

Úvod do teoretické informatiky

Zdeněk Sawa

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

2. února 2017

Jméno: doc. Ing. Zdeněk Sawa, Ph.D.

E-mail: `zdenek.sawa@vsb.cz`

Místnost: EA413

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

- **Zápočet** (22 bodů):

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

Minimum pro získání zápočtu je 12 bodů.

Možnost opravy za 18 bodů.

- **Zkouška** (78 bodů)

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

Teoretická informatika — vědní obor na pomezí mezi informatikou a matematikou

- zkoumání obecných otázek týkajících se algoritmů a výpočtů
- zkoumání různých formalismů pro popis algoritmů
- zkoumání různých prostředků pro popis syntaxe a sémantiky formálních jazyků (zejména s důrazem na programovací jazyky)
- matematický přístup k analýze a řešení problémů (dokazování obecně platných matematických tvrzení týkajících se algoritmů)

Příklady některých typických otázek studovaných v teoretické informatice:

- Je možné daný problém řešit pomocí nějakého algoritmu?
- Pokud je možné daný problém řešit pomocí algoritmu, jaká je výpočetní složitost tohoto algoritmu?
- Existuje pro daný problém nějaký efektivní algoritmus, který ho řeší?
- Jak se přesvědčit o tom, že daný algoritmus je skutečně korektním řešením daného problému?
- Jaké instrukce musí umět vykonat stroj, který by mohl provádět daný algoritmus?

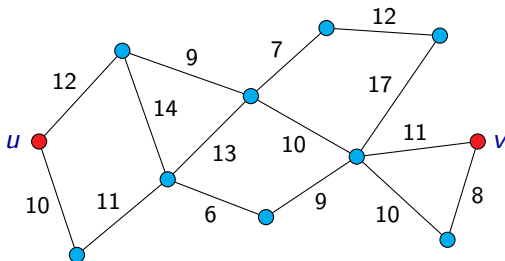
Příklad algoritmického problému

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:



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

- teorie grafů
- teorie čísel
- výpočetní geometrie
- vyhledávání v textu
- teorie her
- ...

Logika — obor zabývající se otázkami správného vyvozování a argumentace

- zkoumání, kdy závěr vyplývá z daných předpokladů
- otázky týkající se důkazů a dokazatelnosti
- poskytuje základní jazyk matematiky a všech na matematice založených věd
- souvisí se zkoumáním základů matematiky
- využívá se v informatice na mnoha různých úrovních

Výroková logika

- *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.*

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 .

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.“

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.“*

Z výroků je možné vytvářet složitější výroky pomocí **logických spojek**:

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 — $p, q, r, \dots$
(případně s indexy — $p_0, p_1, p_2, \dots$)

Výroky jsou reprezentovány pomocí **formulí** — formule mají přesně danou syntaxi a sémantiku.

„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

1	0
pravda	nepravda
ano	ne
true	false

Pro označení pravdivostních hodnot budeme používat 0 a 1.

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

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: $\neg\varphi$

φ	$\neg\varphi$
0	1
1	0

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 — „ $2 + 2 = 4$ “

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.*“

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 — „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“

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

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

$$\neg p \vee q.$$

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 $1 + 1 = 3$.“

Příklady různých způsobů vyjádření tvrzení $p \rightarrow q$ 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 “

Pokud platí $\varphi \rightarrow \psi$ 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 — „ 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“

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

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

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.“

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

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 tato formule není:

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

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.

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 .

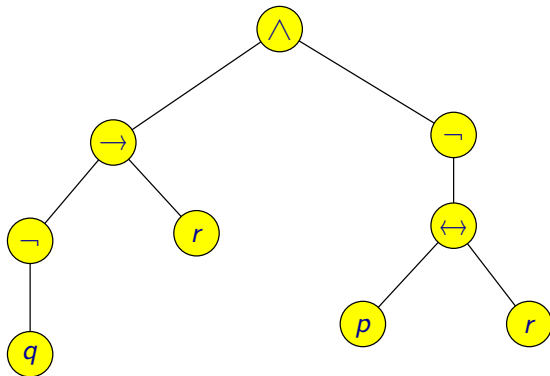
Příklad: Podformule formule $((\neg(p \wedge q)) \leftrightarrow r)$:

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

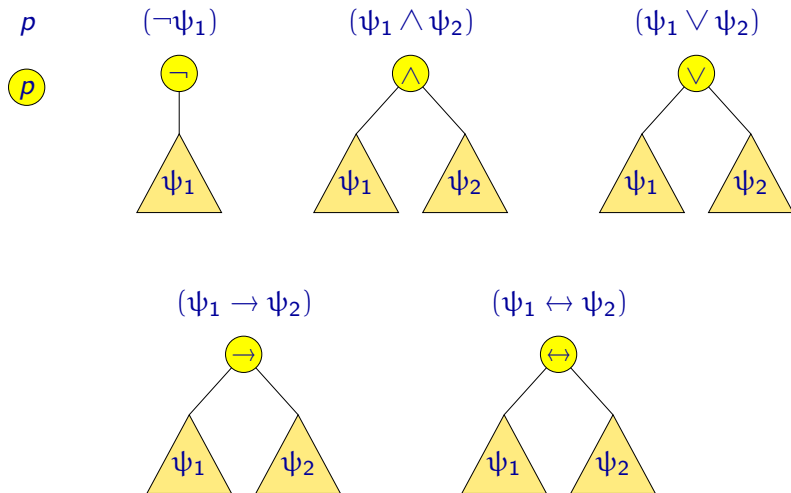
Alternativní symboly pro logické spojky:

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

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



Syntaxe formulí výrokové logiky



Arita logických spojek:

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

Konvence pro vypouštění závorek:

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

$$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow$$

- 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.

At — množina atomických výroků

Například

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

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 $v : At \rightarrow \{0, 1\}$.)

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

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

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

Příklad: $At = \{p, q, r\}$

v_0 :	$v_0(p) = 0$,	$v_0(q) = 0$,	$v_0(r) = 0$
v_1 :	$v_1(p) = 0$,	$v_1(q) = 0$,	$v_1(r) = 1$
v_2 :	$v_2(p) = 0$,	$v_2(q) = 1$,	$v_2(r) = 0$
v_3 :	$v_3(p) = 0$,	$v_3(q) = 1$,	$v_3(r) = 1$
v_4 :	$v_4(p) = 1$,	$v_4(q) = 0$,	$v_4(r) = 0$
v_5 :	$v_5(p) = 1$,	$v_5(q) = 0$,	$v_5(r) = 1$
v_6 :	$v_6(p) = 1$,	$v_6(q) = 1$,	$v_6(r) = 0$
v_7 :	$v_7(p) = 1$,	$v_7(q) = 1$,	$v_7(r) = 1$

Formule φ má při pravdivostním ohodnocení v pravdivostní hodnotu 1:

$$v \models \varphi$$

Formule φ má při pravdivostním ohodnocení v pravdivostní hodnotu 0:

$$v \not\models \varphi$$

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 $v \not\models \psi$.

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

- $v \models p$
- $v \not\models q$
- $v \models r$

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

- $v \models p$
- $v \not\models q$
- $v \models r$
- $v \models \neg q$

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

- $v \models p$
- $v \not\models q$
- $v \models r$
- $v \models \neg q$
- $v \models \neg q \rightarrow r$

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

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

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

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

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

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

Příklad: $At = \{p, q, r\}$

pravdivostní ohodnocení v , kde $v(p) = 1$, $v(q) = 0$ a $v(r) = 1$

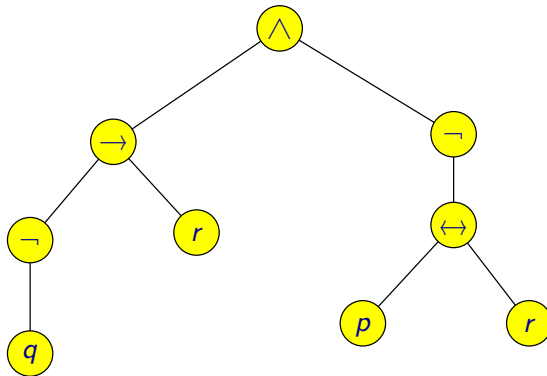
- $v \models p$
- $v \not\models q$
- $v \models r$
- $v \models \neg q$
- $v \models \neg q \rightarrow r$
- $v \models p \leftrightarrow r$
- $v \not\models \neg(p \leftrightarrow r)$
- $v \not\models (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$

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

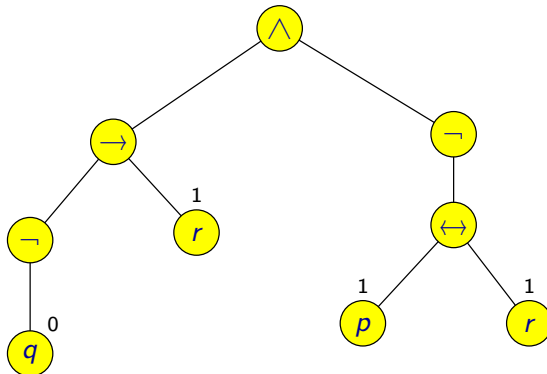
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

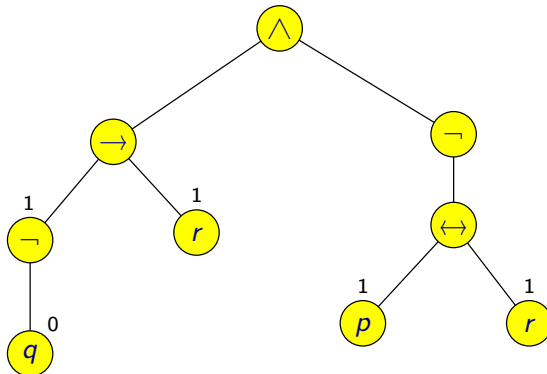
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

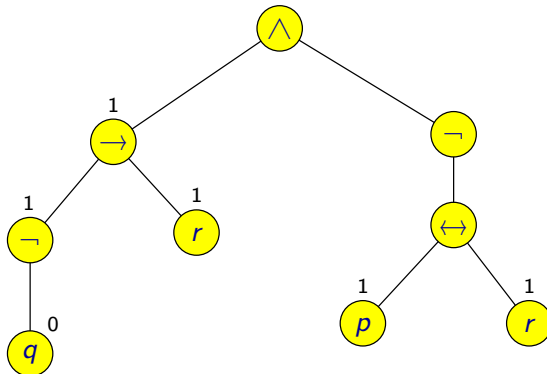
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

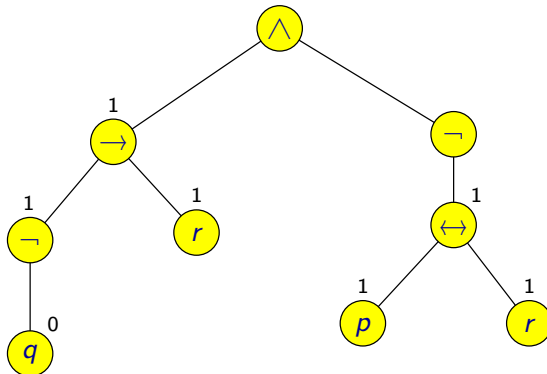
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

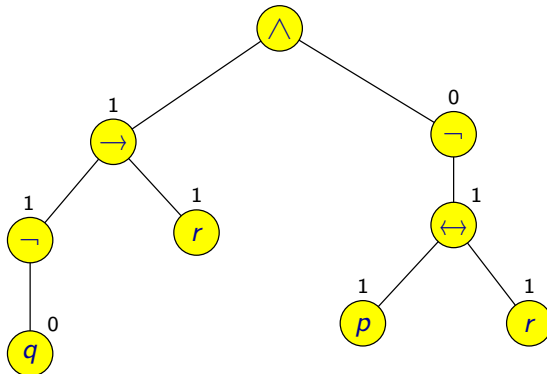
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

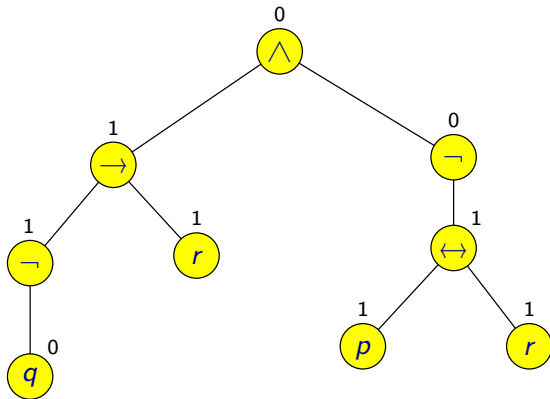
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

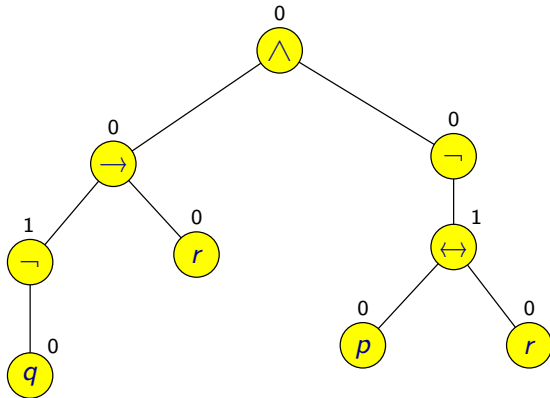
$$\varphi := (\neg q \rightarrow r) \wedge \neg(p \leftrightarrow r)$$



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

Sémantika výrokové logiky

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



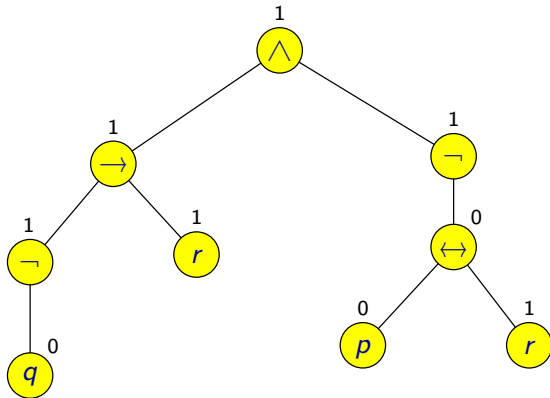
$$v_0(p) = 0$$

$$v_0(q) = 0$$

$$v_0(r) = 0$$

Sémantika výrokové logiky

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



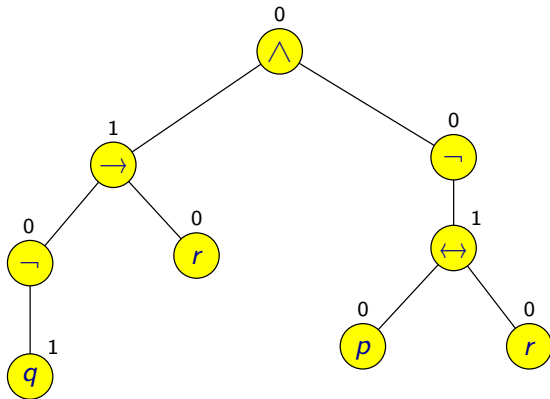
$$v_1(p) = 0$$

$$v_1(q) = 0$$

$$v_1(r) = 1$$

Sémantika výrokové logiky

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



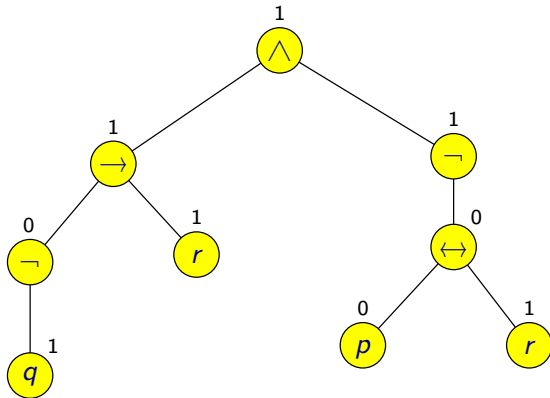
$$v_2(p) = 0$$

$$v_2(q) = 1$$

$$v_2(r) = 0$$

Sémantika výrokové logiky

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



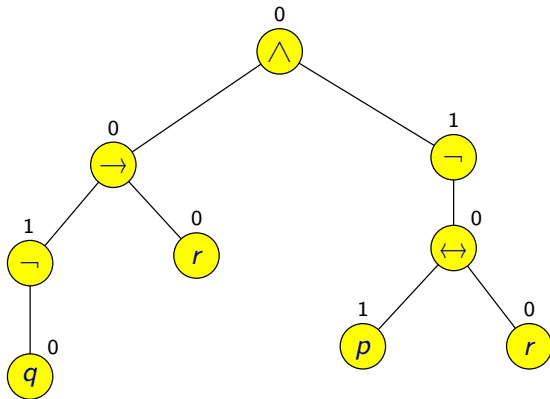
$$v_3(p) = 0$$

$$v_3(q) = 1$$

$$v_3(r) = 1$$

Sémantika výrokové logiky

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



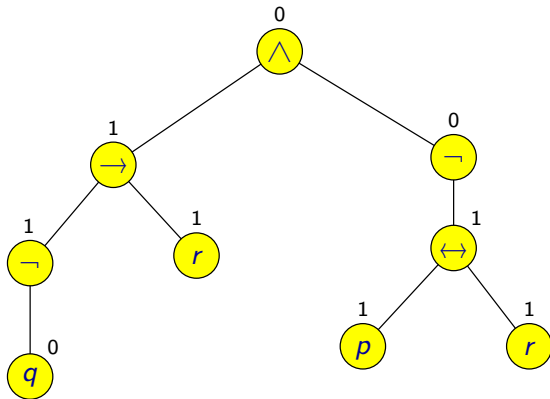
$$v_4(p) = 1$$

$$v_4(q) = 0$$

$$v_4(r) = 0$$

Sémantika výrokové logiky

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

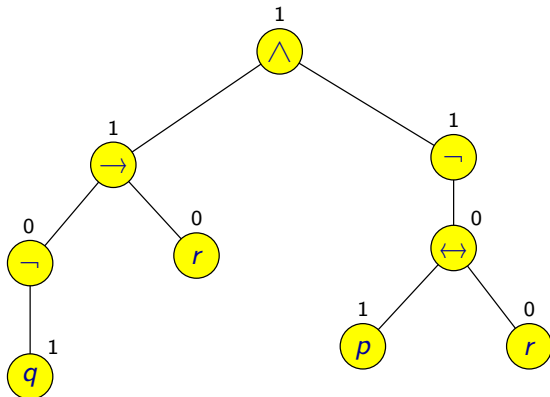


$$v_5(p) = 1$$

$$v_5(q) = 0$$

$$v_5(r) = 1$$

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



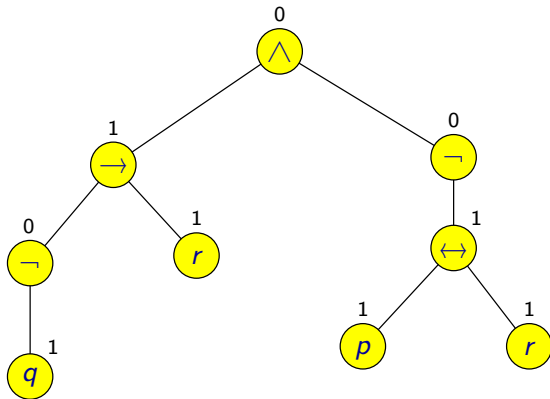
$$v_6(p) = 1$$

$$v_6(q) = 1$$

$$v_6(r) = 0$$

Sémantika výrokové logiky

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



$$v_7(p) = 1$$

$$v_7(q) = 1$$

$$v_7(r) = 1$$

Sémantika výrokové logiky

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

p	q	r	$\neg q$	$\neg q \rightarrow r$	$p \leftrightarrow r$	$\neg(p \leftrightarrow r)$	φ
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 daná formule pravdivá, jsou jejími **modely**:

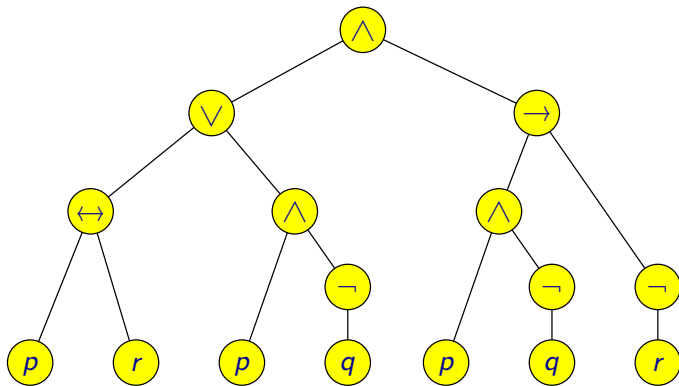
$$v_1: \quad v_1(p) = 0, \quad v_1(q) = 0, \quad v_1(r) = 1,$$

$$v_3: \quad v_3(p) = 0, \quad v_3(q) = 1, \quad v_3(r) = 1,$$

$$v_6: \quad v_6(p) = 1, \quad v_6(q) = 1, \quad v_6(r) = 0,$$

Formule reprezentované jako orientované acyklické grafy

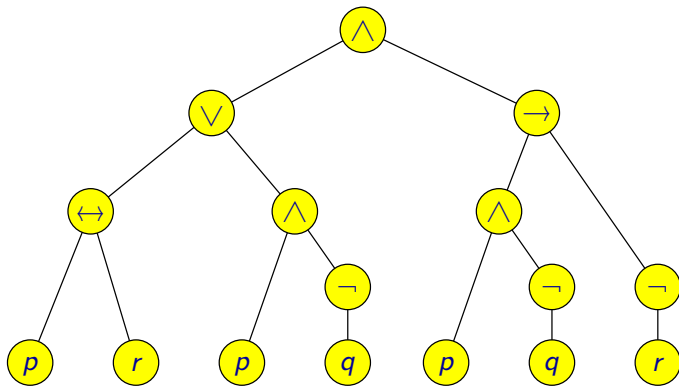
$$\varphi := ((p \leftrightarrow r) \vee (p \wedge \neg q)) \wedge ((p \wedge \neg q) \rightarrow \neg r)$$



V abstraktním syntaktickém stromě odpovídají vrcholy všem **výskytům** jednotlivých podformulí.

Formule reprezentované jako orientované acyklické grafy

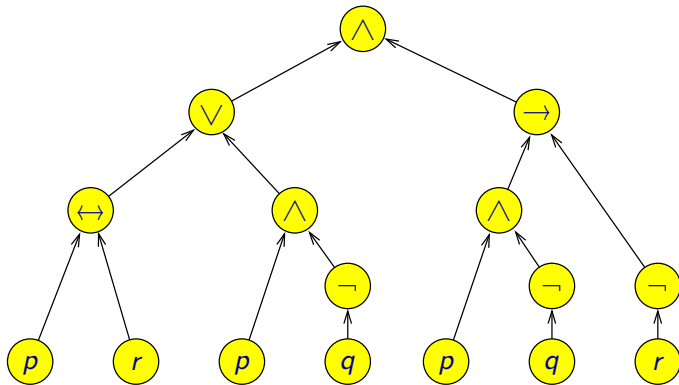
$$\varphi := ((p \leftrightarrow r) \vee (p \wedge \neg q)) \wedge ((p \wedge \neg q) \rightarrow \neg r)$$



Alternativně můžeme formuli znázornit jako **orientovaný acyklický graf**.

Formule reprezentované jako orientované acyklické grafy

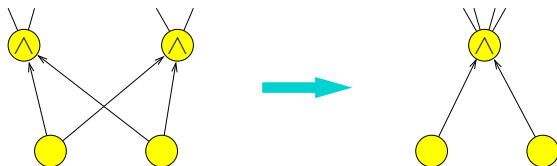
$$\varphi := ((p \leftrightarrow r) \vee (p \wedge \neg q)) \wedge ((p \wedge \neg q) \rightarrow \neg r)$$



Nejprve hranám přiřadíme orientaci od potomků k rodičům.

Formule reprezentované jako orientované acyklické grafy

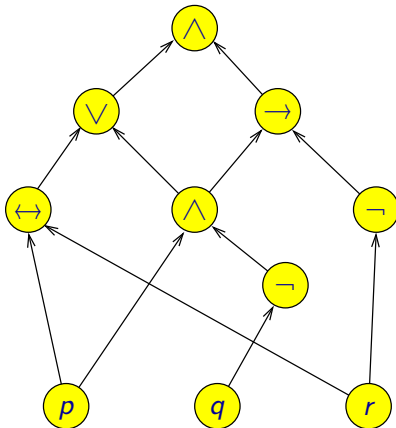
- Ztotožníme všechny listy označené stejným atomickým výrokem.
- Následně je možné ztotožnit vrcholy, které jsou označeny stejným symbolem a mají stejné předchůdce.



- Taková ztotožnění je možné provádět opakovaně — když jsou některé vrcholy ztotožněny, může být možné ztotožnit některé další vrcholy.

Formule reprezentované jako orientované acyklické grafy

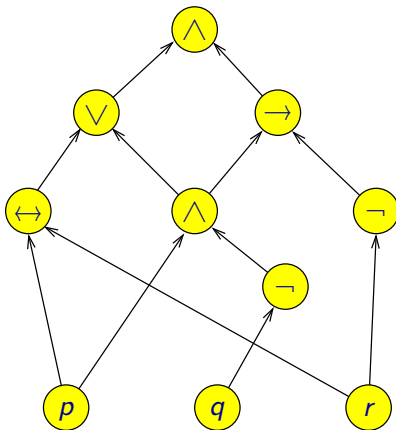
$$\varphi := ((p \leftrightarrow r) \vee (p \wedge \neg q)) \wedge ((p \wedge \neg q) \rightarrow \neg r)$$



- Když tímto způsobem ztotožníme všechny vrcholy, které je takto možno ztotožnit, dostaneme graf, kde jednotlivé vrcholy odpovídají jednotlivým různým podformulím dané formule.
- Na acyklický orientovaný graf reprezentující danou formuli je možné se dívat jako na **logický obvod**:
 - **Vstupy** — vrcholy označené atomickými výroky
 - **Výstup** — vrchol odpovídající celé formuli

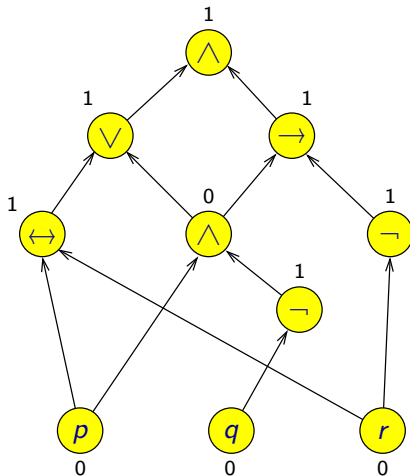
Formule reprezentované jako orientované acyklické grafy

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



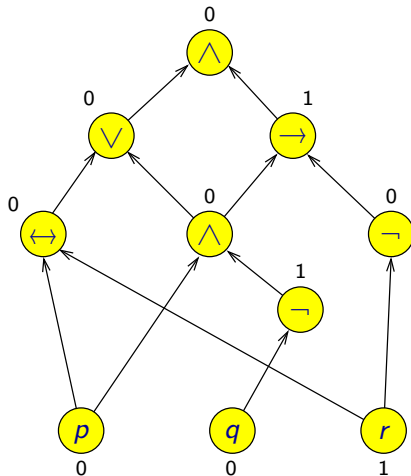
Formule reprezentované jako orientované acyklické grafy

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



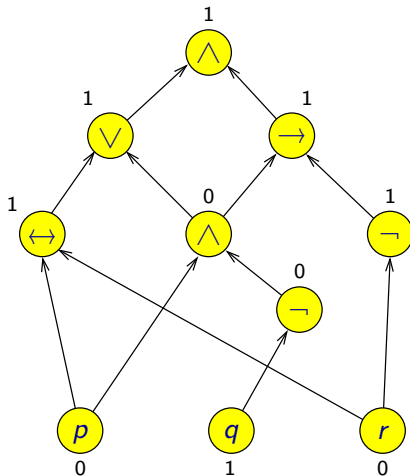
Formule reprezentované jako orientované acyklické grafy

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



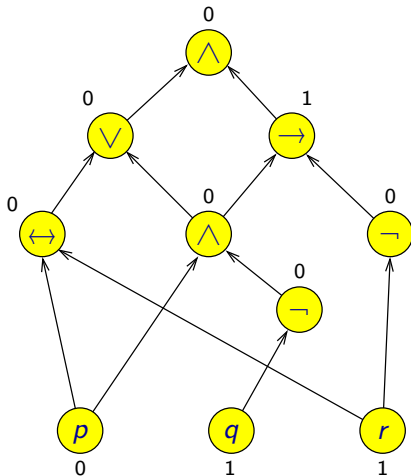
Formule reprezentované jako orientované acyklické grafy

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



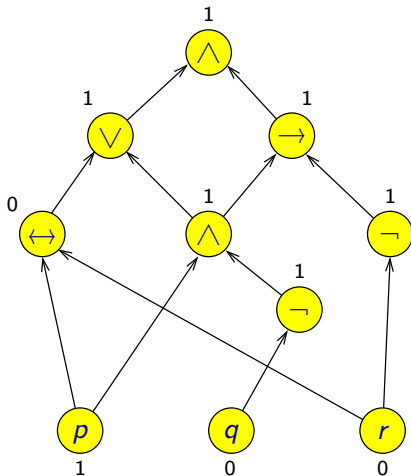
Formule reprezentované jako orientované acyklické grafy

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



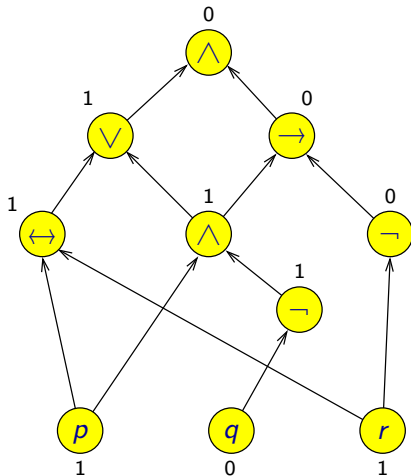
Formule reprezentované jako orientované acyklické grafy

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



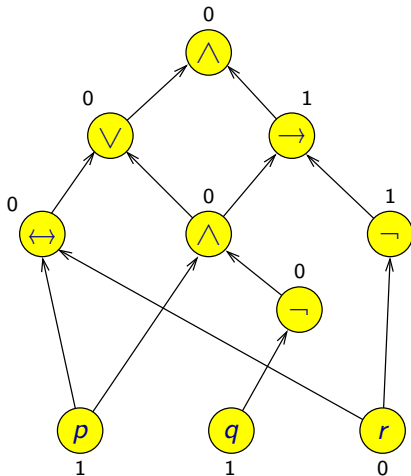
Formule reprezentované jako orientované acyklické grafy

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



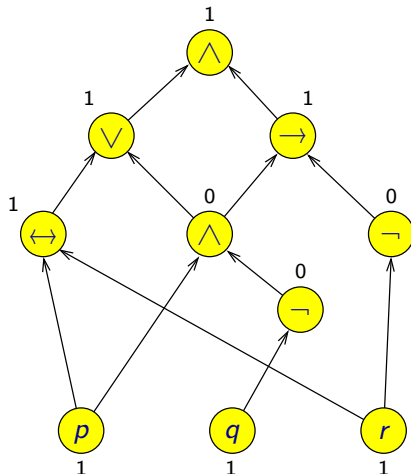
Formule reprezentované jako orientované acyklické grafy

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



Formule reprezentované jako orientované acyklické grafy

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



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.“

$$q \vee \neg q$$

Příklad komplikovanější tautologie:

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

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

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

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

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

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

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

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

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

- 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 $\neg p \rightarrow q$.

