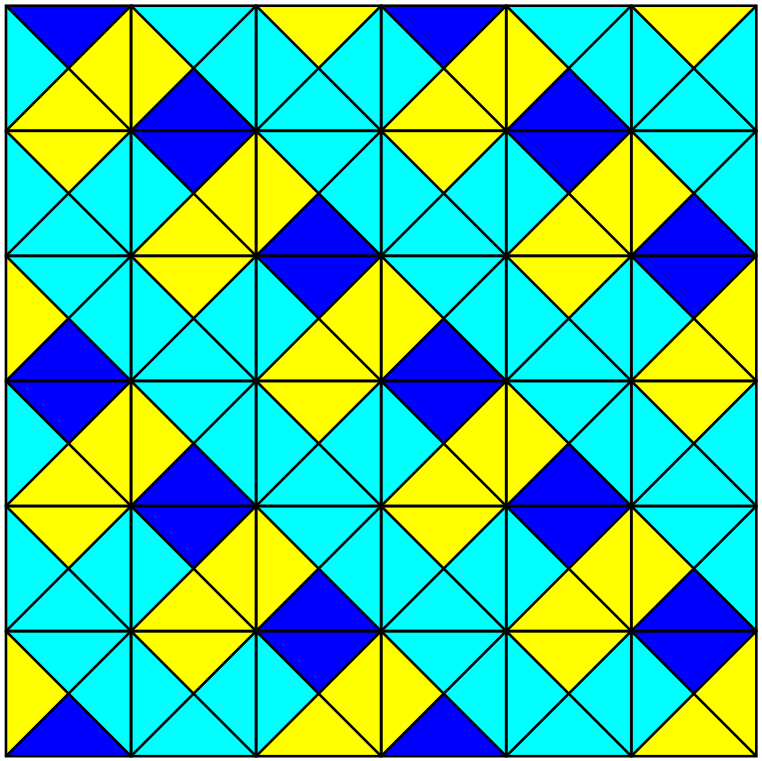
Otázka je, zda je možné použitím daných typů kachliček pokrýt každou libovolně velkou konečnou plochu tak, aby všechny kachličky spolu sousedily stejnými barvami.

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

Kachličky není dovoleno otáčet.

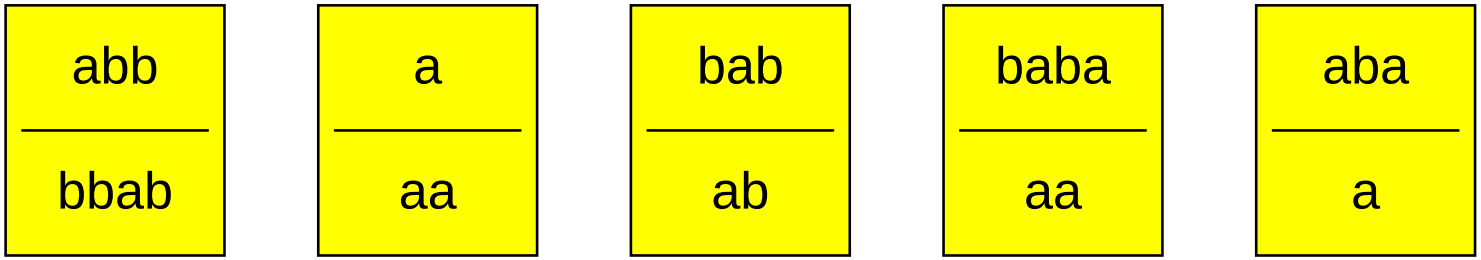
Další nerozhodnutelné problémy



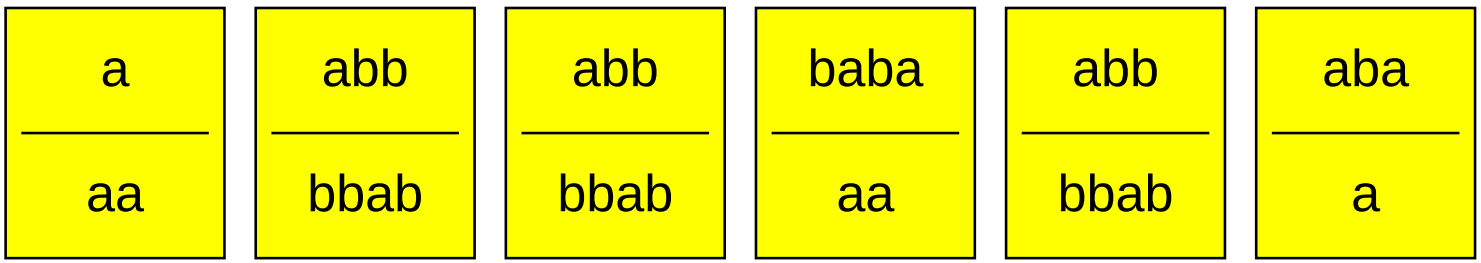


Další nerozhodnutelné problémy

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.

Další nerozhodnutelné problémy

Problém

Vstup: Uzavřená formule predikátové logiky, ve které mohou být použity jako funkční symboly $+$ a $*$ a celočíselné konstanty a jako predikátové symboly $=$ a $<$.

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

Příklad vstupu:

$$\forall x \exists y \forall z ((x * y = z) \wedge (y + 5 = x))$$

Poznámka: Úzce souvisí s Gödelovou větou o neúplnosti.

Další nerozhodnutelné problémy

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, že v nich nesmí být použit funkční symbol $*$ (násobení), tak je problém algoritmicky rozhodnutelný.

Další nerozhodnutelné problémy

Pokud můžeme používat $*$, 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

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

platí v oboru přirozených čísel.

Další nerozhodnutelné problémy

Také následující problém je algoritmicky nerozhodnutelný:

Problém

Vstup: Uzavřená formule φ predikátové logiky.

Otázka: Platí $\models \varphi$?

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