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

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

Pro libovolné formule ψ a χ proto platí, že

$$\psi \rightarrow (\psi \vee \chi)$$

je tautologie.

Náhrada atomických výroků:

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

Dostaneme tautologii

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

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

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.

- 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:

- 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.

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

- Systematicky probrat všechna možná pravdivostní ohodnocení.

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

- Můžeme si nakreslit graf reprezentující danou formuli a zkusit přiřadit vrcholům hodnoty 0 a 1 tak, abychom buď našli příklad ohodnocení, které nás zajímá (např. nějaké ohodnocení, při kterém je daná formule nepravdivá), nebo zjistili, že takové ohodnocení neexistuje.

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

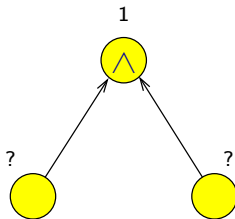
- Je třeba zjistit, zda existuje nějaké ohodnocení, při kterém je tato formule nepravdivá.
- Při takovém ohodnocení by vrchol, který odpovídá celé formuli, měl přiřazenu hodnotu 0.
- Tomuto vrcholu tedy zkusíme přiřadit hodnotu 0.
- Zkousíme postupně doplňovat hodnoty k dalším vrcholům tak, aby byly konzistentní s dříve přiřazenými hodnotami.
- Pokud se podaří celý graf konzistentně ohodnotit, máme ohodnocení, při kterém formule není pravdivá.

V tomto případě je pak jasné, že daná formule není tautologie.

Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

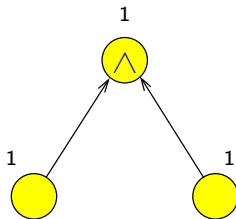
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

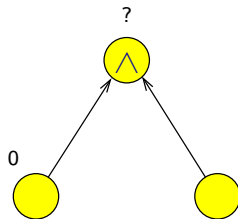
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

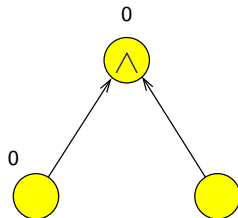
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

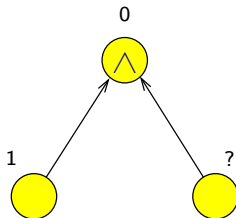
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

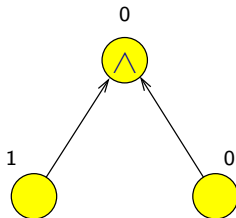
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

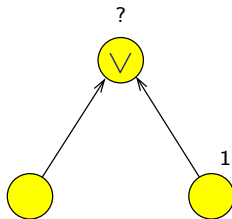
φ	ψ	$\varphi \wedge \psi$
0	0	0
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

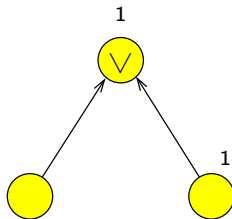
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

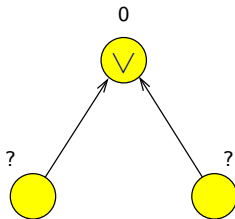
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

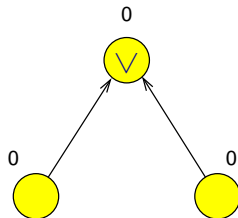
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

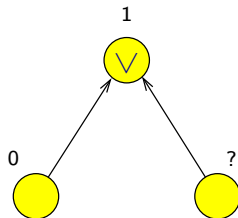
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

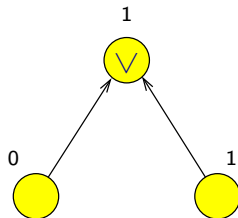
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

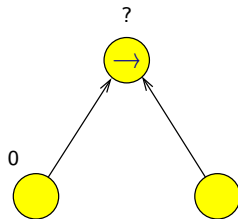
φ	ψ	$\varphi \vee \psi$
0	0	0
0	1	1
1	0	1
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

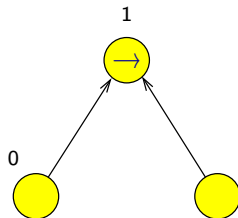
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

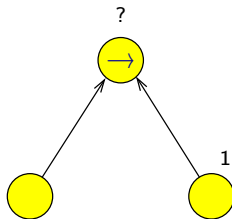
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

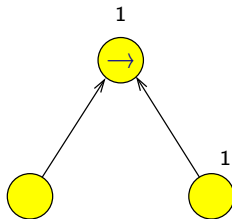
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

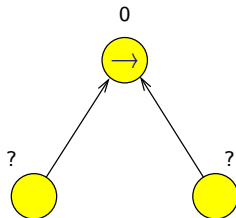
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

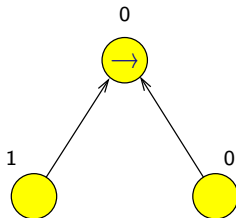
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

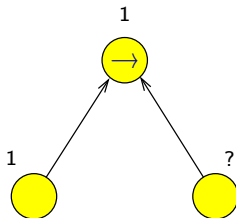
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

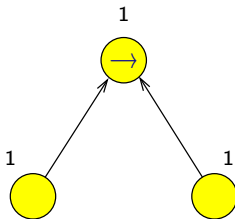
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

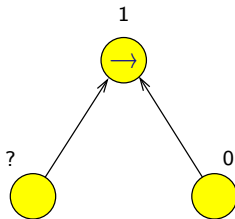
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

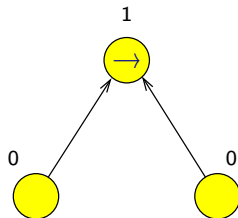
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

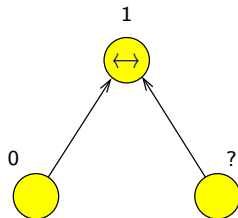
φ	ψ	$\varphi \rightarrow \psi$
0	0	1
0	1	1
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

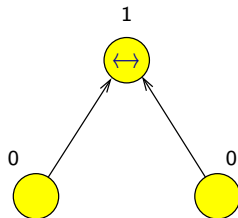
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

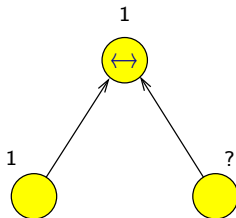
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

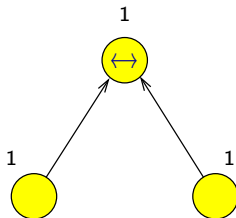
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

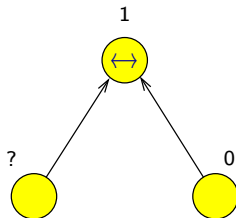
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

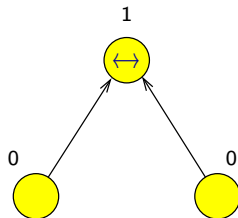
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

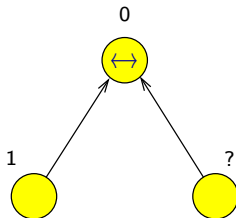
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

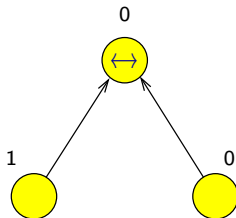
φ	ψ	$\varphi \leftrightarrow \psi$
0	0	1
0	1	0
1	0	0
1	1	1



Pravdivostní ohodnocení

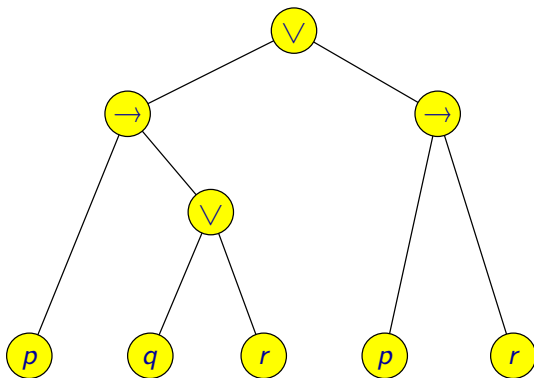
- Pokud už jsou některým vrcholům pravdivostní hodnoty přiřazeny, může toto přiřazení klást omezení na to, jaké hodnoty je možno přiřadit dalším vrcholům.
- Příklady toho, kdy dříve přiřazené hodnoty vynucují určitou hodnotu u dalšího vrcholu (nebo i více vrcholů):

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



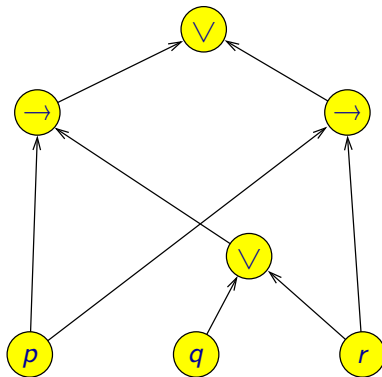
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



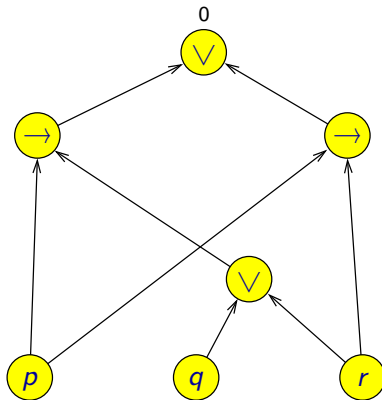
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



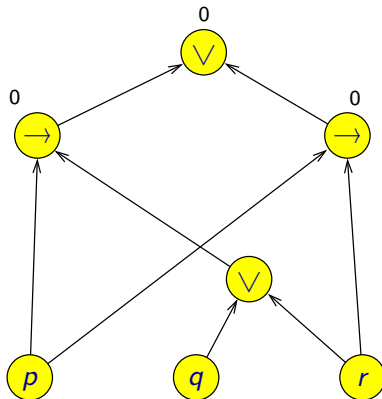
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



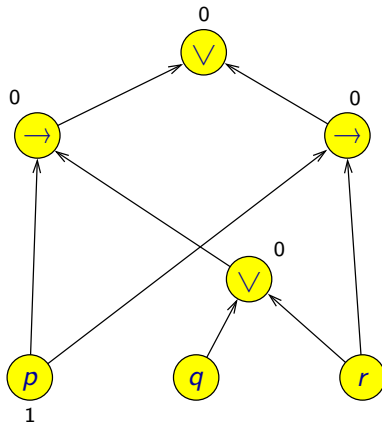
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



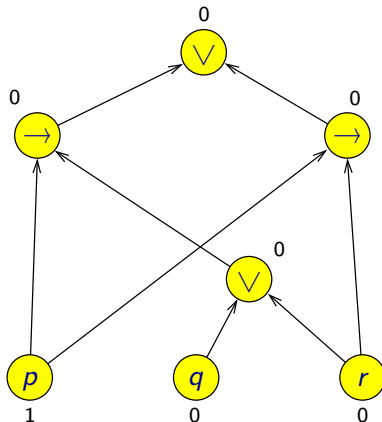
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



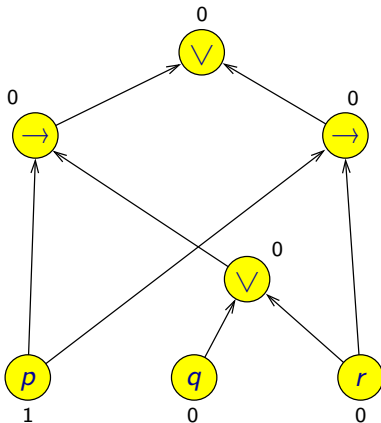
Je daná formule tautologií?

Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



Je daná formule tautologií?

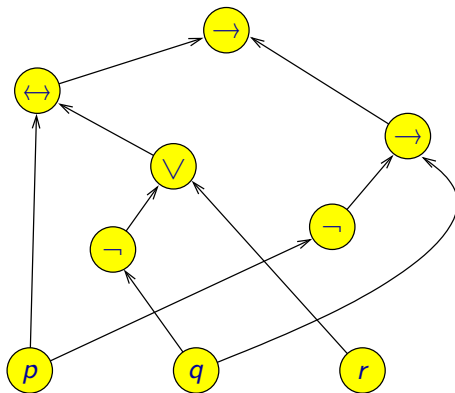
Příklad: $\varphi_1 := (p \rightarrow (q \vee r)) \vee (p \rightarrow r)$



Formule φ_1 není tautologie — při ohodnocení v , kde $v(p) = 1$, $v(q) = 0$, $v(r) = 0$, je nepravdivá.

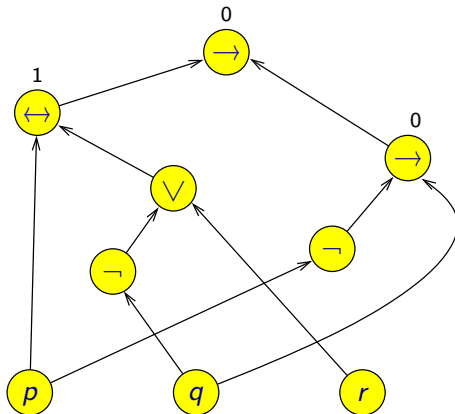
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



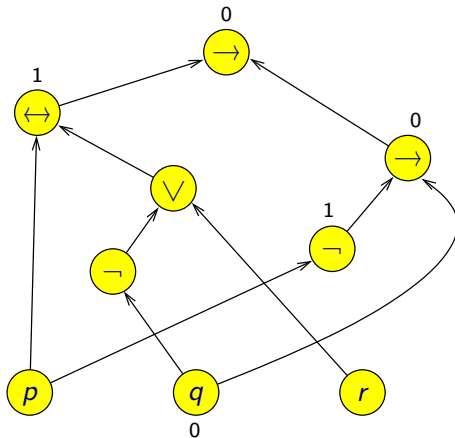
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



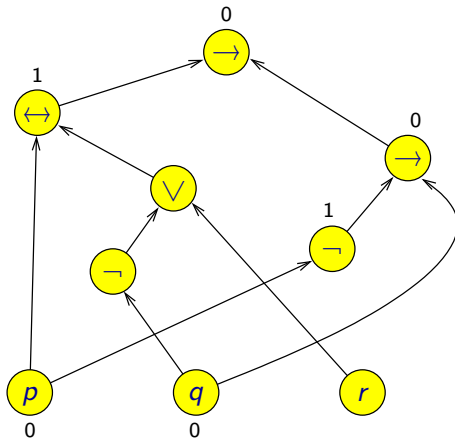
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



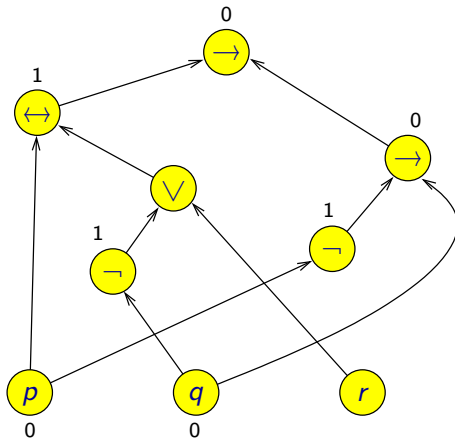
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



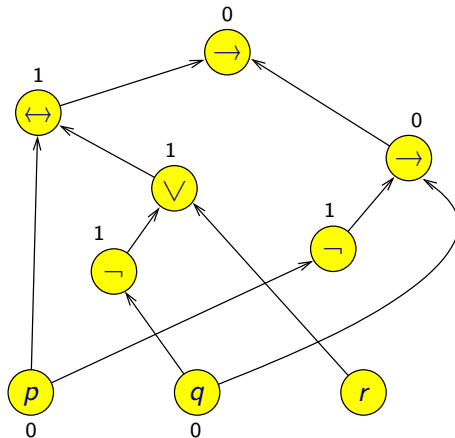
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



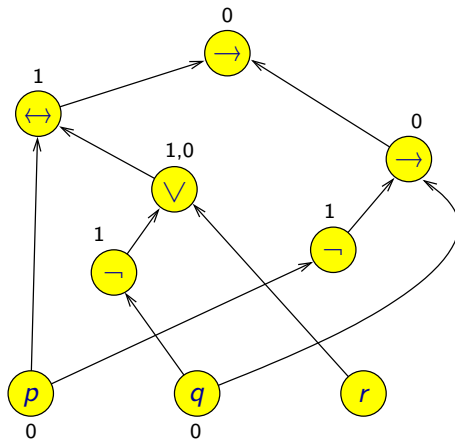
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



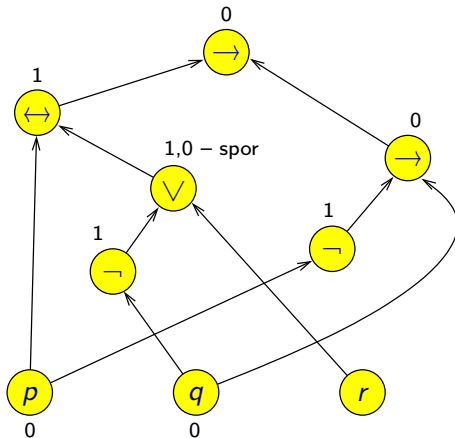
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



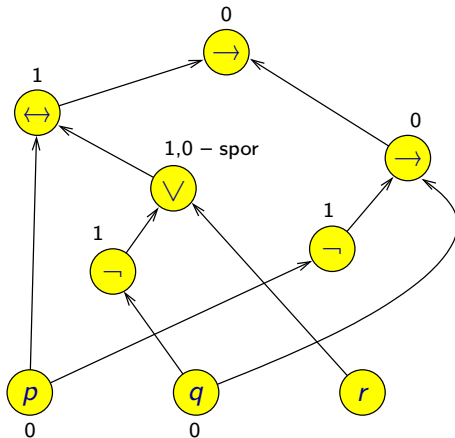
Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



Je daná formule tautologií?

Příklad: $\varphi_2 := (p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$



Formule φ_2 je tautologie.

Sémantický spor — případ, kdy zjistíme, že při ohodnocení s požadovanou vlastností, které hledáme (např. ohodnocení, kde je daná formule nepravdivá), by musela být nějaká formule současně pravdivá i nepravdivá.

- Žádné takové pravdivostní ohodnocení, kde by byla nějaká formule současně pravdivá i nepravdivá, nemůže existovat.
- Tímto způsobem můžeme tedy například zdůvodnit, že daná formule je tautologií (a tedy vždy pravdivá), protože nalezením sémantického sporu ukážeme, že nemůže existovat žádné ohodnocení, při kterém by byla nepravdivá.

Je daná formule tautologií?

Postup z předchozího příkladu je také možné popsat následující posloupností argumentů:

1. Předpokládejme, že $(p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$ není pravda. Potom:
2. $p \leftrightarrow (\neg q \vee r)$ je pravda - plyne z 1.
3. $\neg p \rightarrow q$ není pravda - plyne z 1.
4. $\neg p$ je pravda - plyne z 3.
5. q není pravda - plyne z 3.
6. p není pravda - plyne z 4.
7. $\neg q$ je pravda - plyne z 5.
8. $\neg q \vee r$ je pravda - plyne z 7.
9. $\neg q \vee r$ není pravda - plyne z 2. a 6.
10. Není možné, aby $(p \leftrightarrow (\neg q \vee r)) \rightarrow (\neg p \rightarrow q)$ nebyla pravda, protože v takovém případě by $\neg q \vee r$ musela být současně pravda i nepravda (viz 8. a 9.).

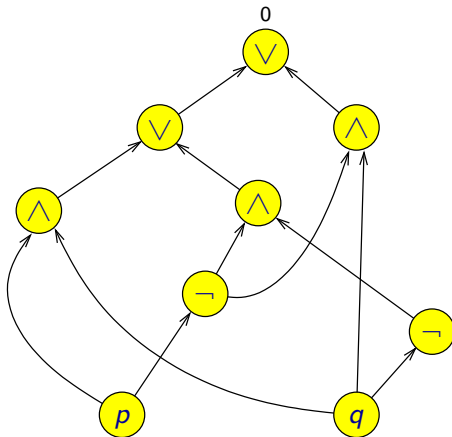
Poznámka: Všimněme si, že v tomto zdůvodnění se vůbec nemluví o grafu reprezentujícím danou formuli, ale jen o pravdivosti a nepravdivosti jejích různých podformulí.

Je daná formule tautologií?

- Zdaleka ne vždy to vychází tak, že hodnoty přiřazené jednotlivým vrcholům jsou jednoznačně určeny dříve přiřazenými hodnotami
- Pokud se dojde do situace, kdy nelze žádnému dalšímu vrcholu přiřadit hodnotu, je třeba zkusit více možností.
- Zvolí se nějaký vrchol a nějaká hodnota, která se mu přiřadí, a odvodí se další hodnoty, které musí být přiřazeny vrcholu v tomto případě.
- Pokud se nenajde hledané ohodnocení, je třeba se vrátit, přiřadit danému vrcholu jinou hodnotu, a vyzkoušet tuto možnost.

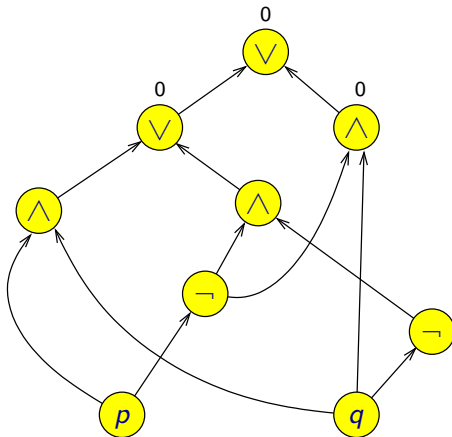
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



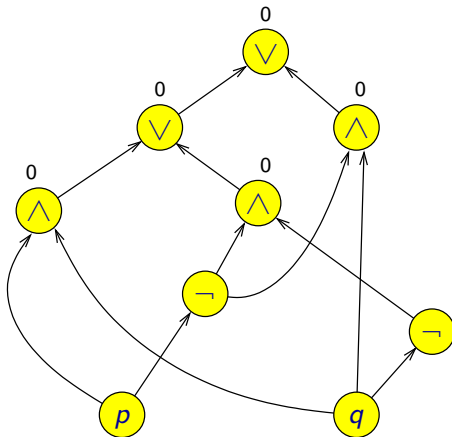
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



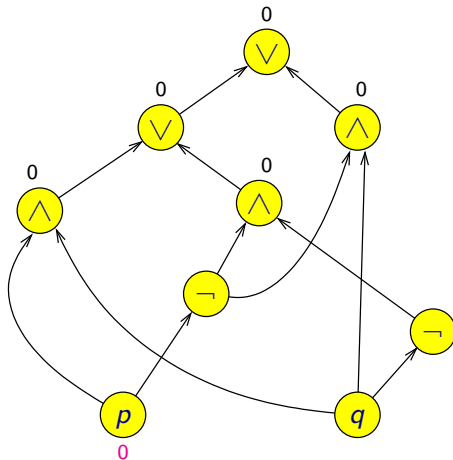
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Je daná formule tautologií?

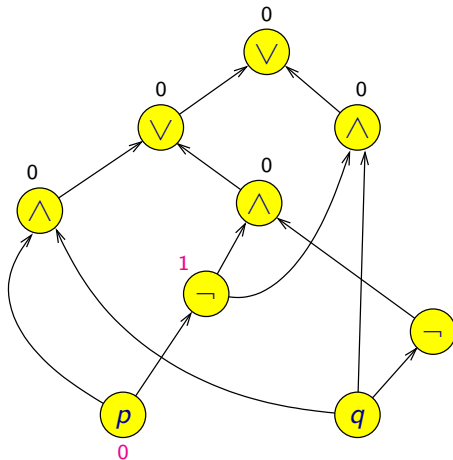
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

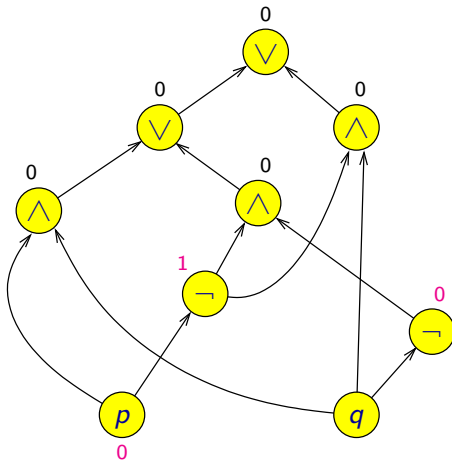
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

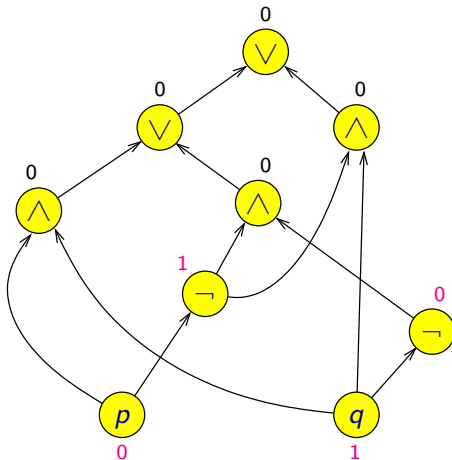
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

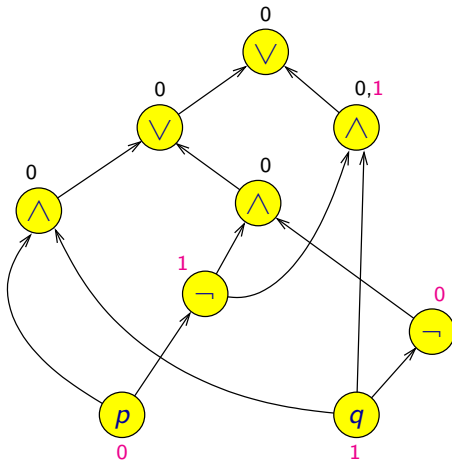
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

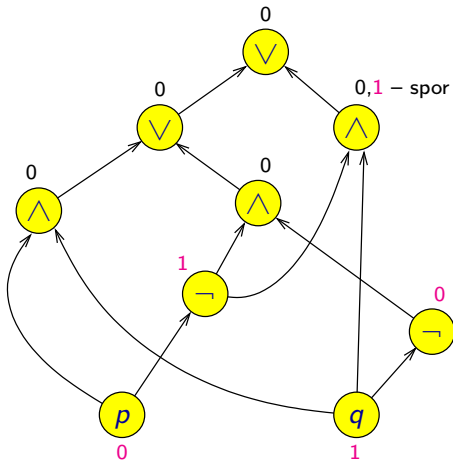
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

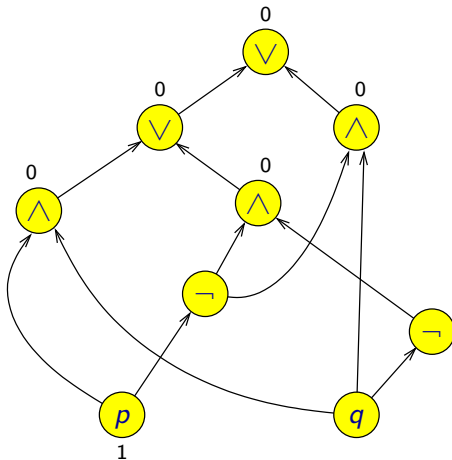
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Zkusíme vrcholu p přiřadit hodnotu 0.

Je daná formule tautologií?

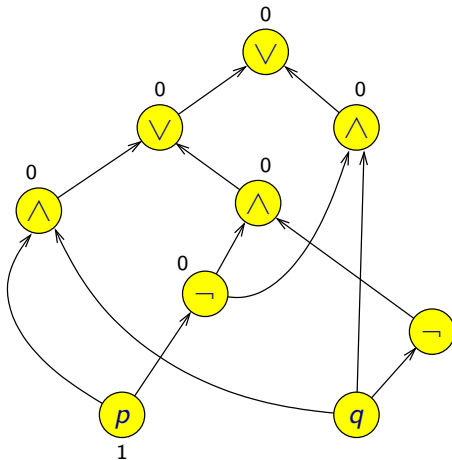
Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Vrchol p tedy musí mít hodnotu **1**.

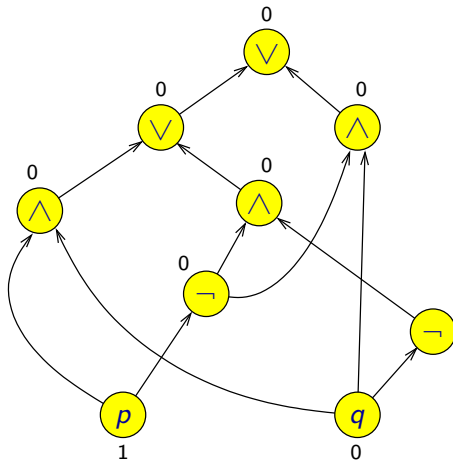
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



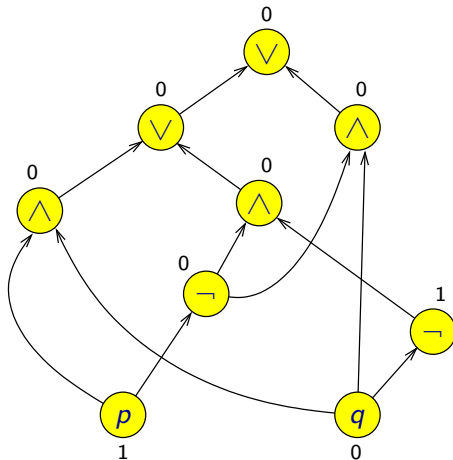
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



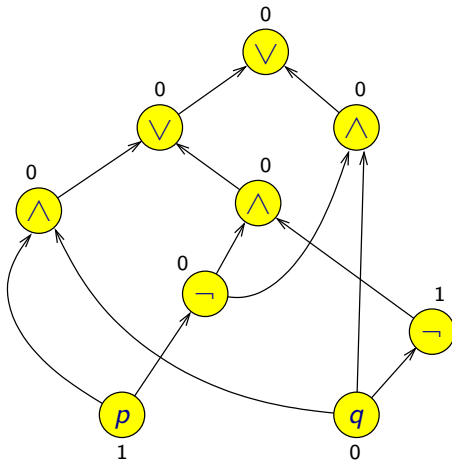
Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Je daná formule tautologií?

Příklad: $\varphi_3 := ((p \wedge q) \vee (\neg p \wedge \neg q)) \vee (\neg p \wedge q)$



Formule φ_3 není tautologie — není pravdivá při ohodnocení v , kde $v(p) = 1$, $v(q) = 0$.

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.

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

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.

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

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

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 $\leftrightarrow$:

$$(p \leftrightarrow q) \leftrightarrow r \Leftrightarrow p \leftrightarrow (q \leftrightarrow r)$$

asociativita

$$p \leftrightarrow q \Leftrightarrow q \leftrightarrow p$$

komutativita

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

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

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

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

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

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

Náhrada atomických výroků:

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

Dostaneme

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

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

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

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

Dostaneme

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

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

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

Příklad: Zdůvodnění toho, že platí

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

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

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:

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

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:

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

Příklad:

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

Pro některé účely se hodí zavést následující dvě speciální formule:

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

Pro každé pravdivostní ohodnocení v tedy platí:

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

Symbole $\top$ a $\perp$ je možné chápat jako „zkratky“:

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

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

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

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

$$\top \Leftrightarrow p \vee \neg p$$

$$\neg \top \Leftrightarrow \perp$$

$$p \wedge \top \Leftrightarrow p$$

$$p \vee \top \Leftrightarrow \top$$

$$\perp \Leftrightarrow p \wedge \neg p$$

$$\neg \perp \Leftrightarrow \top$$

$$p \vee \perp \Leftrightarrow p$$

$$p \wedge \perp \Leftrightarrow \perp$$

Ekvivalence formulí

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

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

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

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.:

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

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

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

Konjunkce a disjunkce více formulí

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

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

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 $\varphi_1, \varphi_2, \dots, \varphi_n$, kde $n \geq 0$, budeme rozumět formuli

$$\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$$

Speciálně:

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

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

$$\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n$$

Speciálně:

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

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

- Celá formule je pravdivá právě tehdy, když všechny formule $\varphi_1, \varphi_2, \dots, \varphi_n$ jsou pravdivé.
- Pokud je některá formule φ_i ekvivalentní $\perp$, je celá formule ekvivalentní $\perp$.
- Pokud je některá formule φ_i ekvivalentní negaci nějaké formule φ_j (tj. $\varphi_i \Leftrightarrow \neg \varphi_j$), pak celá formule je ekvivalentní $\perp$.
- Pokud je některá formule φ_i ekvivalentní $\top$, je možné formuli φ_i z celé formule vypustit.

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

- Celá formule je pravdivá právě tehdy, když alespoň jedna z formulí $\varphi_1, \varphi_2, \dots, \varphi_n$ 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 $\perp$, je možné formuli φ_i z celé formule vypustit.

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

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

Příklad:

- Elementární konjunkce

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

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

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

- Elementární disjunkce

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

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

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

- **Disjunktivní normální forma (DNF)** — disjunkce nula 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 nula nebo více elementárních disjunkcí (klausulí), např.

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

Poznámka: Formule $\perp$ je tedy speciálním případem formule v DNF a formule $\top$ speciálním případem formule v KNF.

Normální formy formulí

Formule v KNF je **tautologie** právě tehdy, když pro každou elementární disjunkci v této formuli platí, že existuje nějaký atomický výrok p takový, že se v dané elementární disjunkci zároveň vyskytují literály p i $\neg p$.

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

Formule v DNF je **kontradikce** právě tehdy, když pro každou elementární konjunkci v této formuli platí, že existuje nějaký atomický výrok p takový, že se v dané elementární konjunkci zároveň vyskytují literály p i $\neg p$.

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

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

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

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

Normální formy formulí

...

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

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

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

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

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

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

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

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:

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

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

- “ $\rightarrow$ ”, “ $\perp$ ”:

$\neg\varphi$ je možno vyjádřit jako $\varphi \rightarrow \perp$

$\varphi \vee \psi$ je možno vyjádřit jako $(\varphi \rightarrow \perp) \rightarrow \psi$

$\varphi \wedge \psi$ je možno vyjádřit jako $(\varphi \rightarrow (\psi \rightarrow \perp)) \rightarrow \perp$

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

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

$\neg\varphi$ je možno vyjádřit jako $\varphi | \varphi$

$\varphi \vee \psi$ je možno vyjádřit jako $(\varphi | \varphi) | (\psi | \psi)$

$\varphi \wedge \psi$ je možno vyjádřit jako $(\varphi | \psi) | (\varphi | \psi)$

- “ $\downarrow$ ” — 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 $(\varphi \downarrow \varphi) \downarrow (\psi \downarrow \psi)$

Definice

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

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

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

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

Příklad: Závěr $r \rightarrow p$ logicky vyplývá z předpokladu $p \vee (q \wedge \neg r)$, tj.

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

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

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.*

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

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

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

*

Pro zjištění toho, zda ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ je možné použít **tabulkovou metodu**:

- Pokud ve všech řádcích odpovídajících ohodnocením, kde všechny předpoklady $\varphi_1, \varphi_2, \dots, \varphi_n$ mají hodnotu 1, má také ψ hodnotu 1, tak závěr ψ **logicky vyplývá** z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$.
- Pokud existuje v tabulce alespoň jedno ohodnocení, kde $\varphi_1, \varphi_2, \dots, \varphi_n$ mají hodnotu 1 a závěr ψ má hodnotu 0, pak tento závěr z těchto předpokladů **logicky nevyplývá**.

Poznámka: Ohodnocení, kde má alespoň jeden z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ hodnotu 0, nejsou z hlediska logického vyplývání důležité — závěr ψ tam může, ale nemusí platit.

Pro zjištění toho, zda ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ je možné také použít **sémantický spor**:

- Vytvoříme orientovaný acyklický graf společný pro všechny formule $\varphi_1, \varphi_2, \dots, \varphi_n$ a ψ .
- Vrcholům odpovídajícím předpokladům $\varphi_1, \varphi_2, \dots, \varphi_n$ přiřadíme hodnoty 1 a vrcholu, který odpovídá závěru ψ , přiřadíme 0.
- Pokud se podaří všechny vrcholy grafu konzistentně ohodnotit, tak máme příklad ohodnocení, při kterém předpoklady platí a závěr ne — tj. závěr z daných předpokladů **logicky nevyplývá**.
- Pokud ukážeme, že takové ohodnocení nemůže existovat (protože každý pokus o doplnění hodnot k ostatním vrcholům vede ke sporu), závěr z daných předpokladů **logicky vyplývá**.

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

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

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.

Otázku, zda daný závěr vyplývá z daných předpokladů, je možné přeformulovat jako otázku, zda určitá formule je tautologie:

$$\varphi_1, \varphi_2, \varphi_3, \dots, \varphi_n \models \psi$$

právě tehdy, když

$\varphi_1 \rightarrow (\varphi_2 \rightarrow (\varphi_3 \rightarrow (\dots \rightarrow (\varphi_n \rightarrow \psi) \dots)))$ je tautologie

Příklad:

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

právě tehdy, když

$\varphi_1 \rightarrow (\varphi_2 \rightarrow (\varphi_3 \rightarrow (\varphi_4 \rightarrow \psi)))$ je tautologie

Následující dvě formule jsou logicky ekvivalentní:

- $\varphi_1 \rightarrow (\varphi_2 \rightarrow (\varphi_3 \rightarrow (\dots \rightarrow (\varphi_n \rightarrow \psi) \dots)))$
- $(\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n) \rightarrow \psi$

Například

$$\varphi_1 \rightarrow (\varphi_2 \rightarrow (\varphi_3 \rightarrow (\varphi_4 \rightarrow \psi))) \Leftrightarrow (\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4) \rightarrow \psi$$

(Dá se to snadno odvodit pomocí ekvivalentních úprav s použitím následující ekvivalence: $p \rightarrow (q \rightarrow r) \Leftrightarrow (p \wedge q) \rightarrow r$)

Platí tedy také, že

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

právě tehdy, když

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

Ještě další možnost, jak charakterizovat, kdy závěr vyplývá z daných předpokladů, je dána následující ekvivalencí:

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

právě tehdy, když

$$\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n \Leftrightarrow \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n \wedge \psi$$

Přidáním nějakého závěru ψ , který logicky vyplývá z daných předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$, k těmto předpokladům jako nový další předpoklad se nijak nezmění množina pravdivostních ohodnocení, při kterých jsou dané předpoklady pravdivé.

(Množiny předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ a $\varphi_1, \varphi_2, \dots, \varphi_n, \psi$ jsou pravdivé při těch samých ohodnoceních.)

Přidáním předpokladu ψ , který logicky vyplývá z daných předpokladů, se tedy nezmění ani množina všech možných závěrů, které z dané množiny předpokladů vyplývají:

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

právě tehdy, když

pro každou formuli χ platí:

$$\varphi_1, \varphi_2, \dots, \varphi_n \models \chi \quad \text{právě tehdy, když} \quad \varphi_1, \varphi_2, \dots, \varphi_n, \psi \models \chi$$

Logické vyplývání

Když z daných předpokladů logicky vyplývají nějaké závěry a z těchto závěrů pak vyplývá nějaký další závěr, vyplývá tento závěr i z původních předpokladů.

Řekněme, že platí

$$\varphi_1, \varphi_2, \dots, \varphi_n \models \chi_1$$

$$\varphi_1, \varphi_2, \dots, \varphi_n \models \chi_2$$

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

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

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

Příklad:

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

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

protože $p \rightarrow q, p \models q$.

Příklad:

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

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

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

Při zdůvodnění toho, že závěr ψ logicky vyplývá z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$, 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:

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

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

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

$$\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$$

je kontradikce.

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

Pokud jsou předpoklady nekonzistentní, vyplývá z nich jakýkoliv závěr.

Pokud jsou předpoklady $\varphi_1, \varphi_2, \dots, \varphi_n$ nekonzistentní, tak pro libovolnou formuli ψ platí

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

Z nekonzistentních předpokladů tak vyplývají například následující závěry:

- $\perp$
- formule χ i $\neg\chi$, kde χ je libovolná formule

Formule $\perp$ nemůže být nikdy pravdivá a stejně tak není možné, aby při nějakém ohodnocení byly současně pravdivé formule χ a $\neg\chi$.

Pokud tedy zjistíme, že z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ logicky vyplývá

- $\perp$ nebo
- χ a zároveň $\neg\chi$,

znamená to, že předpoklady $\varphi_1, \varphi_2, \dots, \varphi_n$ jsou nekonzistentní a vyplývá z nich cokoliv.

Poznámka: Všimněte si, že následující formule jsou tautologie:

- $\perp \rightarrow \psi$
- $\chi \rightarrow (\neg\chi \rightarrow \psi)$

Platí tedy $\perp \models \psi$ a $\chi, \neg\chi \models \psi$.

Princip důkazu sporem

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

právě tehdy, když

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

Zdůvodňování toho, že daný závěr vyplývá z daných předpokladů se tak při použití důkazu sporem převede na zdůvodňování toho, že není možné, aby zároveň platily všechny předpoklady i negace závěru.

Zjišťování toho, zda $\varphi_1, \varphi_2, \dots, \varphi_n \models \psi$, se tak dá převést na zjišťování toho, zda formule

$$\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n \wedge \neg\psi$$

je **kontradikce**.

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

Řeší následující problém:

Vstup: Formule $\varphi_1, \varphi_2, \dots, \varphi_n, \psi$.

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

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.

- 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

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

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

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

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

Těmto formulím se říká **klauzule**.

- Speciálním případem klauzule je **prázdná klauzule** $\perp$, která představuje nalezený spor.
- Algorismus začne svou činnost tím, že formule

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

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

$$\chi_1, \chi_2, \dots, \chi_m.$$

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

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

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

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 $\perp$ a je možné je odstranit.
- Klauzule je možno používat pro aplikaci rezolučního pravidla opakovaně (s jinými klauzulemi).

Speciální případy použití rezolučního pravidla:

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

Z klauzulí

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

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

- Obě klauzule obsahují jeden literál:

Z klauzulí

$$p \qquad \neg p$$

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

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:

$$\neg(j \wedge \neg p)$$

$$\neg j \vee d \vee r$$

$$d \rightarrow \neg p$$

$$j \rightarrow r$$

j – Jana je škole

p – Petr je doma

d – je všední den

r – prší

$$\neg(j \wedge \neg p)$$

$$\neg j \vee d \vee r$$

$$d \rightarrow \neg p$$

$$j \rightarrow r$$

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:

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

Sepíšeme si jednotlivé klauzule:

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

Sepíšeme si jednotlivé klauzule:

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

6. p – rezoluce: 1,4

Sepíšeme si jednotlivé klauzule:

1. $\neg j \vee p$ – předpoklad 1
2. $\neg j \vee d \vee r$ – předpoklad 2
3. $\neg d \vee \neg p$ – předpoklad 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

Sepíšeme si jednotlivé klauzule:

- | | | |
|-------|------------------------|-----------------------------------|
| 1. | $\neg j \vee p$ | – předpoklad 1 |
| 2. | $\neg j \vee d \vee r$ | – předpoklad 2 |
| 3. | $\neg d \vee \neg p$ | – předpoklad 3 |
| 4. | j | – 1. klauzule z negovaného závěru |
| 5. | $\neg r$ | – 2. klauzule z negovaného závěru |
| <hr/> | | |
| 6. | p | – rezoluce: 1,4 |
| 7. | $d \vee r$ | – rezoluce: 2,4 |
| 8. | $\neg d$ | – rezoluce: 3,6 |

Sepíšeme si jednotlivé klauzule:

- | | | |
|-------|------------------------|-----------------------------------|
| 1. | $\neg j \vee p$ | – předpoklad 1 |
| 2. | $\neg j \vee d \vee r$ | – předpoklad 2 |
| 3. | $\neg d \vee \neg p$ | – předpoklad 3 |
| 4. | j | – 1. klauzule z negovaného závěru |
| 5. | $\neg r$ | – 2. klauzule z negovaného závěru |
| <hr/> | | |
| 6. | p | – rezoluce: 1,4 |
| 7. | $d \vee r$ | – rezoluce: 2,4 |
| 8. | $\neg d$ | – rezoluce: 3,6 |
| 9. | r | – rezoluce: 7,8 |

Sepíšeme si jednotlivé klauzule:

- | | | |
|-------|------------------------|-----------------------------------|
| 1. | $\neg j \vee p$ | – předpoklad 1 |
| 2. | $\neg j \vee d \vee r$ | – předpoklad 2 |
| 3. | $\neg d \vee \neg p$ | – předpoklad 3 |
| 4. | j | – 1. klauzule z negovaného závěru |
| 5. | $\neg r$ | – 2. klauzule z negovaného závěru |
| <hr/> | | |
| 6. | p | – rezoluce: 1,4 |
| 7. | $d \vee r$ | – rezoluce: 2,4 |
| 8. | $\neg d$ | – rezoluce: 3,6 |
| 9. | r | – rezoluce: 7,8 |
| 10. | $\perp$ | – rezoluce: 5,9 |

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

Poznámky:

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

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

a která vzniká postupným přidáváním jednotlivých klauzulí.

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

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

$$\varphi_1, \varphi_2, \dots, \varphi_n$$

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 $\varphi_1, \varphi_2, \dots, \varphi_n$.

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

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

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.*

- *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.*

- *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.*

- 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

- 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

- 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

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

$P(x)$	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“)

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.

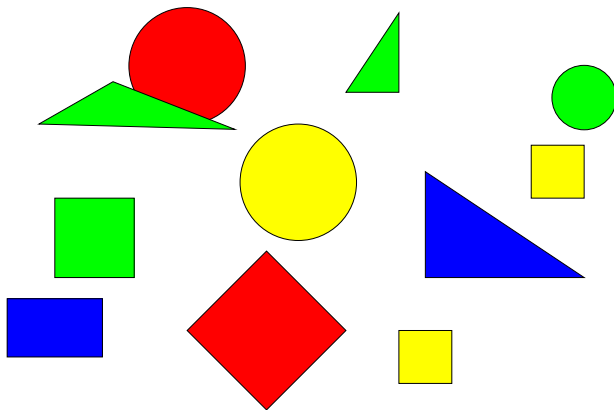
Univerzum — soubor všech objektů v dané interpretaci

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

Valuace — přiřazení prvků universa proměnným

Pravdivostní hodnoty formulí závisí na dané interpretaci a valuaci.

Příklad universa:



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 $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.
- Množina všech bodů v rovině.
- Množina $\{a, b, c, d, e\}$.
- Množina $\{a\}$.

Proměnné — $x, y, z, \dots$, případně s indexy — $x_0, x_1, x_2, \dots$

Předpokládáme, že k dispozici je nekonečný počet proměnných.

Valuace — přiřazení prvků universa proměnným

Příklad:

- Universum — nějaká množina lidí; valuace v , kde:

$$v(x) = \text{„František Vomáčka“}$$

$$v(y) = \text{„Pavla Nováková“}$$

...

- Universum — množina přirozených čísel $\mathbb{N} = \{0, 1, 2, \dots\}$; valuace v , kde

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

Predikáty — $P, Q, R, \dots$

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

Příklad: Predikát P reprezentující vlastnost „být modrý“:

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

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)

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

Příklad: Predikát R reprezentující vztah „být rodičem“:

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

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)

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.

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*“

$P(x)$	—	„ x je modrý“
$P(y)$	—	„ y je modrý“
$Q(y)$	—	„ y je čtverec“
$R(z, x)$	—	„ z překrývá x “
$R(y, y)$	—	„ y překrývá sám sebe“

Poznámka: Později pojem atomické formule poněkud rozšíříme.

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.“

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

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.“

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

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.“

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

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.“

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)

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.“

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

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

Zásadní rozdíl mezi následujícími formulemi:

- $P(x)$ — „ x je čtverec“

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

Pravdivostní hodnota 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.

Pravdivostní hodnota tohoto tvrzení nezávisí na konkrétní valuaci.

Příklad:

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

$$P(x) \rightarrow L(x)$$

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

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

Predikáty:

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

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)

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)

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.“*

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)

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)

Abeceda:

- **logické spojky** — “ $\neg$ ”, “ $\wedge$ ”, “ $\vee$ ”, “ $\rightarrow$ ” a “ $\leftrightarrow$ ”
- **kvantifikátory** — “ $\forall$ ”, “ $\exists$ ”
- **pomocné symboly** — “(”, “)”, “,”
- **proměnné** — “ x ”, “ y ”, “ z ”, $\dots$, “ x_0 ”, “ x_1 ”, “ x_2 ”, $\dots$
- **predikátové symboly** — například symboly “ P ”, “ Q ”, “ R ”, apod.
(u každého symbolu musí být navíc stanovena příslušná arita)
- $\dots$

Poznámka: Další typy symbolů budou uvedeny později.

Definice

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

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

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

Příklad:

$P(x, y)$

$R(z, z, z)$

$S(y)$

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.

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).

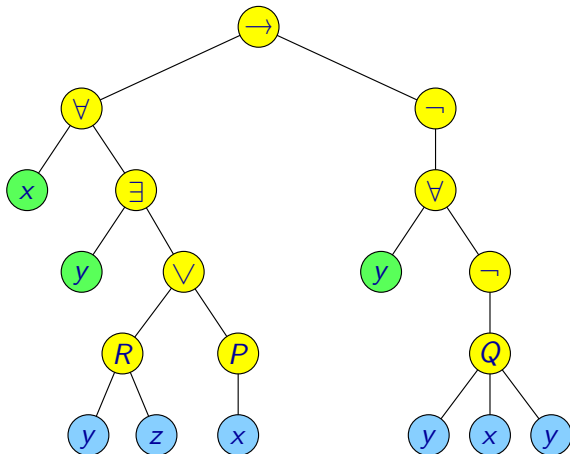
Konvence pro vypouštění závorek:

- Stejně 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

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

Příklad:

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

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

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

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

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

Volné a vázané výskyty proměnných

Formule φ je **uzavřená**, jestliže neobsahuje žádné volné výskyty proměnných (tj. když $\text{free}(\varphi) = \emptyset$).

Formule φ je **otevřená**, jestliže není uzavřená (tj. když $\text{free}(\varphi) \neq \emptyset$).

Poznámka: Uzavřeným formulím se také někdy říká **sentence**.

Příklad:

- Formule $\exists x \exists y P(x, y)$ je uzavřená.
- Formule $\forall x \exists y (R(y, z) \vee P(x)) \rightarrow \neg \forall y \neg Q(y, x, y)$ je otevřená (protože obsahuje volné výskyty proměnných z a x).

Pravdivostní hodnoty uzavřených formulí nezávisí na valuaci, ale jen na příslušné interpretaci.

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

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

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

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

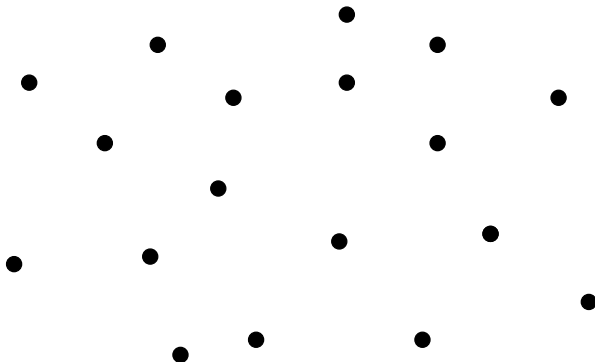
Interpretace $\mathcal{A}$ je struktura skládající se z následujících položek:

- **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.

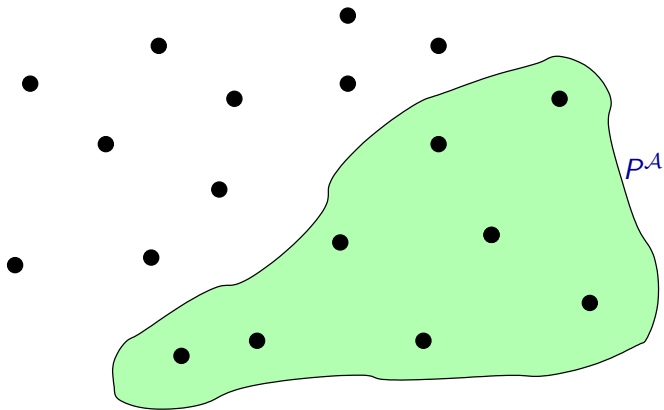
Příklad interpretace $\mathcal{A}$:

universum A



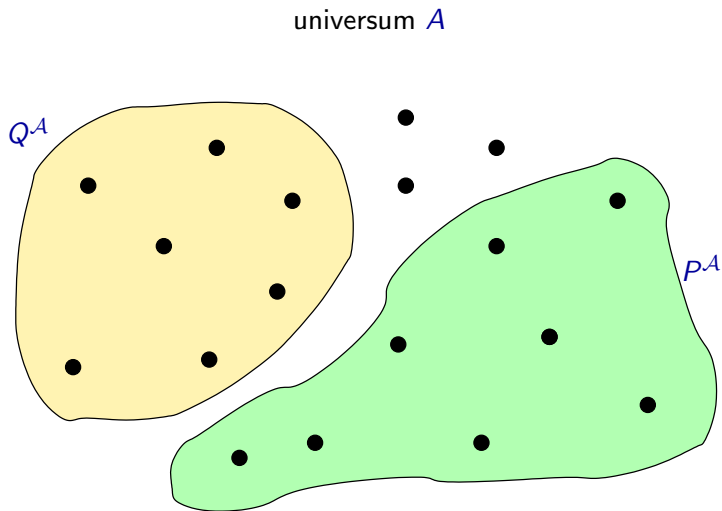
Příklad interpretace $\mathcal{A}$:

universum A



Sémantika predikátové logiky

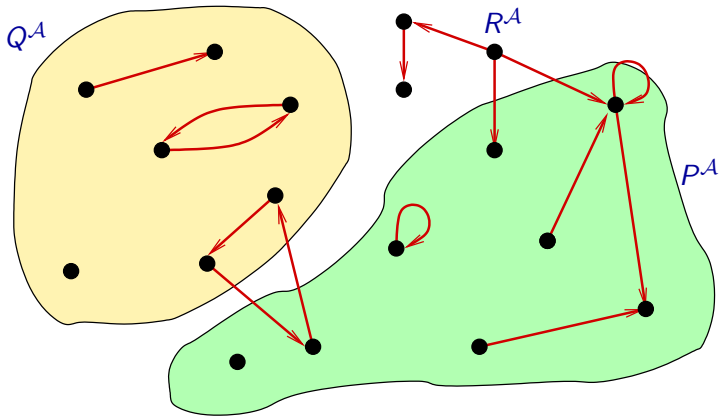
Příklad interpretace $\mathcal{A}$:



Sémantika predikátové logiky

Příklad interpretace $\mathcal{A}$:

universum A



Jiný příklad interpretace $\mathcal{A}$:

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

Označme množinu všech proměnných Var , tj.

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

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

$$v : Var \rightarrow A$$

přiřazující jednotlivým proměnným prvky universa.

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

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

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 $v' : Var \rightarrow A$, 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

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

Příklad:

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

valuace v :

$$v(x_0) = c \quad v(x_1) = e \quad v(x_2) = b \quad v(x_3) = e \quad \dots$$

valuace $v[x_2 \mapsto g]$:

$$v(x_0) = c \quad v(x_1) = e \quad v(x_2) = g \quad v(x_3) = e \quad \dots$$

Definice

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

Pravdivostní hodnoty formulí predikátové logiky v interpretaci $\mathcal{A}$ a valuaci v jsou definovány následovně:

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

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 $\mathcal{A}, v[x \mapsto a] \models \varphi$.

Uzavřená formule φ je **pravdivá** (tj. má pravdivostní hodnotu 1) **v interpretaci** $\mathcal{A}$, jestliže pro každou valuaci v platí $\mathcal{A}, v \models \varphi$.

To, že formule φ je pravdivá v interpretaci $\mathcal{A}$, budeme označovat zápisem

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

Poznámka: U uzavřené formule nezávisí její pravdivostní hodnota v dané interpretaci na tom, jaká je valuace.

Vezměme si libovolnou uzavřenou formuli φ .

Modelem formule φ je libovolná interpretace $\mathcal{A}$ taková, že $\mathcal{A} \models \varphi$.

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.

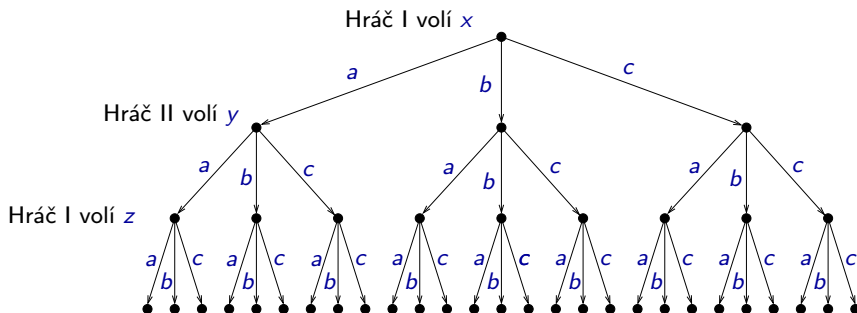
Na vyhodnocování pravdivosti formulí tohoto tvaru v dané interpretaci $\mathcal{A}$ a valuaci v lze nahlížet jako na určitý druh hry:

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



- 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 $x \geq x$.

Formule φ je **logicky platná**, jestliže má v každé interpretaci a valuaci pravdivostní hodnotu 1, tj. jestliže pro každou interpretaci $\mathcal{A}$ a valuaci v platí

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

Příklad:

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

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

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

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 některých 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 $x \notin \text{free}(\varphi)$:

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

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

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 $x \notin \text{free}(\psi)$:

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

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

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

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

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.

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

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

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

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:

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

Přejmenování vázaných proměnných

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

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

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

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

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

Příklad:

$$\exists x\forall yP(x, y) \Leftrightarrow \exists x\forall zP(x, z) \Leftrightarrow \exists y\forall zP(y, z) \Leftrightarrow \exists y\forall xP(y, x)$$

Definice

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

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

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

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

Pokud chceme ukázat, že daný závěr ψ z předpokladů $\varphi_1, \varphi_2, \dots, \varphi_n$ **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.*

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

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

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

$$P(x) \text{ — „}x \text{ je vodní živočich“}$$

$$Q(x) \text{ — „}x \text{ se živí masem“}$$

$$R(x) \text{ — „}x \text{ je ryba“}$$

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

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

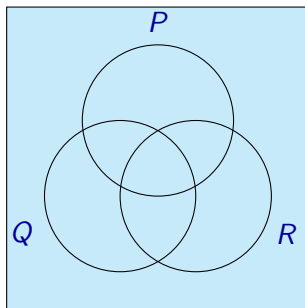
$$Q^{\mathcal{A}} = \{a\}$$

$$R^{\mathcal{A}} = \{b\}$$

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**.



Příklad:

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

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

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

$$\exists xP(x)$$

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

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

$$Q(x) \text{ — „}x \text{ je obratlovec“}$$

$$R(x) \text{ — „}x \text{ žije ve vodě“}$$

⟨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.

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.

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 “.

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 $v(x) = v(y)$.

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 $x \neq y$.

- „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 “:

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

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“:

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

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 ", ...
- ...

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

$$P(c, x)$$

$$Q(d)$$

$$R(a, a)$$

$$x = a$$

- 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 $c^{\mathcal{A}} \in A$.

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“

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

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

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

Platí tedy

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

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

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

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 $S(x, y, z)$.

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

Abeceda:

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

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

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 $P(f(x))$.

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 $G(x_1, x_2, y)$.

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 “

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).

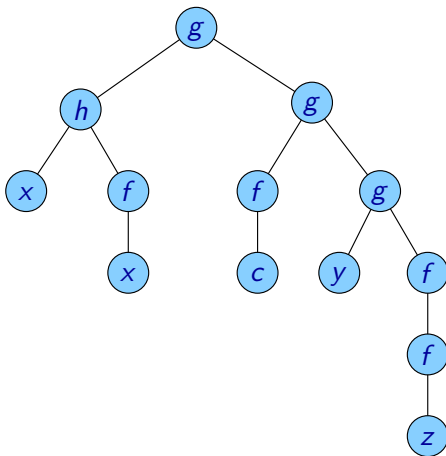
Příklad:

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

Příklady termů:

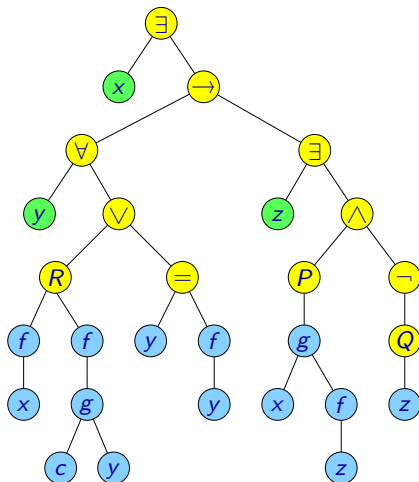
 x $f(y)$ $g(c, x)$ $g(h(x, x), f(c))$ $g(h(x, f(x)), g(f(c), g(y, f(f(z)))))$

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



Syntaktický strom formule

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



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 — konstantní symbol reprezentující prvek “0”

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 “.”)
- R — binární predikátový symbol reprezentující relaci „menší nebo rovno“ (relace “ $\leq$ ”)

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.

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 $t_1, t_2, \dots, t_n$ jsou dobře utvořené termy, tak

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

je dobře utvořený term.

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

Definice

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

- 1 Jestliže P je predikátový symbol s aritou n a $t_1, t_2, \dots, t_n$ jsou dobře utvořené termy, tak

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

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.

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.

Interpretace $\mathcal{A}$:

- universum A
- každému predikátovému symbolu P s aritou n je přiřazena n -ární relace $P^{\mathcal{A}}$, kde $P^{\mathcal{A}} \subseteq A \times A \times \dots \times A$
- každému funkčnímu symbolu f s aritou n je přiřazena n -ární funkce $f^{\mathcal{A}}$, kde $f^{\mathcal{A}} : A \times A \times \dots \times A \rightarrow A$
- 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.

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 $f(t_1, t_2, \dots, t_n)$, kde f je funkční symbol s aritou n a $t_1, t_2, \dots, t_n$ jsou termy — hodnotou tohoto termu je prvek $b \in A$ takový, že

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

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

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

- $a^{\mathcal{A}} = 0$
- $f^{\mathcal{A}}$ 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 ($15 + 1 = 16$)

Pravdivostní hodnoty atomických formulí při interpretaci $\mathcal{A}$ a valuaci v :

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

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

kde $a_1, a_2, \dots, a_n$ jsou hodnoty termů $t_1, t_2, \dots, t_n$ 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 .

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

- $f^{\mathcal{A}}$ 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)

- Jedná se o predikátovou logiku **1. řádu** — kvantifikovat lze pouze přes prvky universa (v predikátové logice 2. řádu je možné kvantifikovat přes relace).

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

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

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

$$x \ f \ y$$

$$x \ R \ y$$

- 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

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

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:

$$E(x, y)$$

- Dvě množiny se rovnají právě tehdy, když 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 :

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

- 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 :

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

- Místo

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

se někdy používá zkrácený zápis

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

Tj. místo

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

se řekne

„existuje $x \in A$ takové, že ...“

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

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

- Místo

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

se někdy používá zkrácený zápis

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

Tj. místo

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

se řekne

„pro každé $x \in A$ platí $\dots$ “

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

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

Formální jazyky

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:

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

Slova v abecedě Σ : AHOJ ABRACADABRA ERROR

Příklad 2:

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

Slovo v abecedě Σ_2 : HELLO_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

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_{  $\leftrightarrow$  _ _ _ _ public _ static _ void _ main(Str...`

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

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.

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).)

$$|\varepsilon| = 0$$

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.

$$\text{OST} \cdot \text{RAVA} = \text{OSTRAVA}$$

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:

$$\text{OST} \cdot \text{RAVA} \neq \text{RAVA} \cdot \text{OST}$$

Zjevně pro libovolná slova v a w platí:

$$|v \cdot w| = |v| + |w|$$

Pro libovolné slovo w také platí:

$$\varepsilon \cdot w = w \cdot \varepsilon = w$$

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**.
- Suffixy 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**.

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 .

Doplňěk – $\overline{L_1}$ 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 .

Příklad:

Uvažujme jazyky nad abecedou $\{a, b\}$.

- L_1 — množina všech slov obsahujících podslovo baa
- L_2 — množina všech slov se sudým počtem výskytů symbolu b

Pak

- $L_1 \cup L_2$ — množina všech slov obsahujících podslovo baa nebo sudý počet symbolů b
- $L_1 \cap L_2$ — množina všech slov obsahujících podslovo baa a sudý počet symbolů b
- $\overline{L_1}$ — množina všech slov, která neobsahují podslovo baa
- $L_1 - L_2$ — množina všech slov, ve kterých se vyskytuje podslovo baa , ale kde počet symbolů b není sudý

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 abbab abbbbb baa baab babbb

Definice

Iterace jazyka L , označovaná zápisem L^* , je jazyk tvořený slovy vzniklými zřetěžením libovolného počtu slov z jazyka L .

Tj. $w \in L^*$ právě tehdy, když

$$\exists n \in \mathbb{N} : \exists w_1, w_2, \dots, w_n \in L : w = w_1 w_2 \cdots w_n$$

Příklad: $L = \{aa, b\}$

$$L^* = \{\varepsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \dots\}$$

Poznámka: Počet slov, která zřetěžujeme, 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 :

$$L^0 = \{\varepsilon\}, \quad L^k = L^{k-1} \cdot L \quad \text{pro } k \geq 1$$

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 aaaab aabaa aabb baaaa baab bbaa bbb

Alternativní definice

Iterace jazyka L je jazyk

$$L^* = \bigcup_{k \geq 0} L^k$$

Poznámka:

$$\bigcup_{k \geq 0} L^k = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$$

Poznámka: Používá se také zápis L^+ jako zkratka za $L \cdot L^*$, tj.

$$L^+ = \bigcup_{k \geq 1} L^k$$

Zrcadlový obraz slova w je slovo w zapsané „pozpátku“.

Zrcadlový obraz slova w značíme w^R .

Příklad: $w = \text{AHOJ}$ $w^R = \text{JOHA}$

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 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 $\Sigma = \{a_1, a_2, \dots, a_n\}$, tak platí

$$a_1 < a_2 < \dots < a_n.$$

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

$$x = uav \quad y = ubw \quad a < b$$

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

$$w_0, w_1, w_2, \dots$$

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:

$$\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, \dots$$

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 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.

Příklad: abeceda $\Sigma = \{0, 1\}$

- Podle definice jsou 0 i 1 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.

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.

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.

Příklad: abeceda $\Sigma = \{0, 1\}$

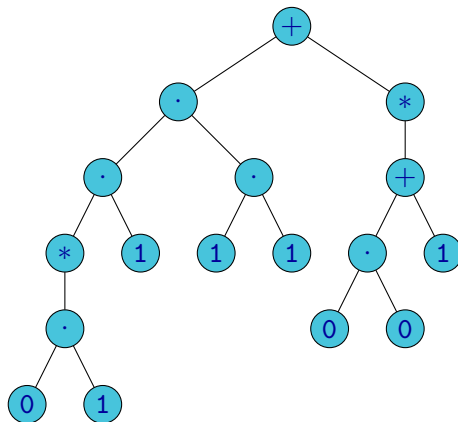
- Podle definice jsou 0 i 1 regulární výrazy.
- Protože 0 i 1 jsou regulární výrazy, je i $(0 + 1)$ regulární výraz.
- Protože 0 je regulární výraz, je i (0^*) regulární výraz.
- Protože $(0 + 1)$ i (0^*) jsou regulární výrazy, je i $((0 + 1) \cdot (0^*))$ regulární výraz.

Poznámka: Jestliže α je regulární výraz, zápisem $[\alpha]$ označujeme jazyk definovaný regulárním výrazem α .

$$[((0 + 1) \cdot (0^*))] = \{0, 1, 00, 10, 000, 100, 0000, 1000, 00000, \dots\}$$

Regulární výrazy

Strukturu regulárního výrazu si můžeme znázornit abstraktním syntaktickým stromem:



$(((((0 \cdot 1)^*) \cdot 1) \cdot (1 \cdot 1)) + (((0 \cdot 0) + 1)^*))$

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 následující 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

$$(01)^*111 + (00 + 1)^*$$

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

Příklady: Ve všech případech $\Sigma = \{0, 1\}$.

0 ... jazyk tvořený jediným slovem 0

01 ... jazyk tvořený jediným slovem 01

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

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 , ...

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 $\varepsilon, 0, 00, 000, \dots$

$(01)^*$... jazyk tvořený slovy $\varepsilon, 01, 0101, 010101, \dots$

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 $\varepsilon, 0, 00, 000, \dots$

$(01)^*$... jazyk tvořený slovy $\varepsilon, 01, 0101, 010101, \dots$

$(0 + 1)^*$... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$

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 $\varepsilon, 0, 00, 000, \dots$

$(01)^*$... jazyk tvořený slovy $\varepsilon, 01, 0101, 010101, \dots$

$(0 + 1)^*$... jazyk tvořený všemi slovy nad abecedou $\{0, 1\}$

$(0 + 1)^*00$... jazyk tvořený všemi slovy končícími 00

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 $\varepsilon, 0, 00, 000, \dots$

$(01)^*$... jazyk tvořený slovy $\varepsilon, 01, 0101, 010101, \dots$

$(0 + 1)^*$... 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

$(0 + 1)^*00 + (01)^*111(01)^*$... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podslovo 111 předcházené i následované libovolným počtem slov 01

$(0 + 1)^*00 + (01)^*111(01)^*$... jazyk tvořený všemi slovy, která buď končí **00** nebo obsahují podслово **111** předcházené i následované libovolným počtem slov **01**

$(0 + 1)^*1(0 + 1)^*$... jazyk tvořený všemi slovy obsahujícími alespoň jeden symbol **1**

$(0 + 1)^*00 + (01)^*111(01)^*$... jazyk tvořený všemi slovy, která buď končí 00 nebo obsahují podслово 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

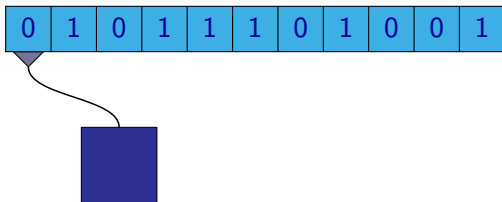
Konečné automaty

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.

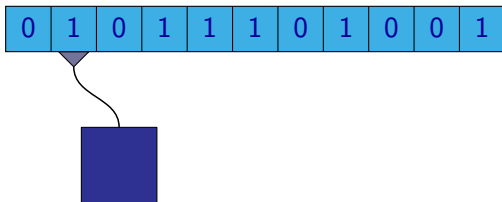


Rozpoznávání jazyka

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.

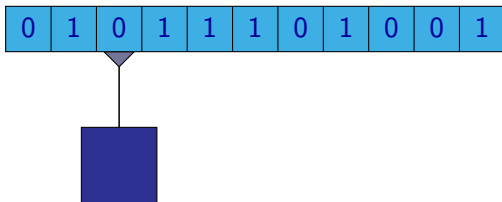


Rozpoznávání jazyka

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.

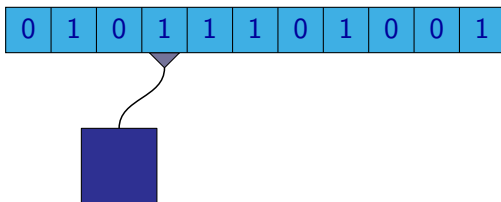


Rozpoznávání jazyka

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.

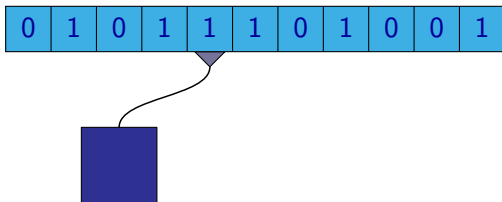


Rozpoznávání jazyka

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.

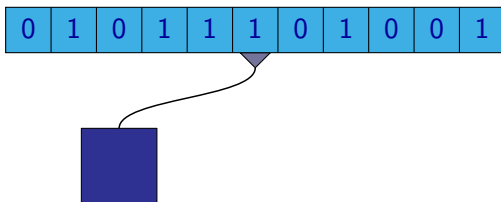


Rozpoznávání jazyka

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.

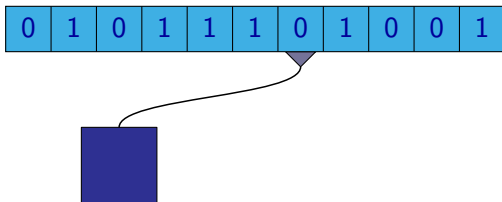


Rozpoznávání jazyka

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.

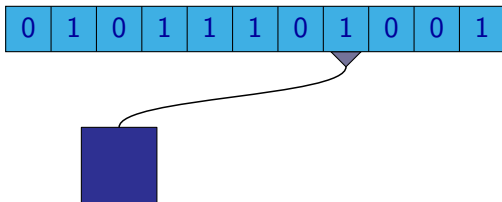


Rozpoznávání jazyka

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.

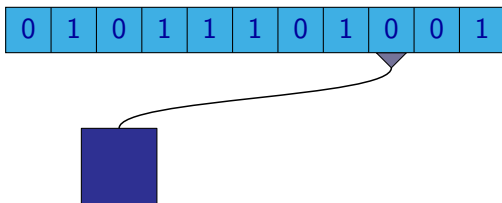


Rozpoznávání jazyka

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.

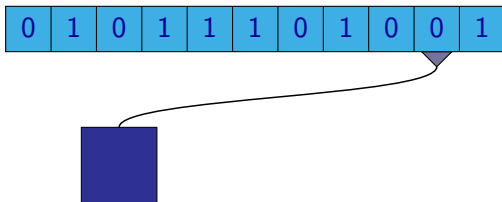


Rozpoznávání jazyka

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.

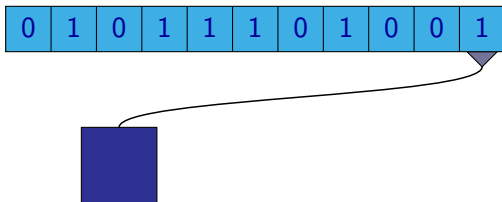


Rozpoznávání jazyka

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.

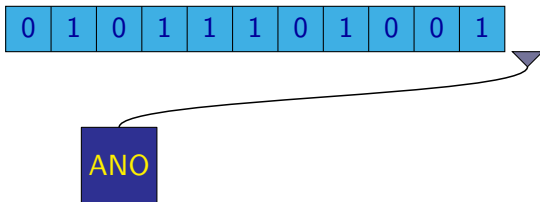


Rozpoznávání jazyka

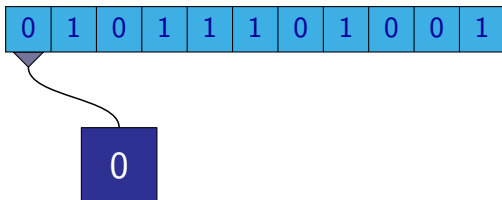
Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

Chtěli bychom rozpoznávat jazyk L , který je tvořen slovy, ve kterých se vyskytuje sudý počet symbolů 1 .

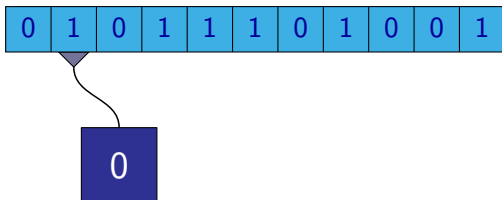
Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.



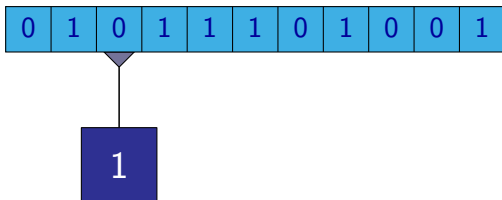
První nápad: Počítat počet výskytů symbolů 1.



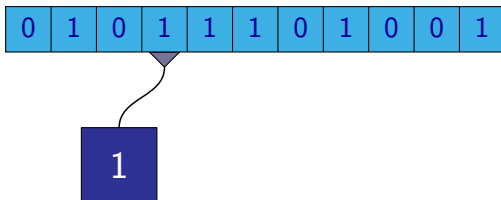
První nápad: Počítat počet výskytů symbolů 1.



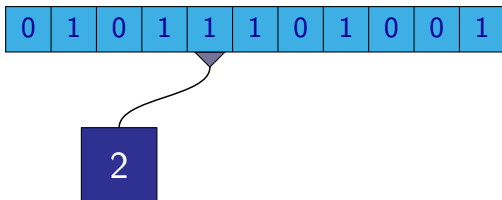
První nápad: Počítat počet výskytů symbolů 1.



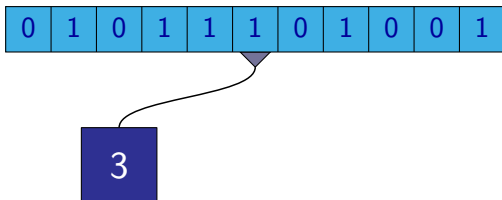
První nápad: Počítat počet výskytů symbolů 1.



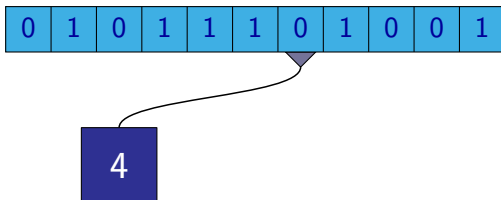
První nápad: Počítat počet výskytů symbolů 1.



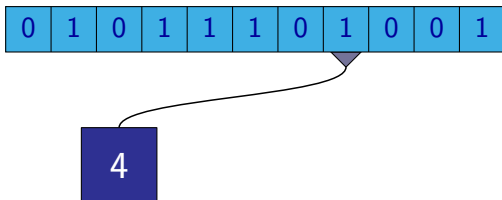
První nápad: Počítat počet výskytů symbolů 1.



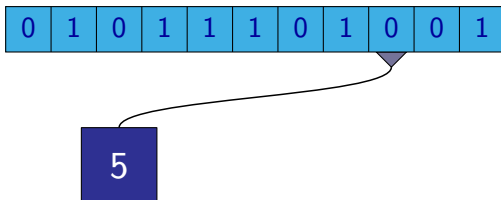
První nápad: Počítat počet výskytů symbolů 1.



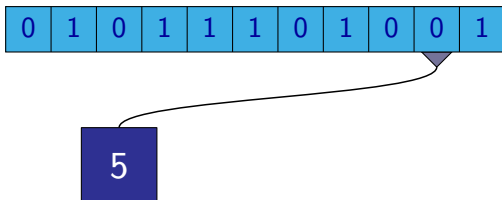
První nápad: Počítat počet výskytů symbolů 1.



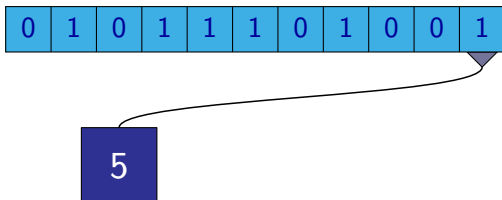
První nápad: Počítat počet výskytů symbolů 1.



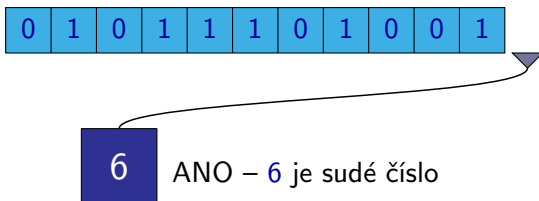
První nápad: Počítat počet výskytů symbolů 1.



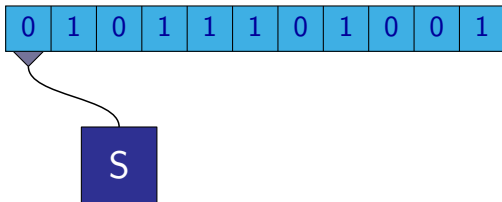
První nápad: Počítat počet výskytů symbolů 1.



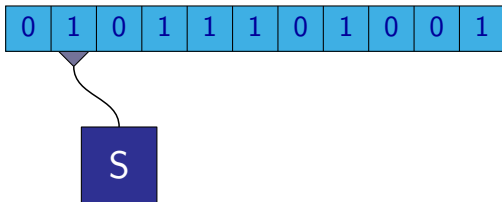
První nápad: Počítat počet výskytů symbolů 1.



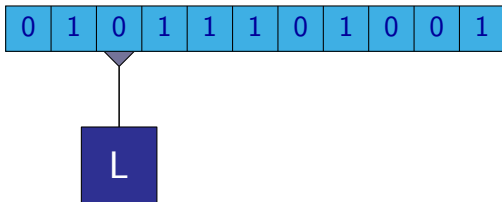
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



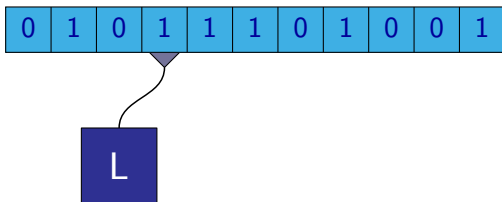
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



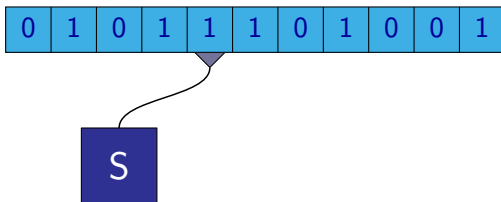
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



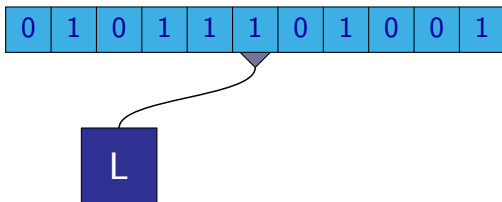
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



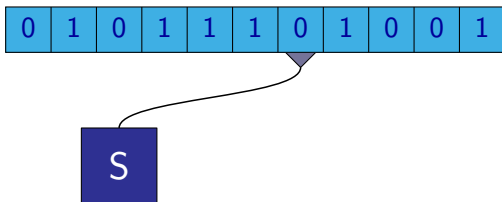
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



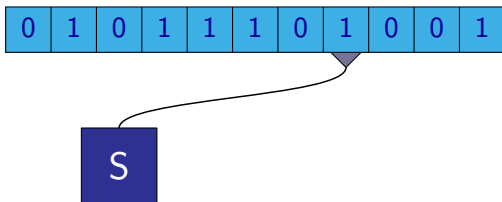
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



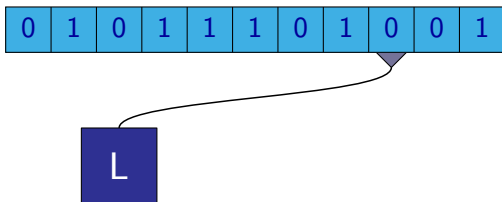
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



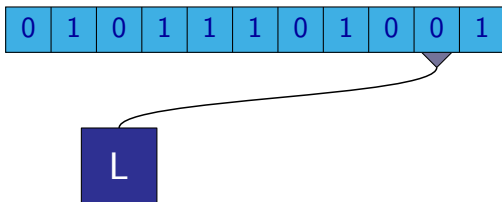
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



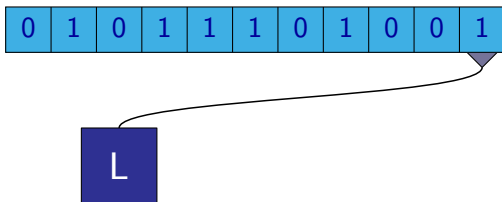
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



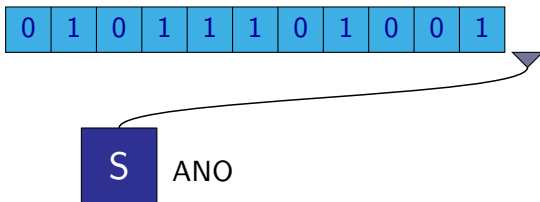
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (tj. místo čísla si stačí pamatovat jen jeho poslední bit).



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).



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).



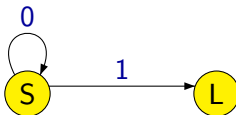
Chování tohoto zařízení můžeme popsat grafem:



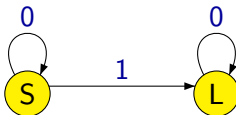
Chování tohoto zařízení můžeme popsat grafem:



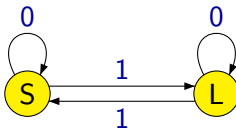
Chování tohoto zařízení můžeme popsat grafem:



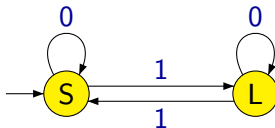
Chování tohoto zařízení můžeme popsat grafem:



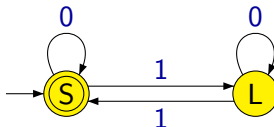
Chování tohoto zařízení můžeme popsat grafem:



Chování tohoto zařízení můžeme popsat grafem:

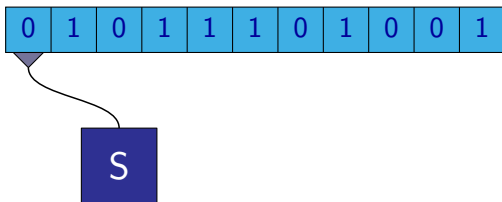
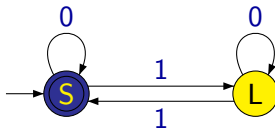


Chování tohoto zařízení můžeme popsat grafem:



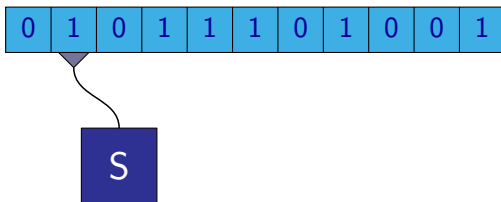
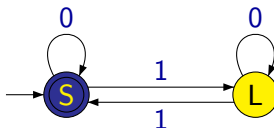
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



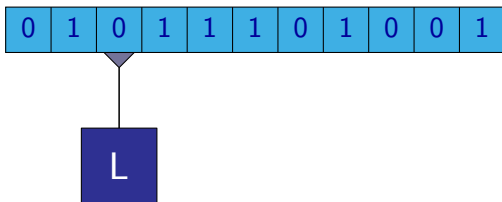
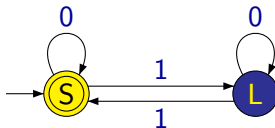
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



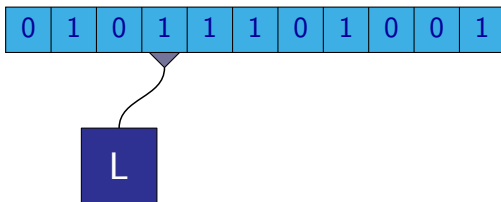
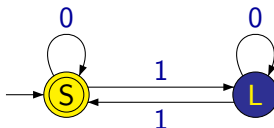
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



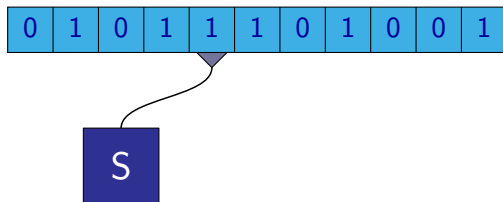
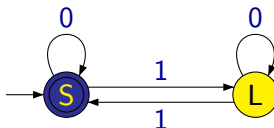
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



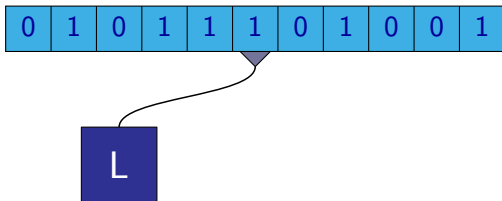
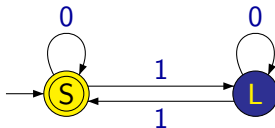
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



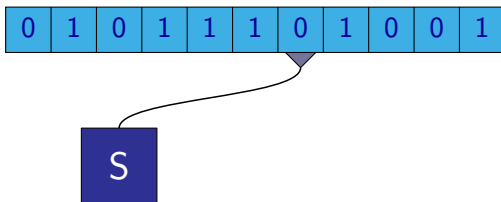
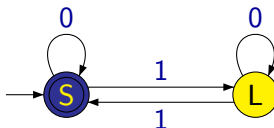
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



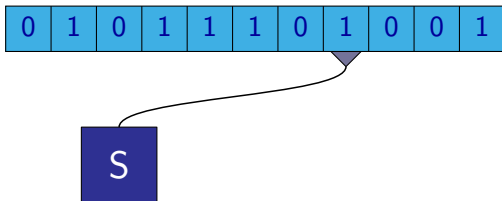
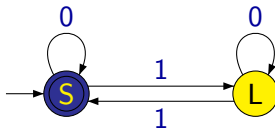
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



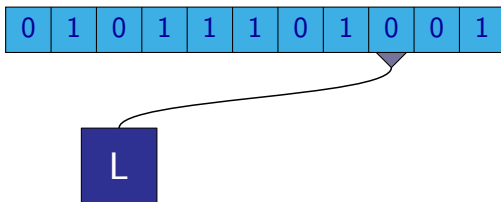
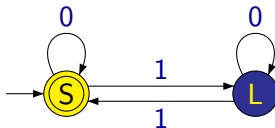
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



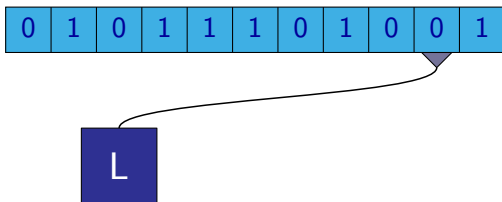
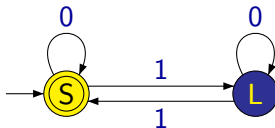
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



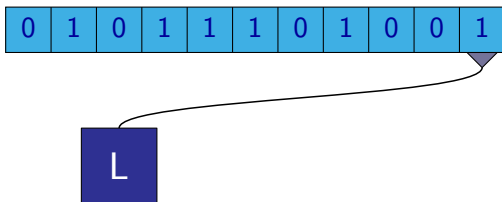
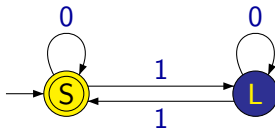
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:



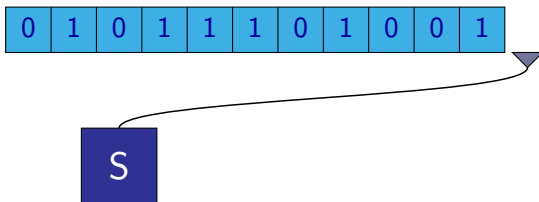
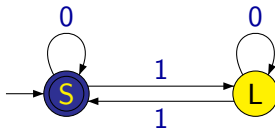
Rozpoznávání jazyka

Chování tohoto zařízení můžeme popsat grafem:

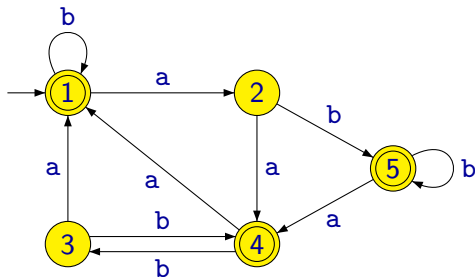


Rozpoznávání jazyka

Chování 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

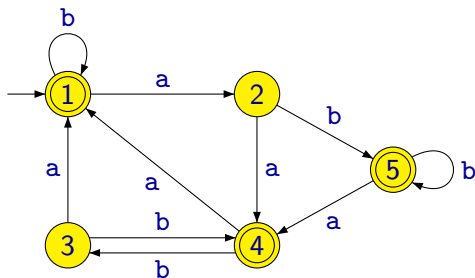
Formálně je **deterministický konečný automat (DKA)** definován jako pětice

$$(Q, \Sigma, \delta, q_0, F)$$

kde:

- Q je neprázdná konečná množina **stavů**
- Σ je **abeceda** (neprázdná konečná množina symbolů)
- $\delta : Q \times \Sigma \rightarrow Q$ je **přechodová funkce**
- $q_0 \in Q$ je **počáteční stav**
- $F \subseteq Q$ je množina **přijímajících stavů**

Deterministický konečný automat



- $Q = \{1, 2, 3, 4, 5\}$

- $\Sigma = \{a, b\}$

- $q_0 = 1$

- $F = \{1, 4, 5\}$

$$\delta(1, a) = 2$$

$$\delta(1, b) = 1$$

$$\delta(2, a) = 4$$

$$\delta(2, b) = 5$$

$$\delta(3, a) = 1$$

$$\delta(3, b) = 4$$

$$\delta(4, a) = 1$$

$$\delta(4, b) = 3$$

$$\delta(5, a) = 4$$

$$\delta(5, b) = 5$$

Deterministický konečný automat

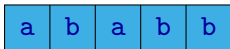
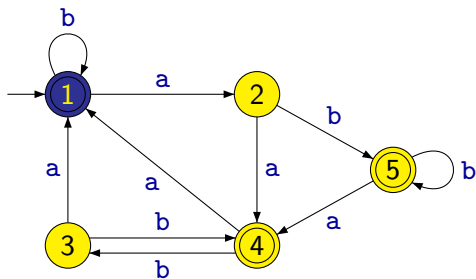
Místo zápisu

$$\begin{array}{ll} \delta(1, a) = 2 & \delta(1, b) = 1 \\ \delta(2, a) = 4 & \delta(2, b) = 5 \\ \delta(3, a) = 1 & \delta(3, b) = 4 \\ \delta(4, a) = 1 & \delta(4, b) = 3 \\ \delta(5, a) = 4 & \delta(5, b) = 5 \end{array}$$

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

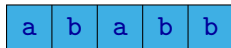
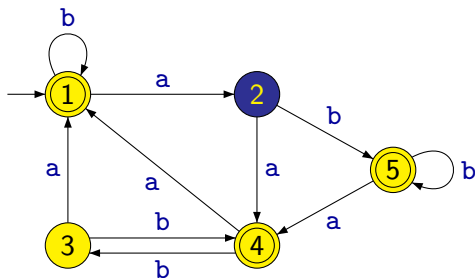
Deterministický konečný automat



1

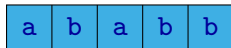
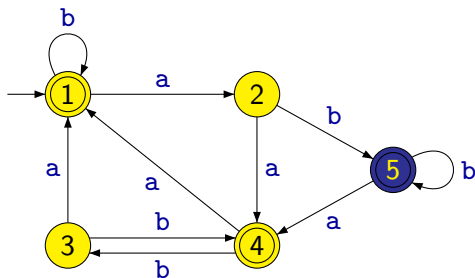


Deterministický konečný automat



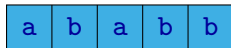
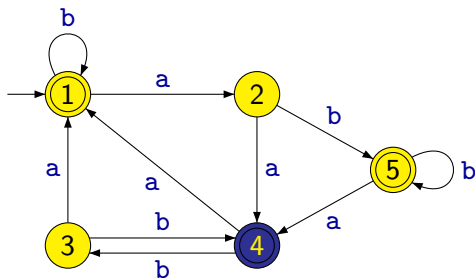
$1 \xrightarrow{a} 2$

Deterministický konečný automat



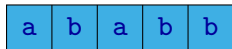
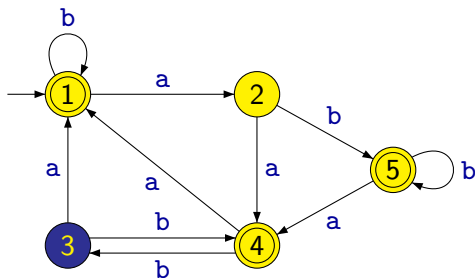
$1 \xrightarrow{a} 2 \xrightarrow{b} 5$

Deterministický konečný automat



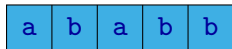
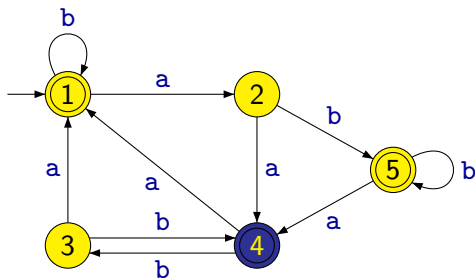
$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4$

Deterministický konečný automat



$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3$

Deterministický konečný automat



$1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{a} 4 \xrightarrow{b} 3 \xrightarrow{b} 4$

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 $q \xrightarrow{w} q'$.

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

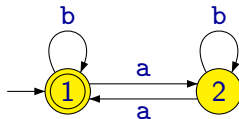
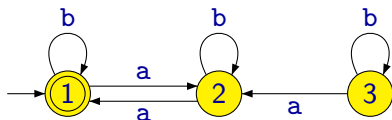
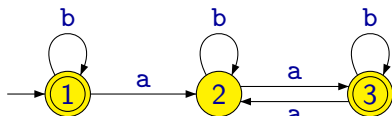
Jazyk rozpoznávaný (přijímaný) daným deterministickým konečným automatem $A = (Q, \Sigma, \delta, q_0, F)$, označovaný $L(A)$, je množina všech slov přijímaných tímto automatem, tj.

$$L(A) = \{w \in \Sigma^* \mid \exists q \in F : q_0 \xrightarrow{w} q\}$$

Definice

Jazyk L je **regulární** právě tehdy, když existuje nějaký deterministický konečný automat A , který jej přijímá, tj. DKA A , takový, že $L(A) = L$.

Ekvivalence automatů

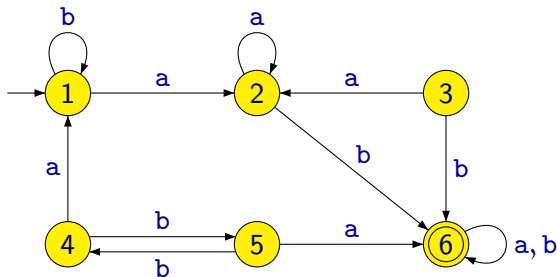


Všechny 3 automaty přijímají jazyk všech slov se sudým počtem a .

Definice

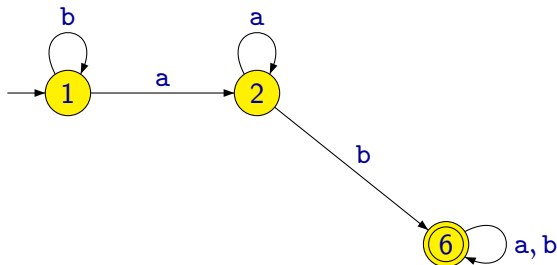
O konečných automatech A_1, A_2 řekneme, že jsou **ekvivalentní**, jestliže $L(A_1) = L(A_2)$.

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 .

Definice

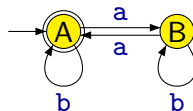
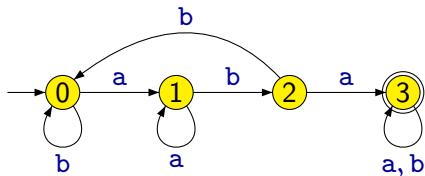
Stav q konečného automatu $A = (Q, \Sigma, \delta, q_0, F)$ je **dosažitelný** pokud existuje nějaké slovo w takové, že $q_0 \xrightarrow{w} q$.

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

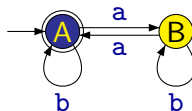
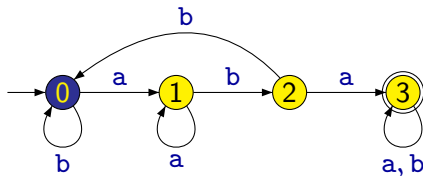
Máme následující dva automaty:



Přijmou oba slovo **ababb**?

Automat pro průnik jazyků

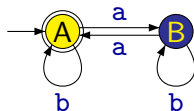
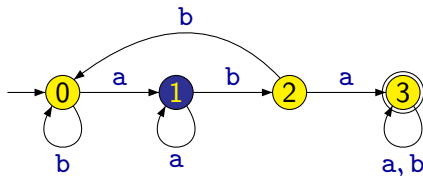
Máme následující dva automaty:



Přijmou oba slovo **a**babbb?

Automat pro průnik jazyků

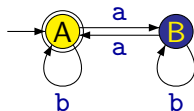
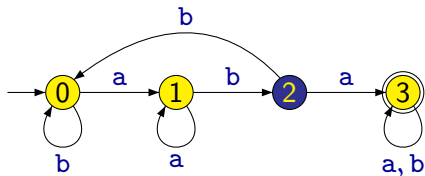
Máme následující dva automaty:



Přijmou oba slovo **a**bab**b**?

Automat pro průnik jazyků

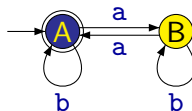
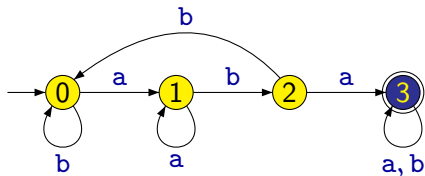
Máme následující dva automaty:



Přijmou oba slovo **ab**abb?

Automat pro průnik jazyků

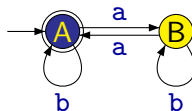
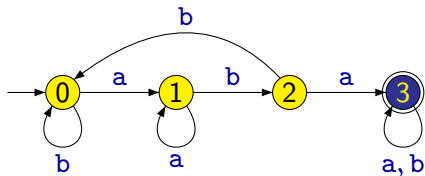
Máme následující dva automaty:



Přijmou oba slovo **abab**b?

Automat pro průnik jazyků

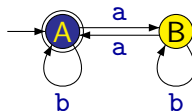
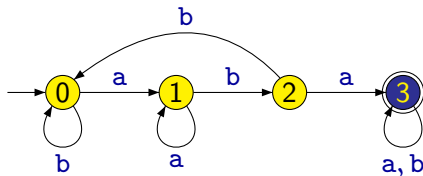
Máme následující dva automaty:



Přijmou oba slovo **abab****b**?

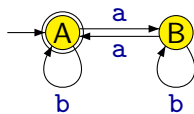
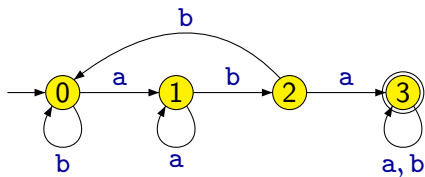
Automat pro průnik jazyků

Máme následující dva automaty:

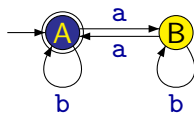
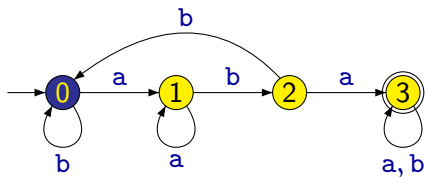


Přijmou oba slovo **ababb**?

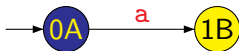
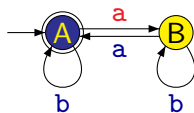
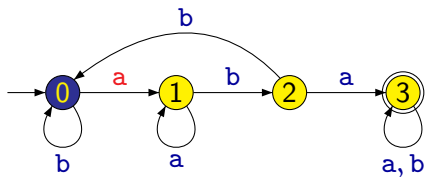
Automat pro průnik jazyků



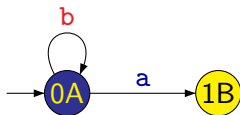
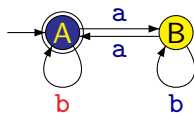
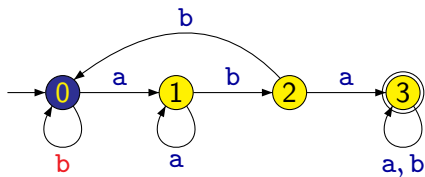
Automat pro průnik jazyků



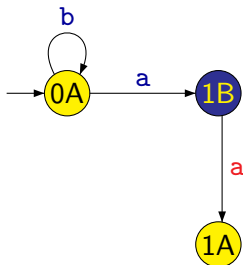
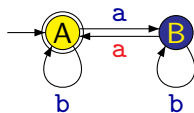
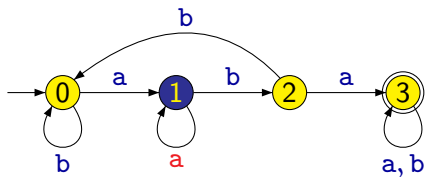
Automat pro průnik jazyků



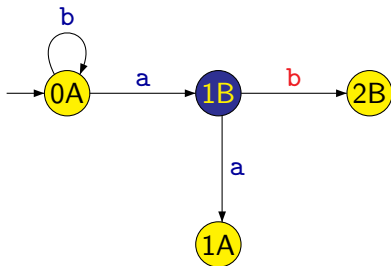
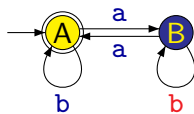
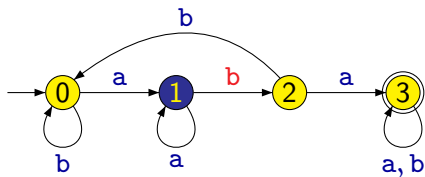
Automat pro průnik jazyků



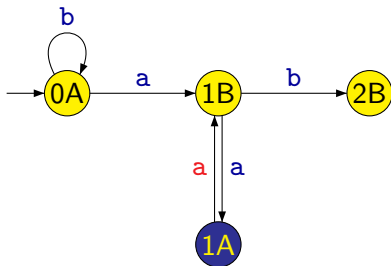
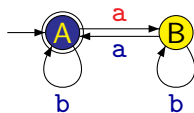
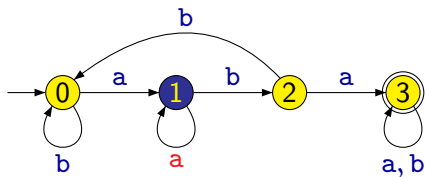
Automat pro průnik jazyků



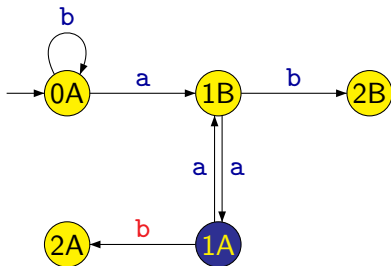
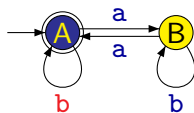
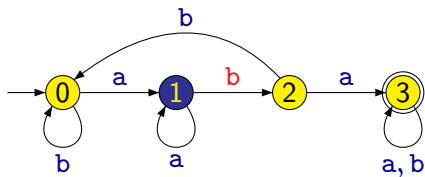
Automat pro průnik jazyků



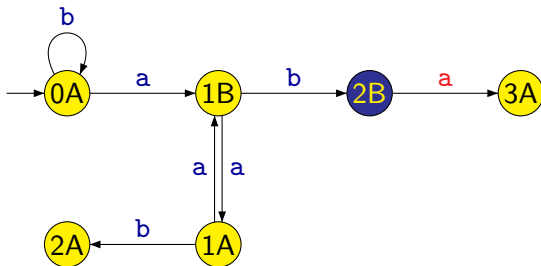
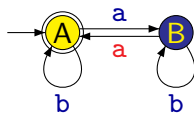
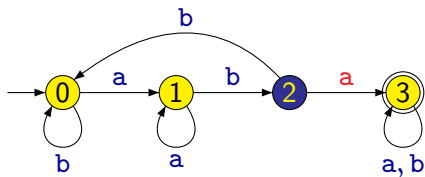
Automat pro průnik jazyků



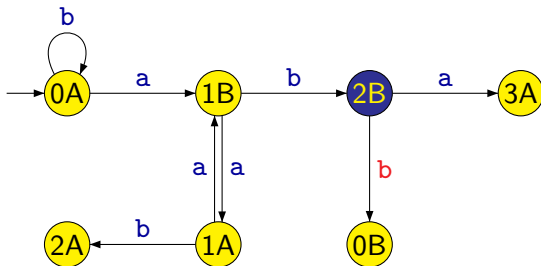
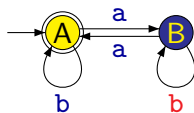
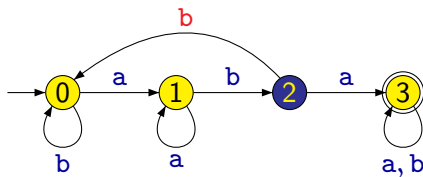
Automat pro průnik jazyků



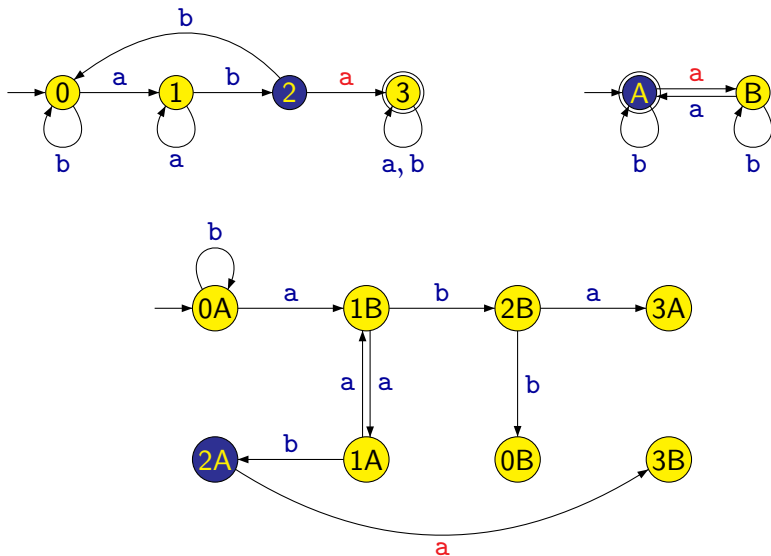
Automat pro průnik jazyků



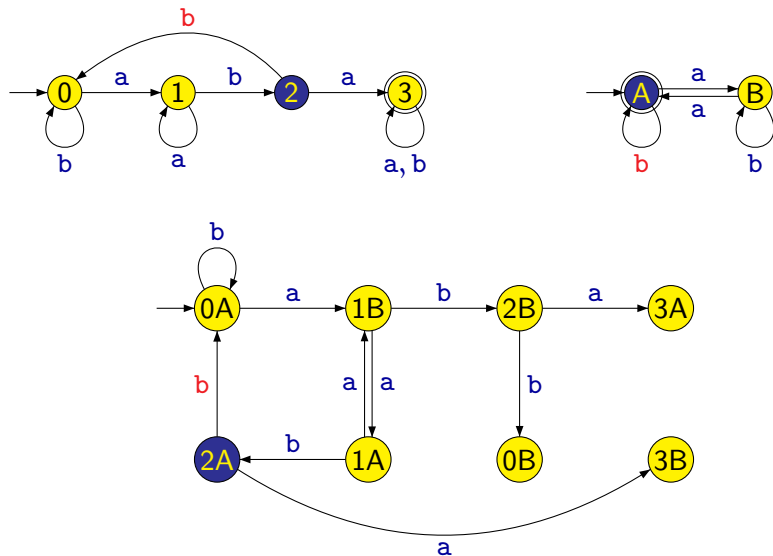
Automat pro průnik jazyků



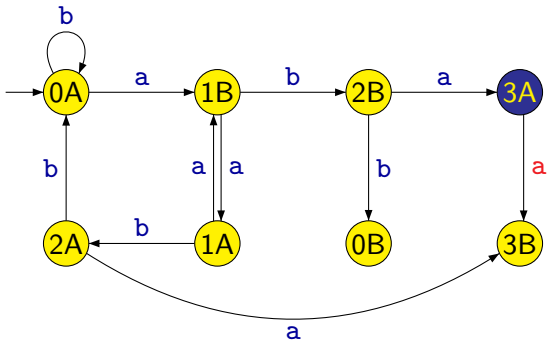
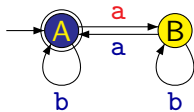
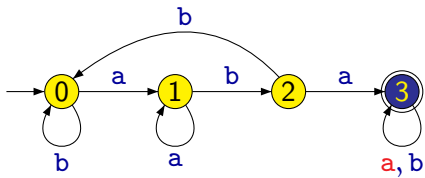
Automat pro průnik jazyků



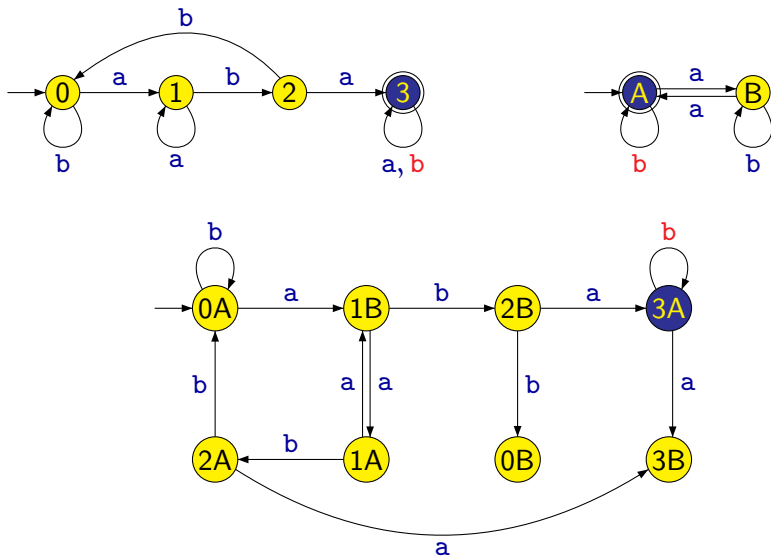
Automat pro průnik jazyků



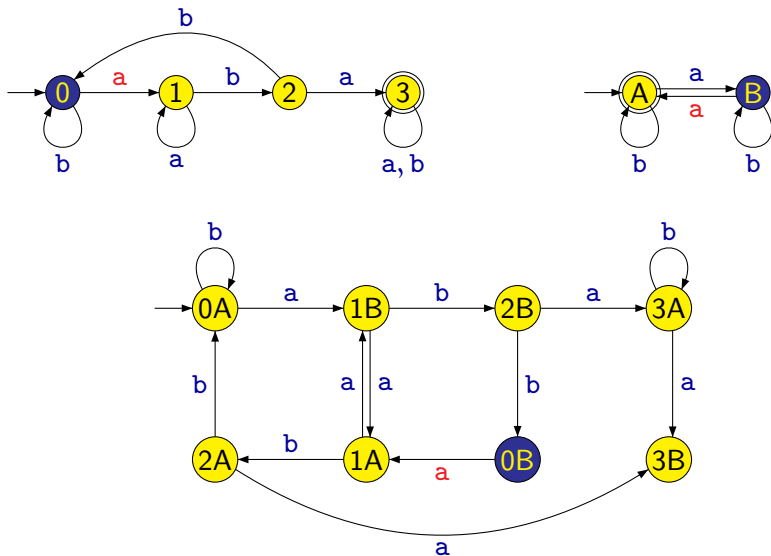
Automat pro průnik jazyků



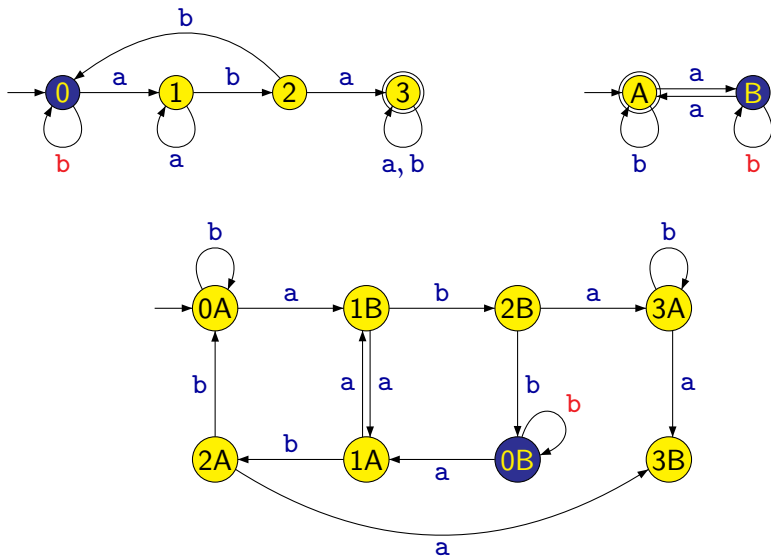
Automat pro průnik jazyků



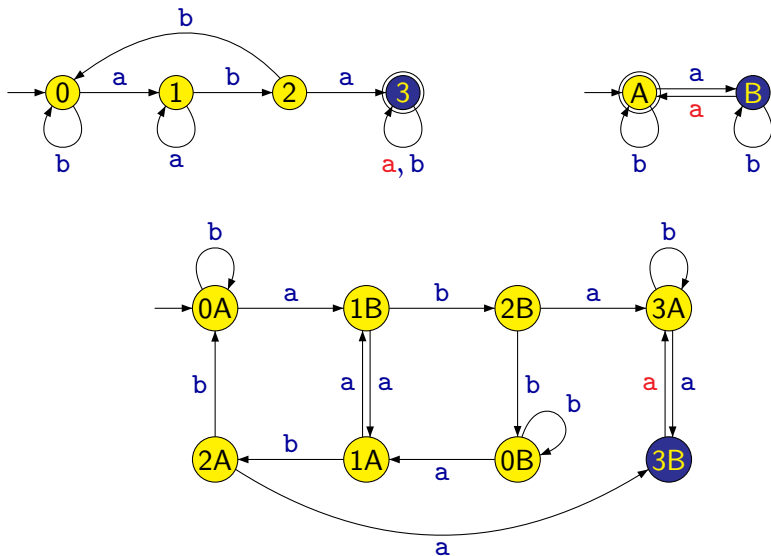
Automat pro průnik jazyků



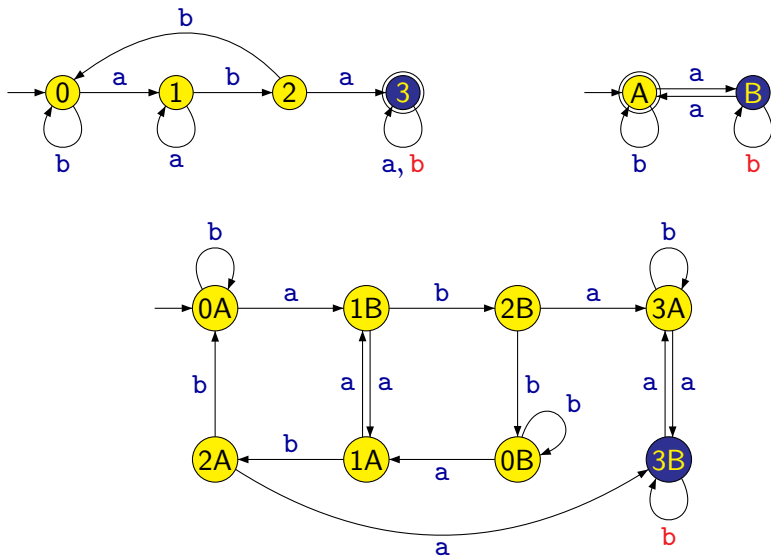
Automat pro průnik jazyků



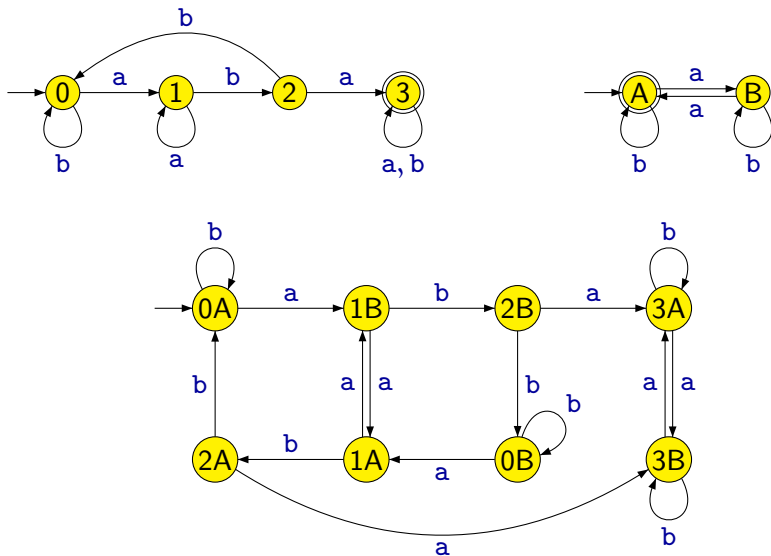
Automat pro průnik jazyků



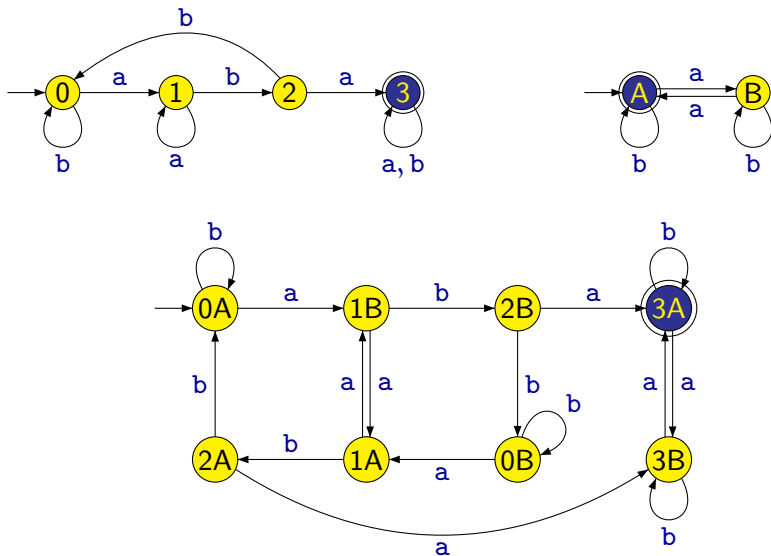
Automat pro průnik jazyků



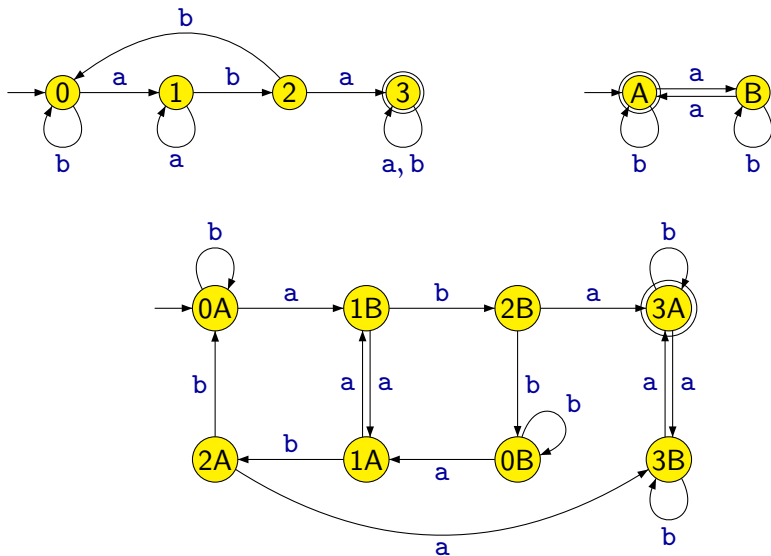
Automat pro průnik jazyků



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 těžké 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.

$$L(A) = L(A_1) \cap L(A_2)$$

Věta

Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cap L_2$ je regulární.

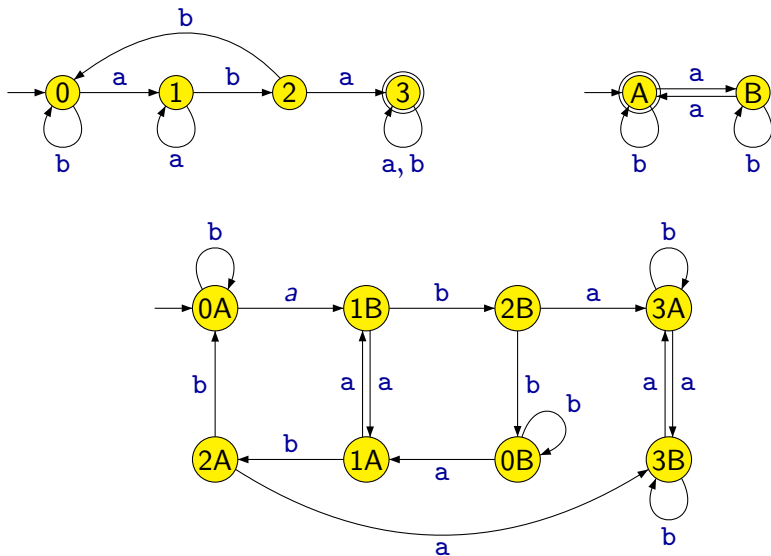
Důkaz: Předpokládejme, že A_1 a A_2 jsou deterministické konečné automaty takové, že

$$L_1 = L(A_1) \qquad L_2 = L(A_2)$$

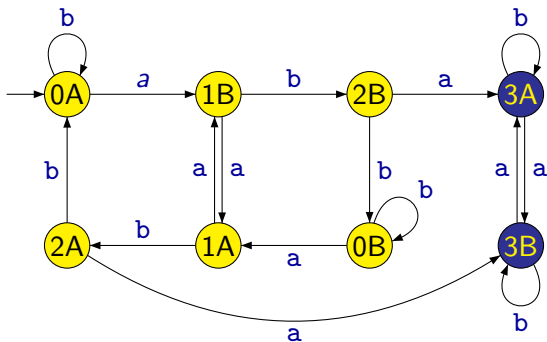
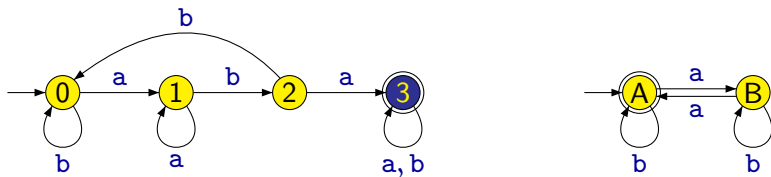
Popsanou konstrukcí k nim můžeme sestrojit deterministický konečný automat A takový, že

$$L(A) = L(A_1) \cap L(A_2) = L_1 \cap L_2$$

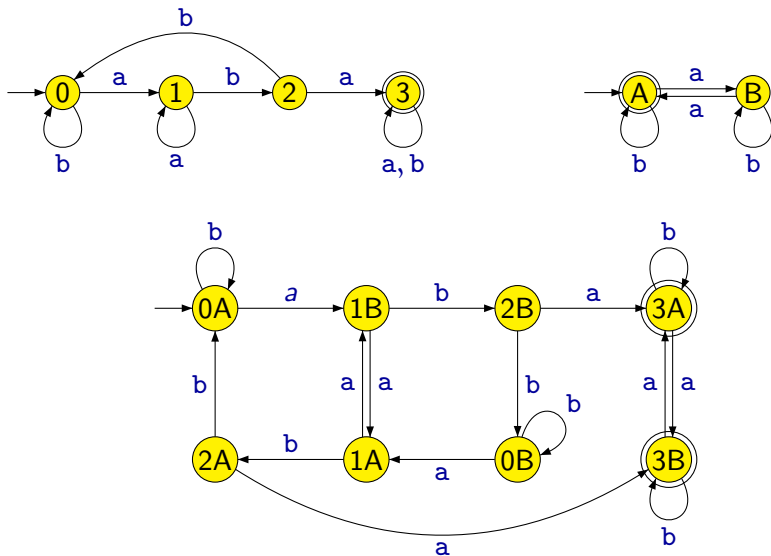
Automat pro sjednocení jazyků



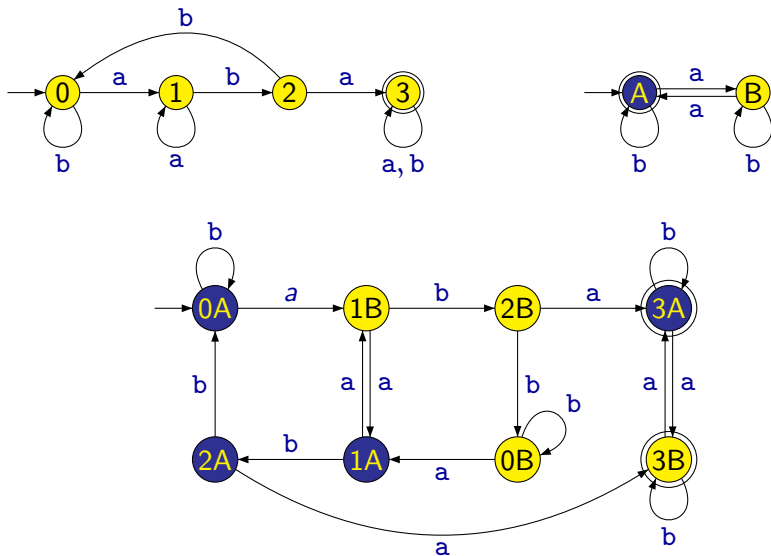
Automat pro sjednocení jazyků



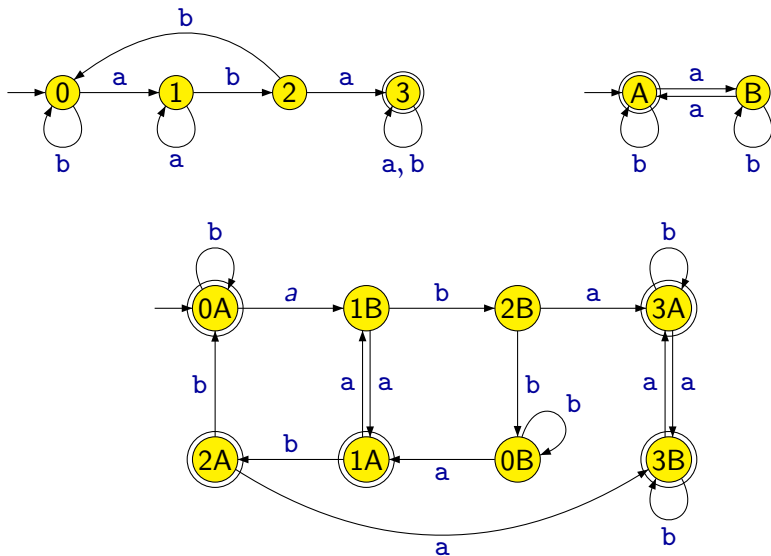
Automat pro sjednocení jazyků



Automat pro sjednocení jazyků



Automat pro sjednocení jazyků



Sjednocení regulárních jazyků

Konstrukce automatu A , který přijímá **sjednocení** jazyků přijímaných automaty A_1 a A_2 , tj. jazyk

$$L(A_1) \cup L(A_2)$$

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

Konstrukce automatu A , který přijímá **sjednocení** jazyků přijímaných automaty A_1 a A_2 , tj. jazyk

$$L(A_1) \cup L(A_2)$$

je téměř stejná jako v případě automatu přijímajícího $L(A_1) \cap L(A_2)$.

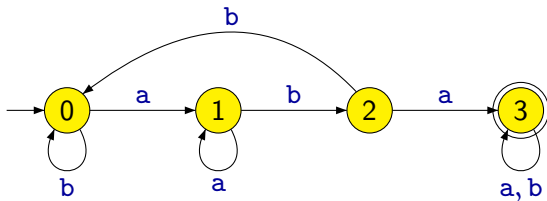
Jediný rozdíl je v definici množiny přijímajících stavů:

- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

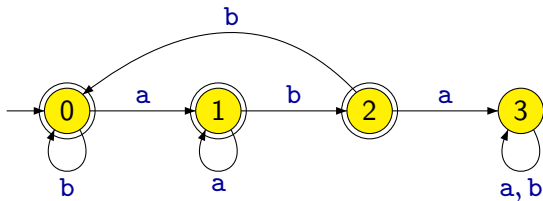
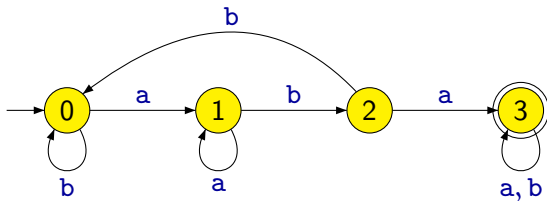
Věta

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



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.

$$L(A') = \overline{L(A)}$$

K DKA $A = (Q, \Sigma, \delta, q_0, F)$ sestrojíme DKA $A' = (Q, \Sigma, \delta, q_0, Q - F)$.

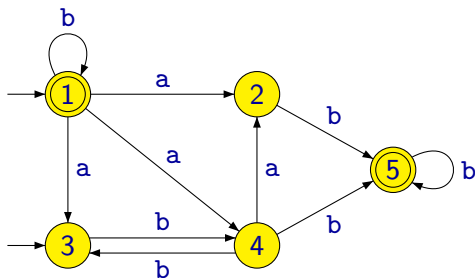
Je očividné, že pro každé slovo $w \in \Sigma^*$ platí, že $w \in L(A')$ právě tehdy, když $w \notin L(A)$, tj.

$$L(A') = \overline{L(A)}$$

Věta

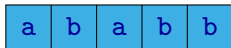
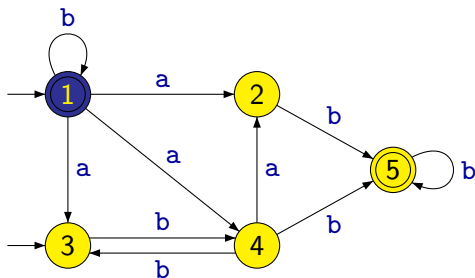
Jestliže jazyk L je regulární, pak také jeho doplněk $\overline{L}$ je regulární.

Nedeterministický konečný automat



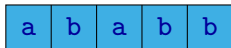
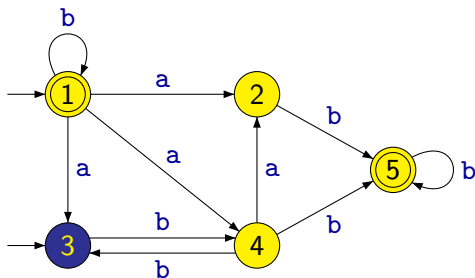
- Z jednoho stavu může vézt libovolný (i nulový) počet přechodů označených stejným symbolem.
- V automatu může být víc než jeden počáteční stav.

Nedeterministický konečný automat



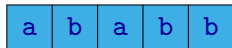
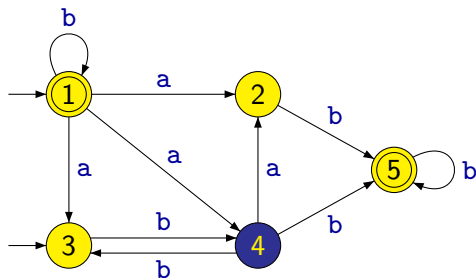
1

Nedeterministický konečný automat



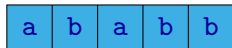
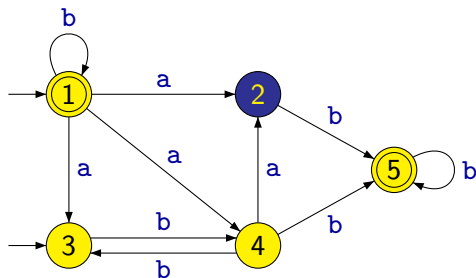
$$1 \xrightarrow{a} 3$$

Nedeterministický konečný automat



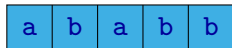
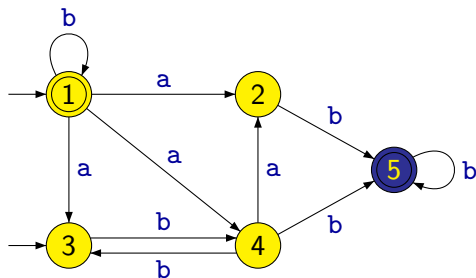
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

Nedeterministický konečný automat



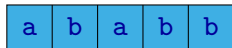
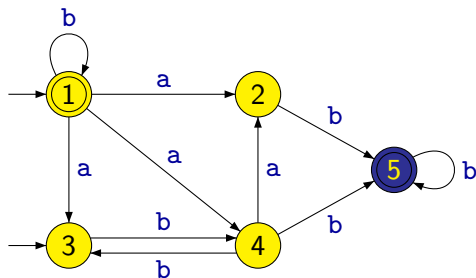
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2$$

Nedeterministický konečný automat



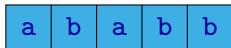
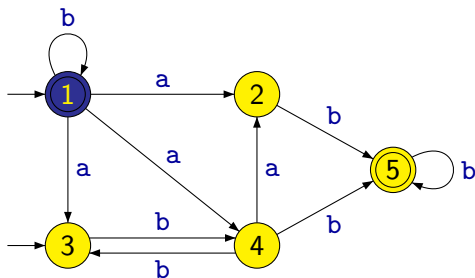
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5$$

Nedeterministický konečný automat



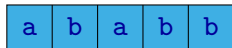
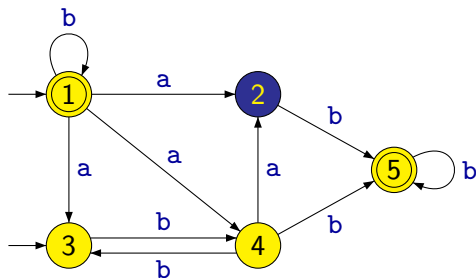
$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$

Nedeterministický konečný automat



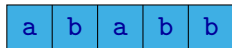
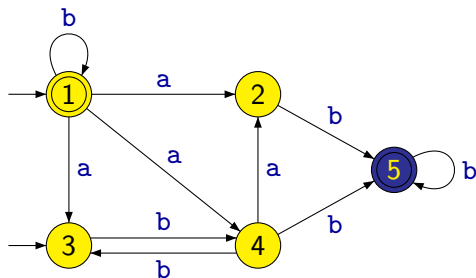
1

Nedeterministický konečný automat



$$1 \xrightarrow{a} 2$$

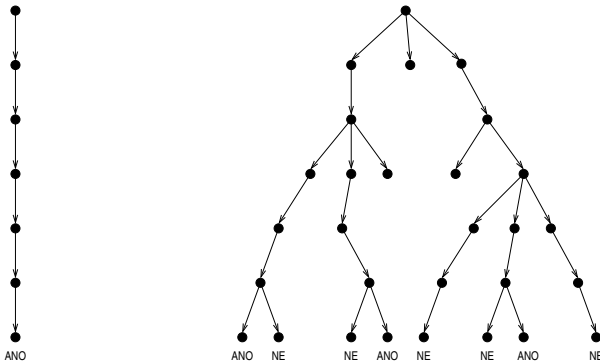
Nedeterministický konečný automat



$$1 \xrightarrow{a} 2 \xrightarrow{b} 5$$

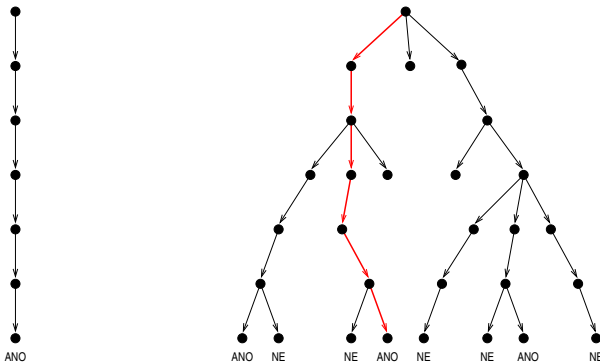
Nedeterministický konečný automat

Nedeterministický konečný automat přijímá dané slovo, jestliže **existuje** alespoň jeden jeho výpočet, který vede k přijetí tohoto slova.



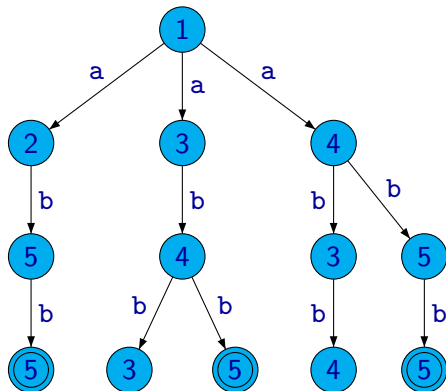
Nedeterministický konečný automat

Nedeterministický konečný automat přijímá dané slovo, jestliže **existuje** alespoň jeden jeho výpočet, který vede k přijetí tohoto slova.



Nedeterministický konečný automat

	a	b
↔ 1	2, 3, 4	1
2	—	5
→ 3	—	4
4	2	3, 5
← 5	—	5



Příklad: Les reprezentující všechny možné výpočty nad slovem **abb**.

Nedeterministický konečný automat

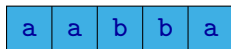
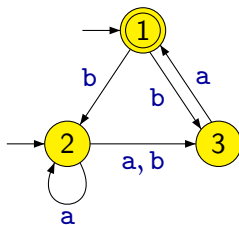
Formálně je **nedeterministický konečný automat (NKA)** definován jako pětice

$$(Q, \Sigma, \delta, I, F)$$

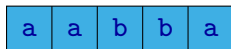
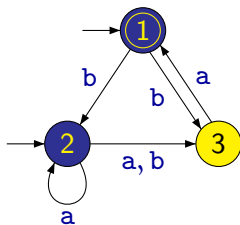
kde:

- Q je konečná množina **stavů**
- Σ je konečná **abeceda**
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ je **přechodová funkce**
- $I \subseteq Q$ je množina **počátečních stavů**
- $F \subseteq Q$ je množina **přijímajících stavů**

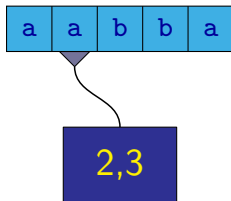
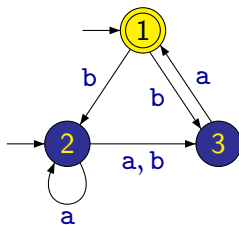
Převod NKA na DKA



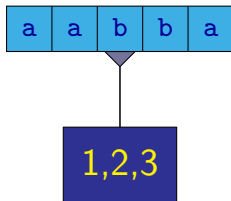
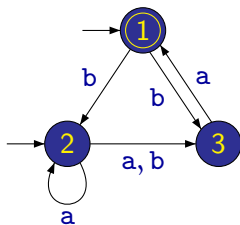
Převod NKA na DKA



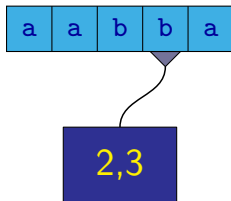
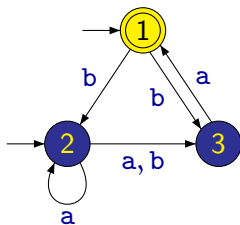
Převod NKA na DKA



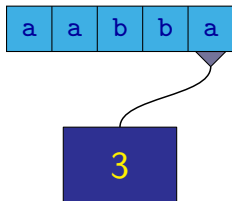
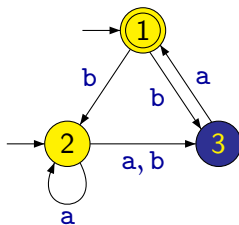
Převod NKA na DKA



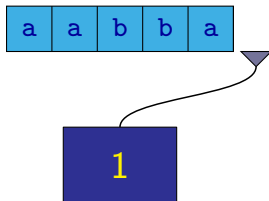
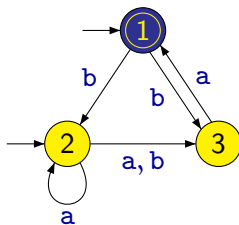
Převod NKA na DKA



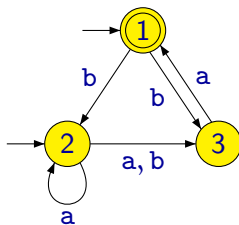
Převod NKA na DKA



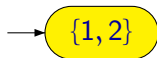
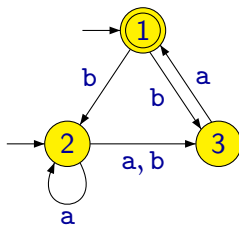
Převod NKA na DKA



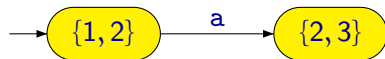
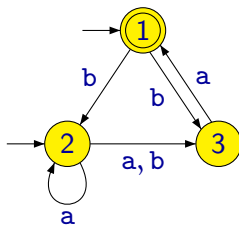
Převod NKA na DKA



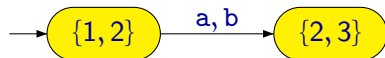
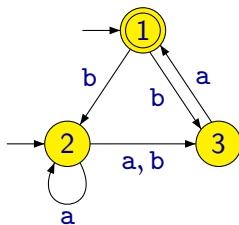
Převod NKA na DKA



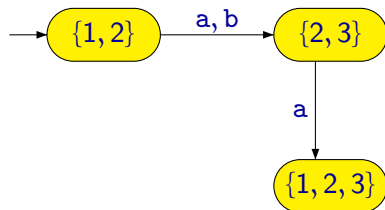
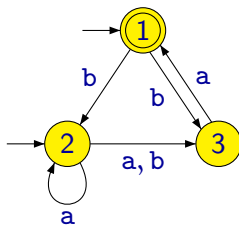
Převod NKA na DKA



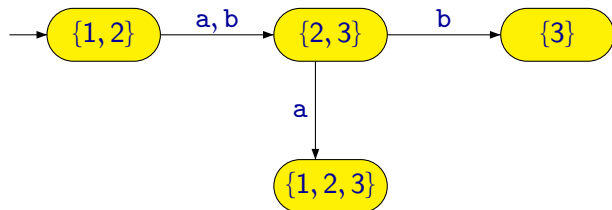
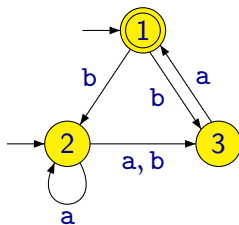
Převod NKA na DKA



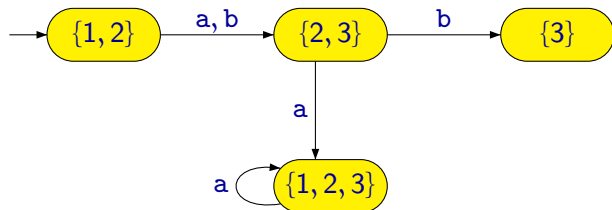
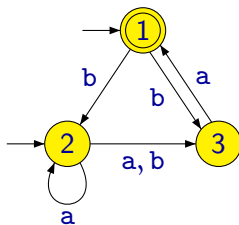
Převod NKA na DKA



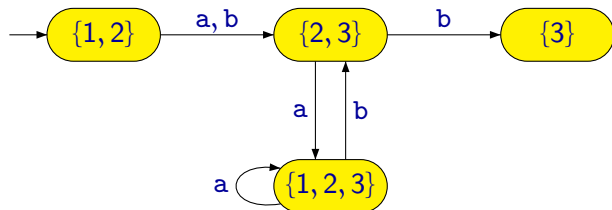
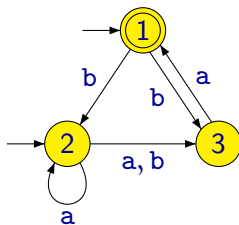
Převod NKA na DKA



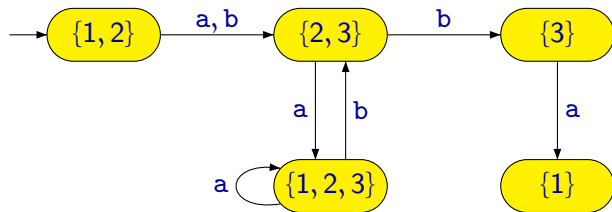
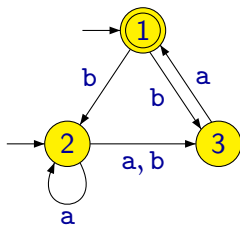
Převod NKA na DKA



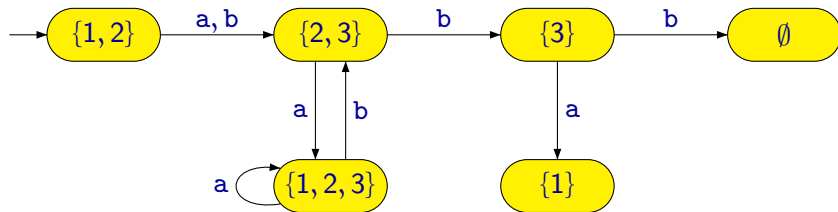
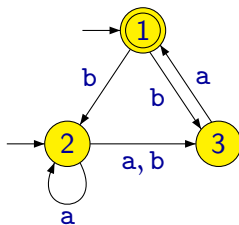
Převod NKA na DKA



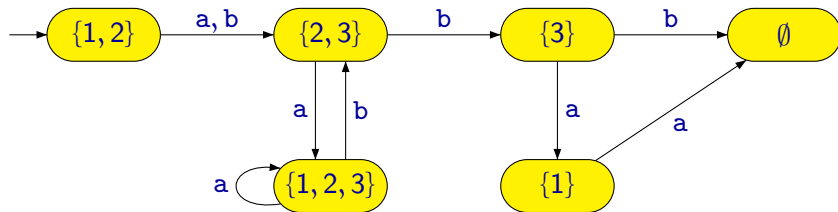
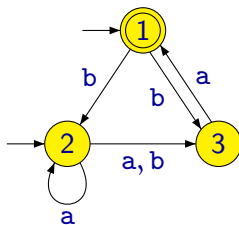
Převod NKA na DKA



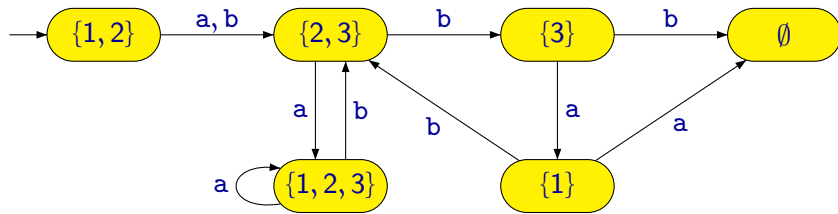
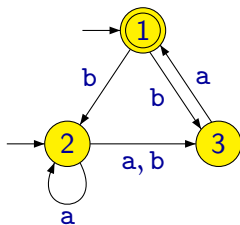
Převod NKA na DKA



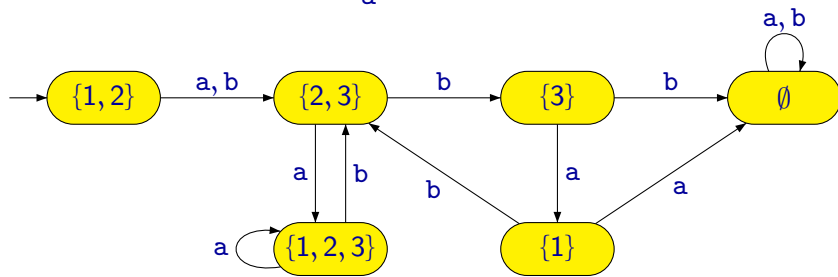
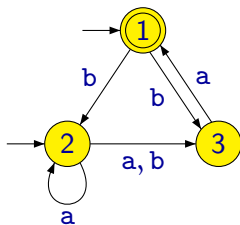
Převod NKA na DKA



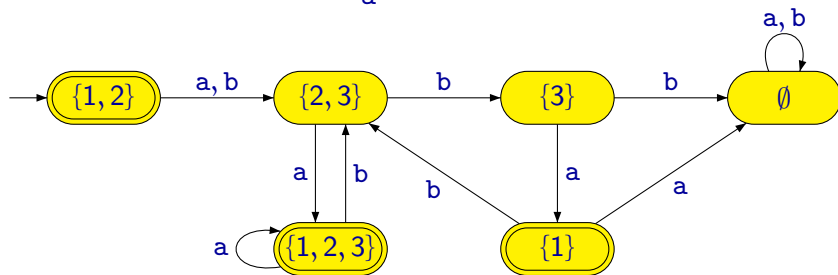
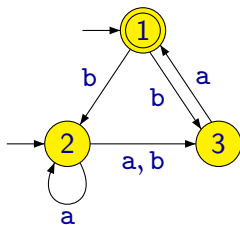
Převod NKA na DKA



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
$\leftrightarrow \{1, 2\}$	$\{2, 3\}$	$\{2, 3\}$
$\{2, 3\}$	$\{1, 2, 3\}$	$\{3\}$
$\leftarrow \{1, 2, 3\}$	$\{1, 2, 3\}$	$\{2, 3\}$
$\{3\}$	$\{1\}$	$\emptyset$
$\leftarrow \{1\}$	$\emptyset$	$\{2, 3\}$
$\emptyset$	$\emptyset$	$\emptyset$

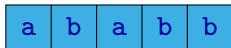
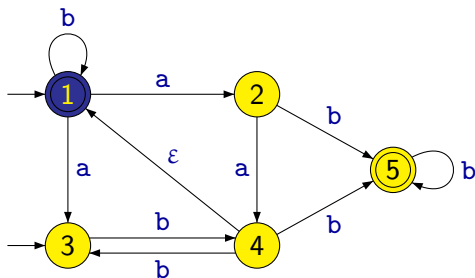
	a	b
$\leftrightarrow 1$	2	2
2	3	4
$\leftarrow 3$	3	2
4	5	6
$\leftarrow 5$	6	2
6	6	6

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.

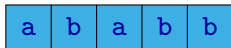
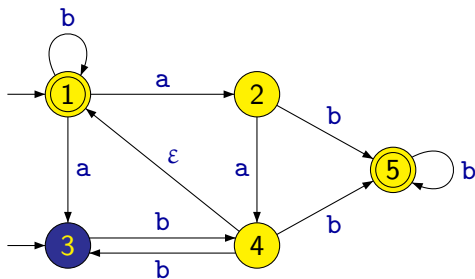
Zobecněný nedeterministický konečný automat



1

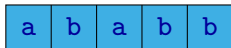
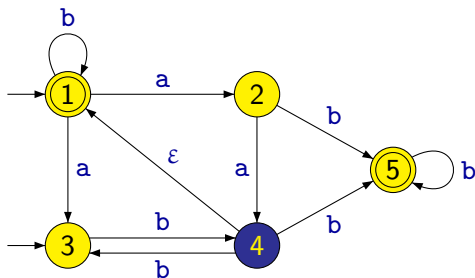


Zobecněný nedeterministický konečný automat



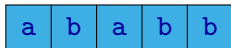
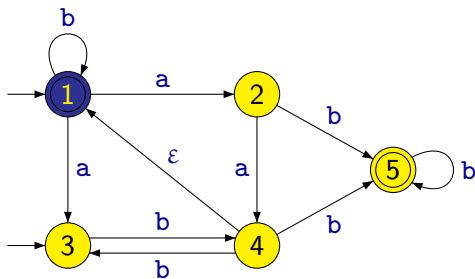
$1 \xrightarrow{a} 3$

Zobecněný nedeterministický konečný automat



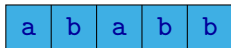
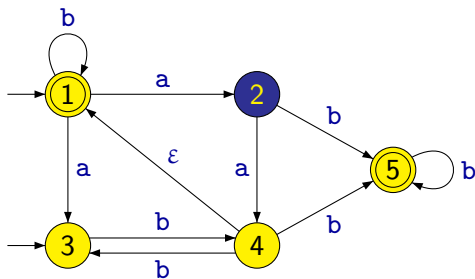
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4$$

Zobecněný nedeterministický konečný automat



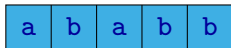
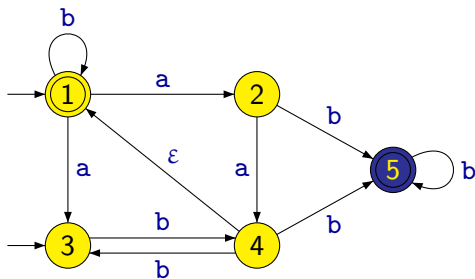
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1$$

Zobecněný nedeterministický konečný automat



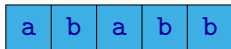
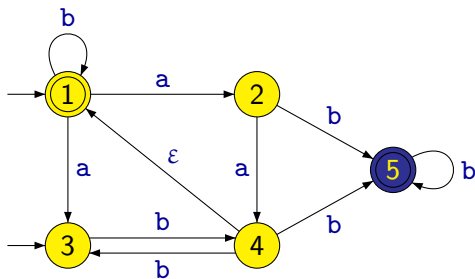
$$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2$$

Zobecněný nedeterministický konečný automat



$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\epsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5$

Zobecněný nedeterministický konečný automat



$1 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{\epsilon} 1 \xrightarrow{a} 2 \xrightarrow{b} 5 \xrightarrow{b} 5$

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 řídicí 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

$$(Q, \Sigma, \delta, I, F)$$

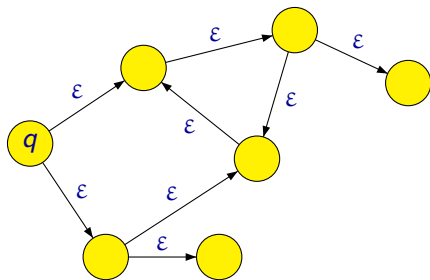
kde:

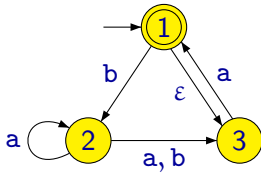
- Q je konečná množina **stavů**
- Σ je konečná **abeceda**
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ je **přechodová funkce**
- $I \subseteq Q$ je množina **počátečních stavů**
- $F \subseteq Q$ je množina **přijímajících stavů**

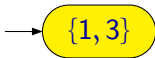
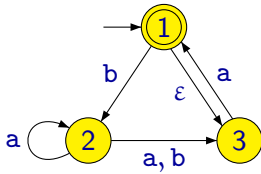
Poznámka: Na NKA můžeme nahlížet jako na speciální případ ZNKA, kde $\delta(q, \varepsilon) = \emptyset$ pro všechna $q \in Q$.

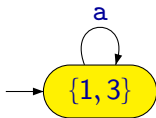
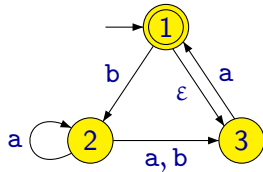
Převod na deterministický konečný automat

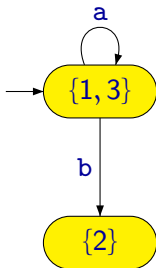
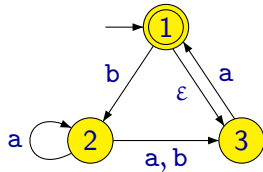
Zobecněný nedeterministický konečný automat je možné převést na deterministický podobnou konstrukcí jako nedeterministický konečný automat, s tím rozdílem, že do množin stavů musíme vždy přidat navíc i všechny stavy dosažitelné z již přidanych stavů nějakou sekvencí ϵ -přechodů.

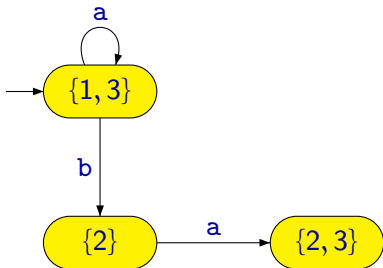
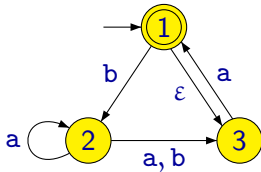


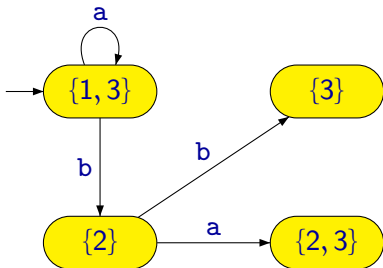
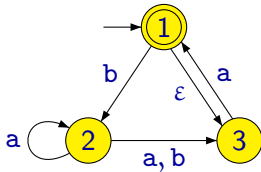


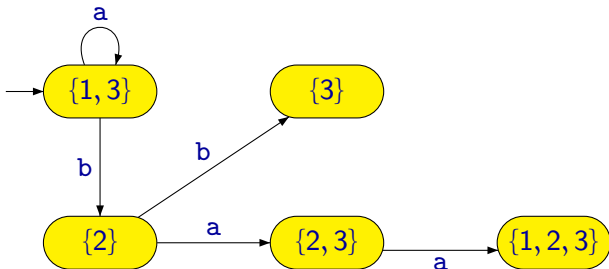
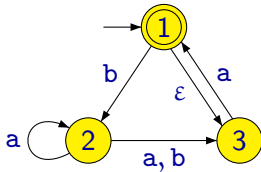


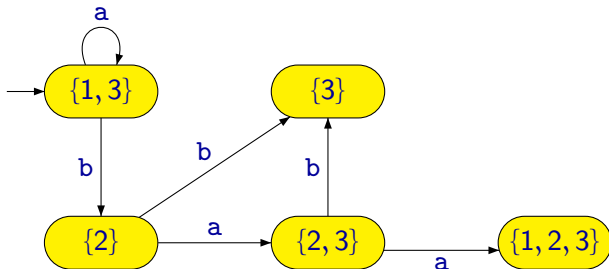
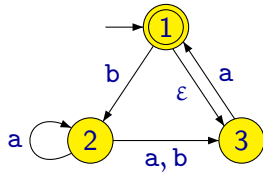


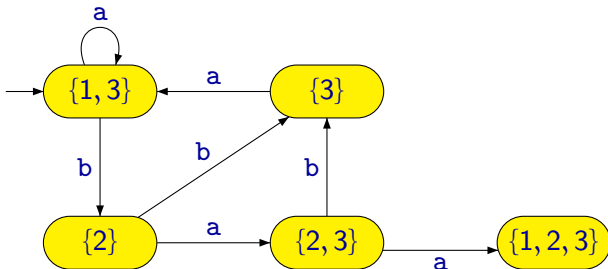
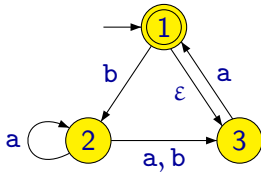


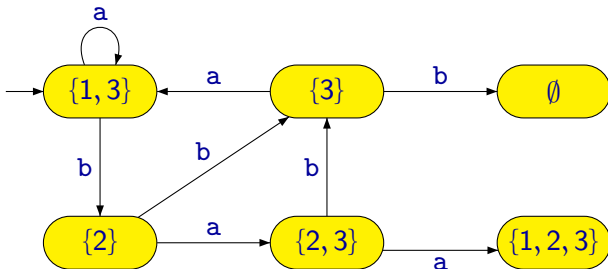
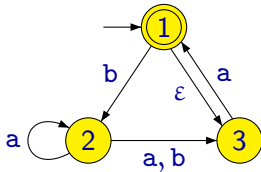


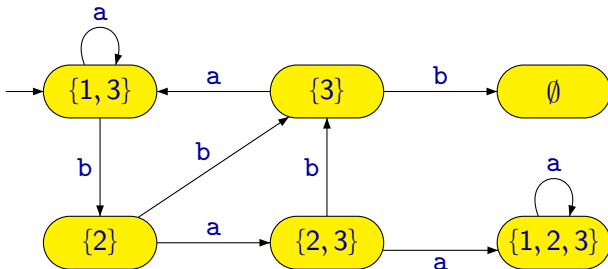
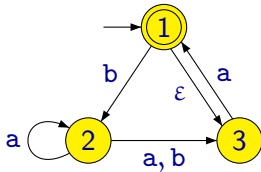


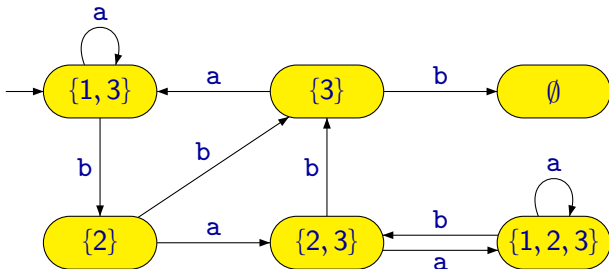
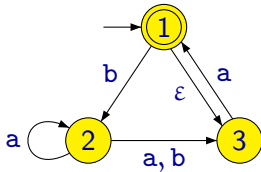


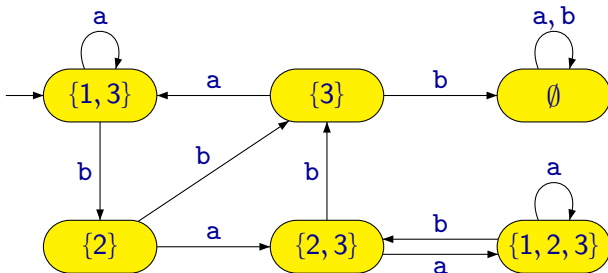
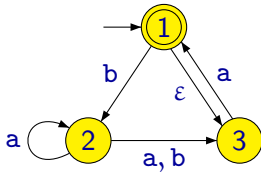


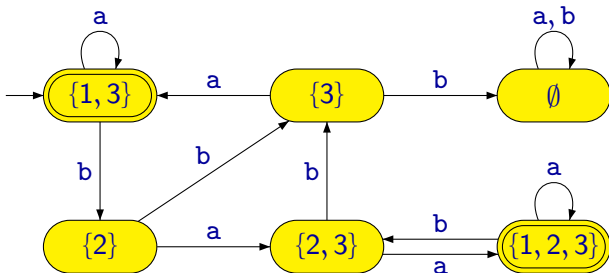
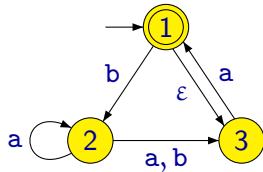












Předtím, než formálně popíšeme převod ZNKA na DKA, zavedme si několik pomocných definic.

Předpokládejme nějaký daný ZNKA $A = (Q, \Sigma, \delta, I, F)$.

Definujme funkci $\hat{\delta} : \mathcal{P}(Q) \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$ tak, že pro $K \subseteq Q$ a $a \in \Sigma \cup \{\varepsilon\}$ je

$$\hat{\delta}(K, a) = \bigcup_{q \in K} \delta(q, a)$$

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 $Cl_\varepsilon : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$ je definována tak, že pro $K \subseteq Q$ je $Cl_\varepsilon(K)$ nejmenší (vzhledem 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 $K \subseteq Q$)
- $\delta' : Q' \times \Sigma \rightarrow Q'$ je definová tak, že pro $K \in Q'$ a $a \in \Sigma$ je

$$\delta'(K, a) = Cl_\varepsilon(\hat{\delta}(Cl_\varepsilon(K), a))$$

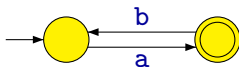
- $q'_0 = Cl_\varepsilon(I)$
- $F' = \{K \in Q' \mid Cl_\varepsilon(K) \cap F \neq \emptyset\}$

Není těžké ověřit, že $L(A) = L(A')$.

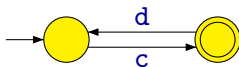
Zřetězení jazyků

$$\Sigma = \{a, b, c, d\}$$

A_1 :



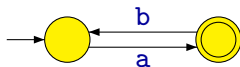
A_2 :



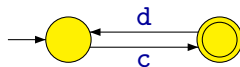
Zřetězení jazyků

$$\Sigma = \{a, b, c, d\}$$

A_1 :



A_2 :



A :

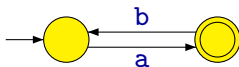


$$L(A) = L(A_1) \cdot L(A_2)$$

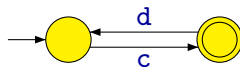
Zřetězení jazyků

$$\Sigma = \{a, b, c, d\}$$

A_1 :

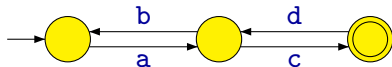


A_2 :

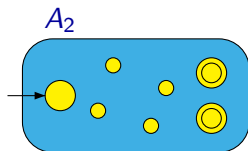
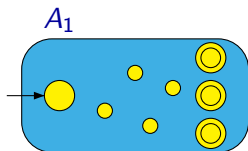


Chybná konstrukce:

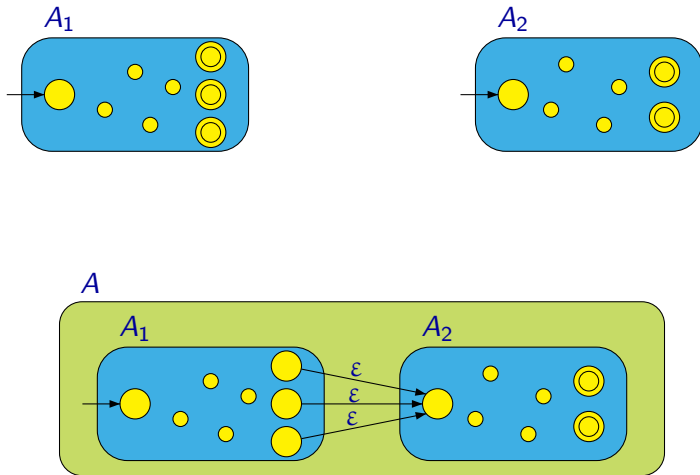
A :

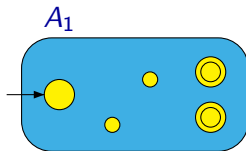


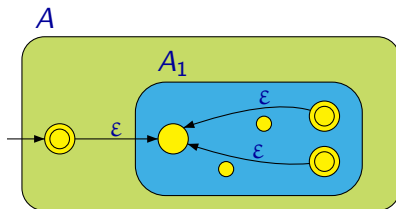
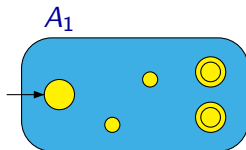
$acdbac \in L(A)$, ale $acdbac \notin L(A_1) \cdot L(A_2)$



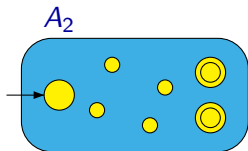
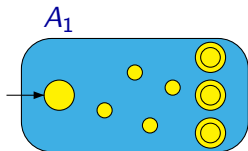
Zřetězení jazyků



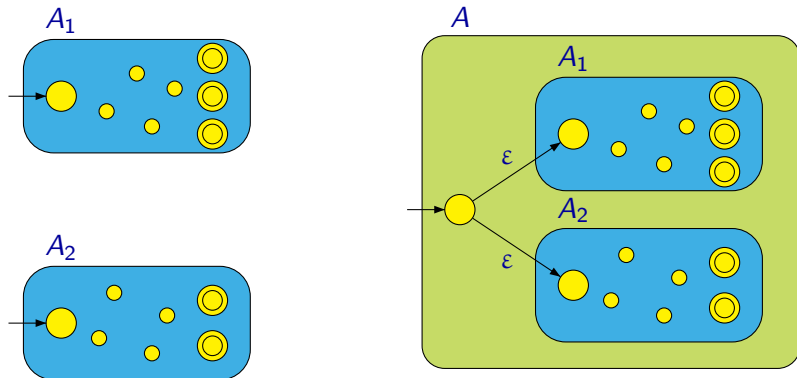




Alternativní konstrukce pro sjednocení jazyků:



Alternativní konstrukce pro sjednocení jazyků:



Množina (všech) regulárních jazyků je uzavřená vůči operacím:

- sjednocení
- průnik
- doplněk
- zřetězení
- iterace
- ...

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:



$\emptyset$

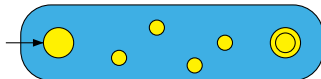


ϵ



a

Konstrukce pro sjednocení:



Převod regulárního výrazu na konečný automat

Automaty pro elementární výrazy:



$\emptyset$

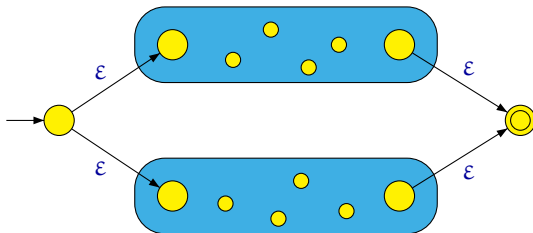


ϵ



a

Konstrukce pro sjednocení:



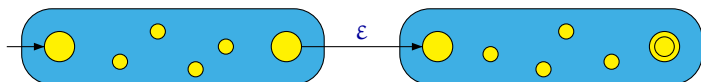
Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:



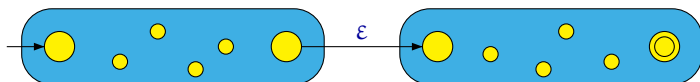
Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:

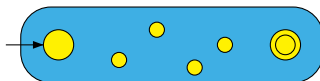


Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:

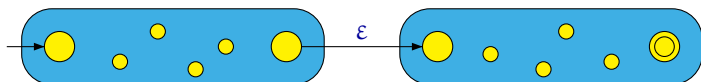


Konstrukce pro iteraci:

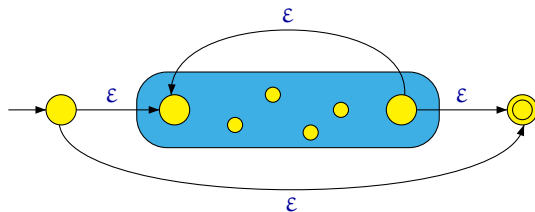


Převod regulárního výrazu na konečný automat

Konstrukce pro zřetězení:



Konstrukce pro iteraci:



Příklad: Konstrukce automatu pro výraz $((0 + 1) \cdot 1)^*$:

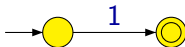
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0 + 1) \cdot 1)^*$:



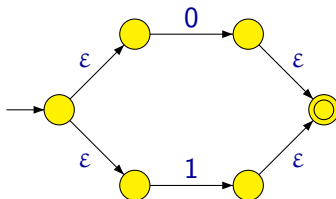
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0 + 1) \cdot 1)^*$:



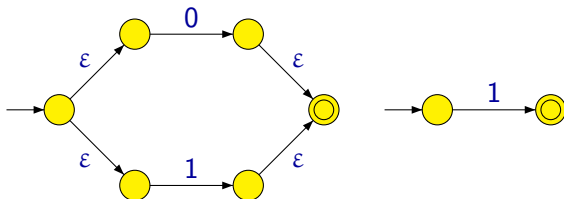
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0 + 1) \cdot 1)^*$:



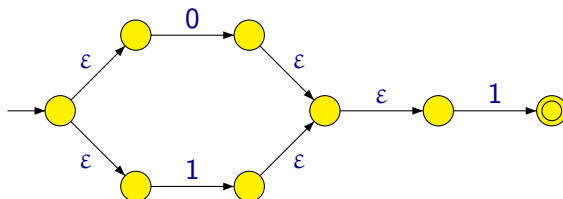
Převod regulárního výrazu na konečný automat

Příklad: Konstrukce automatu pro výraz $((0 + 1) \cdot 1)^*$:



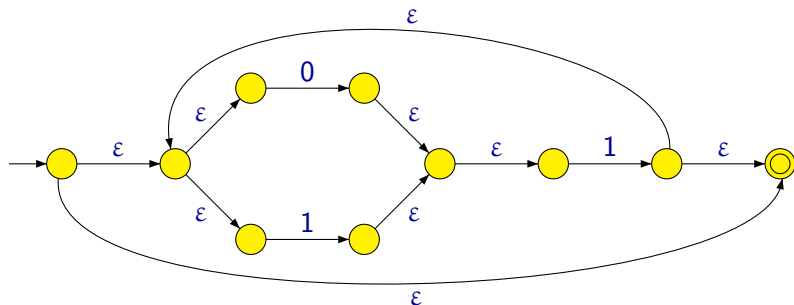
Převod regulárního výrazu na konečný automat

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.

Tvrzení

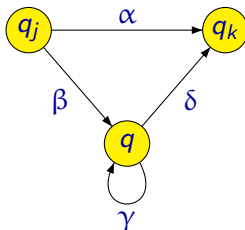
Každý regulární jazyk je možné popsat nějakým regulárním výrazem.

Důkaz: Stačí ukázat, jak pro libovolný konečný automat A zkonstruovat regulární výraz α takový, že $[\alpha] = L(A)$.

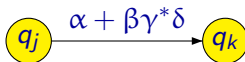
- A upravíme tak, aby měl právě jeden počáteční a právě jeden koncový stav.
- Budeme postupně odebírat jednotlivé stavy.
- Přejchody budou označeny regulárními výrazy.
- Zbude automat se dvěma stavy – počátečním a koncovým, a jedním přechodem ohodnoceným výsledným regulárním výrazem.

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 .

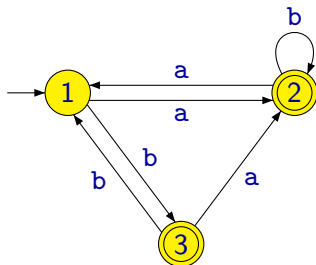


Po odstranění stavu q :



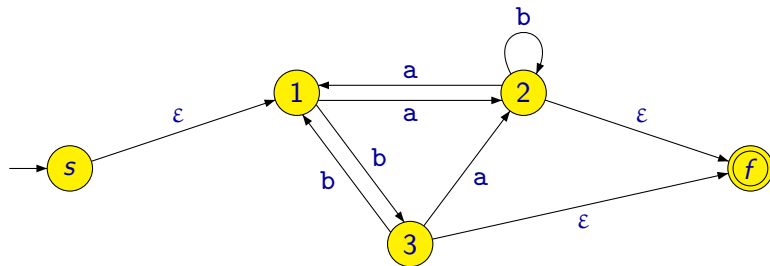
Převod konečného automatu na regulární výraz

Příklad:



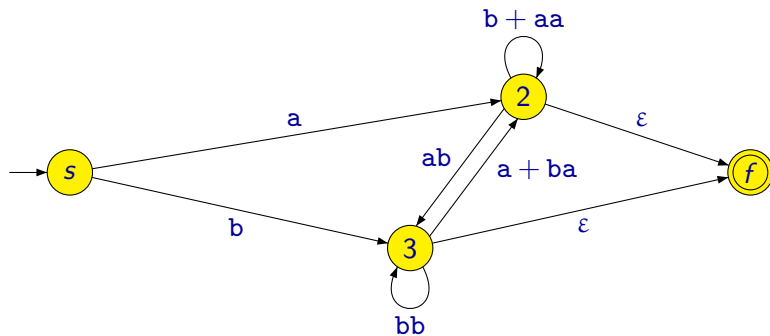
Převod konečného automatu na regulární výraz

Příklad:



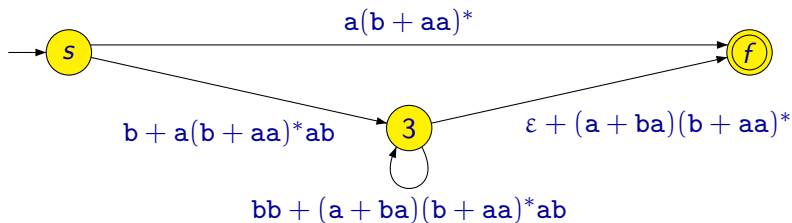
Převod konečného automatu na regulární výraz

Příklad:



Převod konečného automatu na regulární výraz

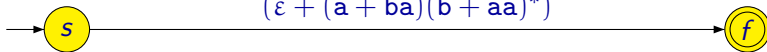
Příklad:



Převod konečného automatu na regulární výraz

Příklad:

$$\begin{aligned} & a(b + aa)^* + \\ & (b + a(b + aa)^* ab) \\ & (bb + (a + ba)(b + aa)^* ab)^* \\ & (\varepsilon + (a + ba)(b + aa)^*) \end{aligned}$$



Věta

Jazyk je regulární právě tehdy, když je ho možné popsat regulárním výrazem.

Bezkontextové gramatiky

Příklad: Chtěli bychom popsat jazyk aritmetických výrazů obsahující výrazy jako například:

$$175 \quad (9+15) \quad (((10-4)*((1+34)+2))/(3+(-37)))$$

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: $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)\}$

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.

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

$$D \rightarrow 0$$

$$D \rightarrow 1$$

$$D \rightarrow 2$$

$$D \rightarrow 3$$

$$D \rightarrow 4$$

$$D \rightarrow 5$$

$$D \rightarrow 6$$

$$D \rightarrow 7$$

$$D \rightarrow 8$$

$$D \rightarrow 9$$

$$C \rightarrow D$$

$$C \rightarrow DC$$

$$E \rightarrow C$$

$$E \rightarrow (-E)$$

$$E \rightarrow (E+E)$$

$$E \rightarrow (E-E)$$

$$E \rightarrow (E * E)$$

$$E \rightarrow (E / E)$$

Příklad (pokr.): Stručnější způsob zápisu:

$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

$$C \rightarrow D \mid DC$$

$$E \rightarrow C \mid (-E) \mid (E+E) \mid (E-E) \mid (E * E) \mid (E/E)$$

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 “,”):

$$S \rightarrow T \mid \varepsilon$$

$$T \rightarrow E \mid E, T$$

$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

$$C \rightarrow D \mid DC$$

$$E \rightarrow C \mid (-E) \mid (E+E) \mid (E-E) \mid (E * E) \mid (E/E)$$

Příklad: Příkazy nějakého programovacího jazyka (fragment gramatiky):

$$\begin{aligned} S &\rightarrow E; \mid T \mid \text{if } (E) S \mid \text{if } (E) S \text{ else } S \\ &\quad \mid \text{while } (E) S \mid \text{do } S \text{ while } (E); \mid \text{for } (F; F; F) S \\ &\quad \mid \text{return } F; \\ T &\rightarrow \{ U \} \\ U &\rightarrow \varepsilon \mid SU \\ F &\rightarrow \varepsilon \mid E \\ E &\rightarrow \dots \\ &\quad \dots \end{aligned}$$

Poznámka:

- S — příkaz
- T — blok příkazů
- U — sekvence příkazů
- E — výraz
- F — výraz, který je možno vynechat

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**

Poznámky:

- Pro označení neterminálních symbolů budeme používat velká písmena $A, B, C, \dots$
- Pro označení terminálních symbolů budeme používat malá písmena $a, b, c, \dots$ nebo číslice $0, 1, 2, \dots$
- Pro označení řetězců z $(\Pi \cup \Sigma)^*$ budeme používat malá písmena řecké abecedy $\alpha, \beta, \gamma, \dots$
- Místo zápisu (A, α) budeme pro pravidla používat zápis

$$A \rightarrow \alpha$$

A – levá strana pravidla

α – pravá strana pravidla

Příklad: Gramatika $G = (\Pi, \Sigma, S, P)$, kde

- $\Pi = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $S = A$
- P obsahuje pravidla

$$A \rightarrow aBBb$$

$$A \rightarrow AaA$$

$$B \rightarrow \varepsilon$$

$$B \rightarrow bCA$$

$$C \rightarrow AB$$

$$C \rightarrow a$$

$$C \rightarrow b$$

Poznámka: Pokud máme více pravidel se stejnou levou stranou, jako třeba

$$A \rightarrow \alpha_1$$

$$A \rightarrow \alpha_2$$

$$A \rightarrow \alpha_3$$

můžeme je stručněji zapsat jako

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$$

Například pravidla dříve uvedené gramatiky můžeme zapsat jako

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

A

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

A

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \rightarrow \underline{aBBb} \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $\underline{a}bbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$\underline{A} \Rightarrow \underline{aBBb}$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow a\underline{B}Bb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \varepsilon \mid \underline{bCA}$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow a\underline{B}Bb \Rightarrow a\underline{bCA}Bb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$\underline{A} \rightarrow \underline{aBBb} \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo $abbabb$ je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \underline{\varepsilon} \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$\underline{C} \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$\underline{C} \rightarrow AB \mid a \mid \underline{b}$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \underline{\varepsilon} \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

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

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb$$

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

$$A \rightarrow aBBb \mid AaA$$

$$\underline{B} \rightarrow \underline{\varepsilon} \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb \Rightarrow abbabb$$

Gramatiky slouží ke generování slov.

Příklad: $G = (\Pi, \Sigma, A, P)$, kde $\Pi = \{A, B, C\}$, $\Sigma = \{a, b\}$ a P obsahuje pravidla

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

Například slovo *abbabb* je možné v gramatice G vygenerovat následujícím způsobem:

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb \Rightarrow abbabb$$

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 $(A \rightarrow \gamma) \in P$.

Příklad: Jestliže $(B \rightarrow bCA) \in P$, 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 $A \rightarrow \gamma$, kde se A vyskytuje na levé straně.

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 $(A \rightarrow \gamma) \in P$.

Příklad: Jestliže $(B \rightarrow bCA) \in P$, 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 $A \rightarrow \gamma$, kde se A vyskytuje na levé straně.

Derivace délky n je posloupnost $\beta_0, \beta_1, \beta_2, \dots, \beta_n$, 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

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \dots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$$

Skutečnost, že pro dané $\alpha, \alpha' \in (\Pi \cup \Sigma)^*$ a $n \in \mathbb{N}$ existuje nějaká derivace $\beta_0 \Rightarrow \beta_1 \Rightarrow \beta_2 \Rightarrow \dots \Rightarrow \beta_{n-1} \Rightarrow \beta_n$, 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 $\Rightarrow$).

Větné formy jsou ty $\alpha \in (\Pi \cup \Sigma)^*$, pro které platí

$$S \Rightarrow^* \alpha$$

kde S je počáteční neterminál.

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.

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

Příklad: Chceme vytvořit gramatiku generující jazyk

$$L = \{a^n b^n \mid n \geq 0\}$$

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

$$S \rightarrow aSb \mid \varepsilon$$

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

$$S \rightarrow aSb \mid \varepsilon$$

$$S \Rightarrow \varepsilon$$

$$S \Rightarrow aSb \Rightarrow ab$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

...

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.

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:

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

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:

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$$

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abaSaba \Rightarrow abaaaba$$

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.
Například $((()())()) \in L$, ale $()() \notin L$.

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.
Například $((()())()) \in L$, ale $()() \notin L$.

Řešení:

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

Příklad: Chceme vytvořit gramatiku generující jazyk L tvořený všemi dobře uzávorkovanými sekvencemi symbolů '(' a ')'.
Například $((()())()) \in L$, ale $()()) \notin L$.

Řešení:

$$S \rightarrow \varepsilon \mid (S) \mid SS$$

$$\begin{aligned} S &\Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (SS)(S) \Rightarrow ((S)S)(S) \Rightarrow \\ &(()S)(S) \Rightarrow (() (S))(S) \Rightarrow (() ())(S) \Rightarrow (() ())((S)) \Rightarrow \\ &(() ())(()) \end{aligned}$$

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 $(a + a) * a + (a * a) \in L$.

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:

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$

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:

$$E \rightarrow a \mid E + E \mid E * E \mid (E)$$

$$\begin{aligned} E &\Rightarrow E + E \Rightarrow E * E + E \Rightarrow (E) * E + E \Rightarrow (E + E) * E + E \Rightarrow \\ &(a + E) * E + E \Rightarrow (a + a) * E + E \Rightarrow (a + a) * a + E \Rightarrow (a + a) * a + (E) \Rightarrow \\ &(a + a) * a + (E * E) \Rightarrow (a + a) * a + (a * E) \Rightarrow (a + a) * a + (a * a) \end{aligned}$$

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

A

$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

A

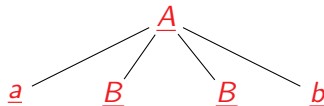
A

A $\rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

A

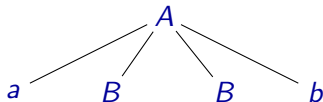


A $\rightarrow$ aBBb | AaA

B $\rightarrow$ ε | bCA

C $\rightarrow$ AB | a | b

A $\Rightarrow$ aBBb

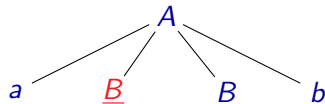


$$A \rightarrow aBBb \mid AaA$$

$$B \rightarrow \varepsilon \mid bCA$$

$$C \rightarrow AB \mid a \mid b$$

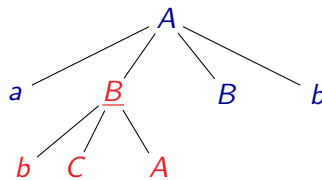
$$A \Rightarrow aBBb$$


$$A \rightarrow aBBb \mid AaA$$
$$\underline{B} \rightarrow \varepsilon \mid bCA$$
$$C \rightarrow AB \mid a \mid b$$
$$A \Rightarrow a\underline{B}Bb$$

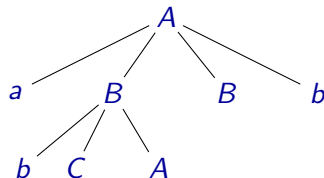
$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \varepsilon \mid \underline{bCA}$

$C \rightarrow AB \mid a \mid b$



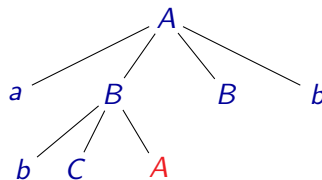
$A \Rightarrow a\underline{B}Bb \Rightarrow a\underline{bCA}Bb$

$$A \rightarrow aBBb \mid AaA$$
$$B \rightarrow \varepsilon \mid bCA$$
$$C \rightarrow AB \mid a \mid b$$

$$A \Rightarrow aBBb \Rightarrow abCABb$$

A $\rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

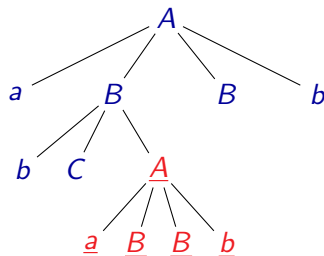


$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb$

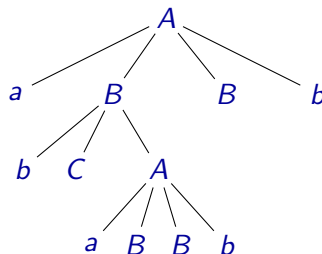
$\underline{A} \rightarrow \underline{aBBb} \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abC\underline{A}Bb \Rightarrow abC\underline{aBBb}Bb$

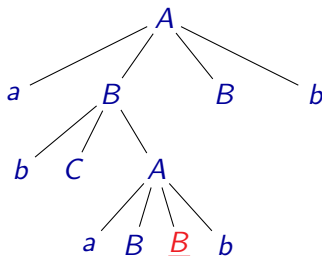
$$A \rightarrow aBBb \mid AaA$$
$$B \rightarrow \varepsilon \mid bCA$$
$$C \rightarrow AB \mid a \mid b$$

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb$$

Derivační strom

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

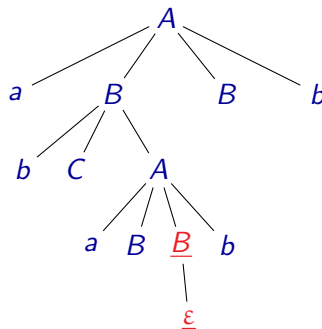


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb$

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \underline{\epsilon} \mid bCA$

$C \rightarrow AB \mid a \mid b$

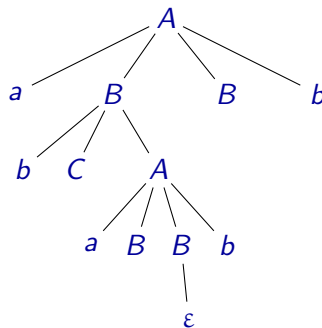


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaB\underline{B}bBb \Rightarrow abCaBbBb$

$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

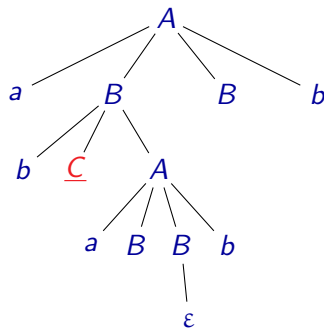


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb$

$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$\underline{C} \rightarrow AB \mid a \mid b$

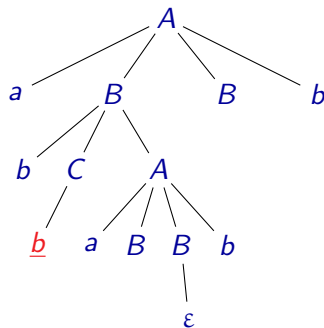


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb$

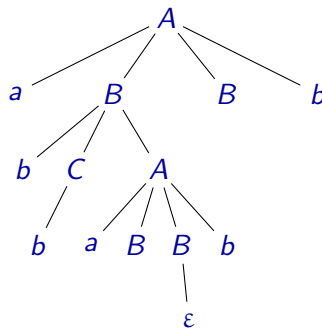
$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$\underline{C} \rightarrow AB \mid a \mid \underline{b}$



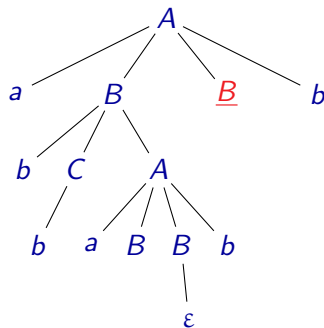
$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow ab\underline{C}aBbBb \Rightarrow ab\underline{b}aBbBb$

$$A \rightarrow aBBb \mid AaA$$
$$B \rightarrow \varepsilon \mid bCA$$
$$C \rightarrow AB \mid a \mid b$$

$$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb$$

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$

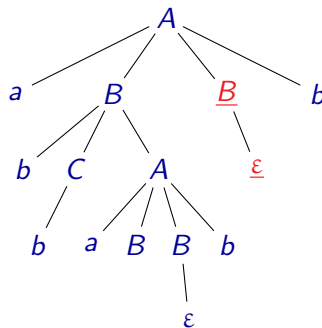


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b$

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \underline{\epsilon} \mid bCA$

$C \rightarrow AB \mid a \mid b$

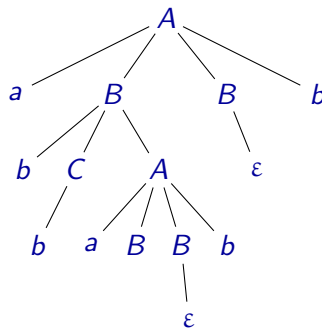


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBb\underline{B}b \Rightarrow abbaBbb$

$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$



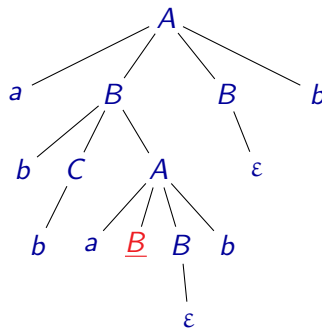
$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abbaBbb$

Derivační strom

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$



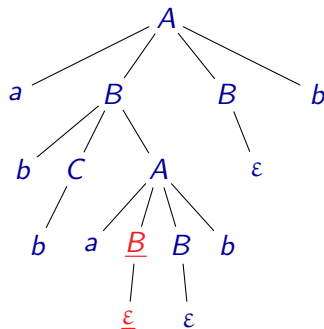
$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow abba\underline{B}bb$

Derivační strom

$A \rightarrow aBBb \mid AaA$

$\underline{B} \rightarrow \underline{\epsilon} \mid bCA$

$C \rightarrow AB \mid a \mid b$

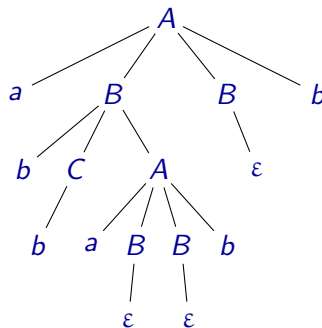


$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow$
 $abba\underline{B}bb \Rightarrow abbabb$

$A \rightarrow aBBb \mid AaA$

$B \rightarrow \varepsilon \mid bCA$

$C \rightarrow AB \mid a \mid b$



$A \Rightarrow aBBb \Rightarrow abCABb \Rightarrow abCaBBbBb \Rightarrow abCaBbBb \Rightarrow abbaBbBb \Rightarrow$
 $abbaBbb \Rightarrow abbabb$

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 $A \rightarrow \alpha$.

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

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:

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow E * \underline{E} + E \Rightarrow E * a + \underline{E} \Rightarrow \underline{E} * a + a \Rightarrow a * a + a$$

- Jednomu derivačnímu stromu může odpovídat více různých derivací.
- Každému derivačnímu stromu odpovídá právě jedna levá a právě jedna pravá derivace.

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^* .

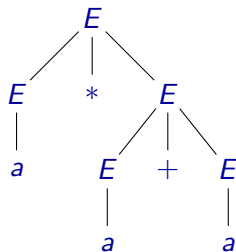
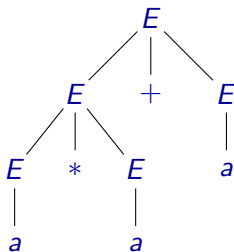
Nejednoznačné gramatiky

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

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

můžeme nahradit ekvivalentní gramatikou

$$E \rightarrow T \mid T + E$$

$$T \rightarrow F \mid F * T$$

$$F \rightarrow a \mid (E)$$

Poznámka: Pokud se nejednoznačná gramatika žádnou ekvivalentní jednoznačnou gramatikou nahradit nedá, říkáme, že je **podstatně nejednoznačná**.

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

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 $L(G) = L(G_1)L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\})$$

- Gramatika G taková, že $L(G) = L(G_1) \cup L(G_2)$:

$$G = (\Pi_1 \cup \Pi_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\})$$

- Gramatika G taková, že $L(G) = L(G_1)^*$:

$$G = (\Pi_1 \cup \{S\}, \Sigma, S, P_1 \cup \{S \rightarrow \varepsilon, S \rightarrow S_1 S\})$$

Příklad: Chtěli bychom rozpoznávat jazyk aritmetických výrazů obsahující výrazy jako například:

34

$x+1$

$-x * 2 + 128 * (y - z / 3)$

- Ve výrazech se mohou vyskytovat číselné konstanty — posloupnosti číslic 0, 1, . . . , 9.
- Ve výrazech se mohou vyskytovat jména proměnných — posloupnosti tvořené písmeny, číslicemi a znakem “_”, které nezačínají číslicí.
- Výrazy mohou obsahovat základní aritmetické operace — “+”, “-”, “*”, “/” a unární “-”.
- Je možné používat závorky — “(” a “)” a běžnou prioritu aritmetických operací.

- **Vstup:** posloupnost znaků (např. řetězec, textový soubor, apod.)
- **Výstup:** abstraktní syntaktický strom reprezentující strukturu daného výrazu nebo informace o syntaktické chybě ve výrazu

Vytváření **abstraktního syntaktického stromu**:

- Výčtový typ reprezentující binární aritmetické operace:

enum *Bin_op* { **Add**, **Sub**, **Mul**, **Div** }

- Výčtový typ reprezentující unární aritmetické operace:

enum *Un_op* { **Un_minus** }

- Funkce pro vytváření různých typů vrcholů abstraktního syntaktického stromu:

- **MK-VAR**(*ident*) — vytvoření listu reprezentujícího proměnnou
- **MK-NUM**(*num*) — vytvoření listu reprezentujícího číselnou konstantu
- **MK-UNARY**(*op*, *e*) — vytvoření vrcholu s jedním potomkem *e*, na který je aplikována unární operace *op* (typu *Un_op*)
- **MK-BINARY**(*op*, *e1*, *e2*) — vytvoření vrcholu se dvěma potomky *e1* a *e2*, na které je aplikována binární operace *op* (typu *Bin_op*)

Výčtový typ *Token_kind* reprezentující různé druhy **tokenů**:

T_EOF	— konec vstupu
T_Ident	— identifikátor
T_Number	— číselná konstanta
T_LParen	— “(”
T_RParen	— “)”
T_Plus	— “+”
T_Minus	— “-”
T_Star	— “*”
T_Slash	— “/”

Proměnná `c` : naposledy načtený znak (resp. speciální hodnota `<eof>` reprezentující, že byl dosažen konec vstupu):

- na začátku se do proměnné `c` načte první znak ze vstupu
- funkce `NEXT-CHAR()` vrací následující znak ze vstupu

Některé pomocné funkce:

- `ERROR()` — vypíše informaci o syntaktické chybě a ukončí načítání výrazu
- `is-ident-start-char(c)` — testuje, zda `c` je znak, kterým může začínat identifikátor
- `is-ident-normal-char(c)` — testuje, zda `c` je znak, který se může vyskytovat v identifikátoru (jinde než na začátku)
- `is-digit(c)` — testuje, zda je `c` číslice

Některé další pomocné funkce:

- `CREATE-IDENT(s)` — vytváří identifikátor z daného řetězce *s*
- `CREATE-NUMBER(s)` — vytváří číslo z daného řetězce *s*

Pomocné proměnné:

- *last-ident* — naposledy načtený identifikátor
- *last-num* — naposledy načtená číselná konstanta

Funkce `NEXT-TOKEN()` — hlavní část lexikálního analyzátoru, vrací následující token ze vstupu

```
1 NEXT-TOKEN():
2 begin
3   while  $c \in \{ " ", "\t" \}$  do
4      $c := \text{NEXT-CHAR}()$ ;
5   end
6   if  $c == \langle \text{eof} \rangle$  then
7     return T_EOF
8   else
9     switch  $c$  do
10      case "(": do  $c := \text{NEXT-CHAR}()$ ; return T_LParen
11      case ")": do  $c := \text{NEXT-CHAR}()$ ; return T_RParen
12      case "+": do  $c := \text{NEXT-CHAR}()$ ; return T_Plus
13      case "-": do  $c := \text{NEXT-CHAR}()$ ; return T_Minus
14      case "*": do  $c := \text{NEXT-CHAR}()$ ; return T_Star
15      case "/": do  $c := \text{NEXT-CHAR}()$ ; return T_Slash
16      otherwise do
17        if is-ident-start-char( $c$ ) then
18          return SCAN-IDENT()
19        else if is-digit( $c$ ) then
20          return SCAN-NUMBER()
21        else ERROR()
22      end
23    end
24  end
25 end
```

```
1 SCAN-IDENT():  
2 begin  
3    $s := c$   
4    $c := \text{NEXT-CHAR}()$   
5   while is-ident-normal-char( $c$ ) do  
6      $s := s \cdot c$   
7      $c := \text{NEXT-CHAR}()$   
8   end  
9    $\text{last-ident} := \text{CREATE-IDENT}(s)$   
10  return T_Ident  
11 end
```

```
1 SCAN-NUMBER():  
2 begin  
3    $s := c$   
4    $c := \text{NEXT-CHAR}()$   
5   while is-digit( $c$ ) do  
6      $s := s \cdot c$   
7      $c := \text{NEXT-CHAR}()$   
8   end  
9    $\text{last-num} := \text{CREATE-NUMBER}(s)$   
10  return T_Number  
11 end
```

Proměnná t :

- naposledy načtený token

Pomocná funkce:

- `INIT-SCANNER()`:
 - inicialuje lexikální analyzátor
 - načte do proměnné c první znak ze vstupu

Bezkontextová gramatika pro daný jazyk:

$$S \rightarrow E \langle eof \rangle$$

$$E \rightarrow T G$$

$$G \rightarrow \varepsilon \mid A T G$$

$$A \rightarrow + \mid -$$

$$T \rightarrow F U$$

$$U \rightarrow \varepsilon \mid M F U$$

$$M \rightarrow * \mid /$$

$$F \rightarrow - F \mid (E) \mid \langle ident \rangle \mid \langle num \rangle$$

Jednou z často používaných metod syntaktické analýzy je tzv. **rekurzivní sestup**:

- Každému neterminálu odpovídá jedna funkce — funkce odpovídající neterminálu A implementuje všechna pravidla s neterminálem A na levé straně.
- Na základě následujícího tokenu se vybírá v rámci dané funkce mezi jednotlivými pravidly.
- Instrukce v těle funkce odpovídají zpracování pravých stran jednotlivých pravidel:
 - výskyt neterminálu B — zavolá se funkce odpovídající neterminálu B
 - výskyt terminálu a — zkontroluje se, že následující token odpovídá terminálu a , pokud odpovídá, načte se další token, pokud neodpovídá, ohlásí se chyba

$$S \rightarrow E \langle \text{eof} \rangle$$

```
1 PARSE():  
2 begin  
3   INIT-SCANNER()  
4   t := NEXT-TOKEN()  
5   e := PARSE-E()  
6   if t == T_EOF then return e  
7   else ERROR()  
8 end
```

$$E \rightarrow T G$$

```
1 PARSE-E ():  
2 begin  
3   e1 := PARSE-T()  
4   return PARSE-G(e1)  
5 end
```

$$G \rightarrow \varepsilon \mid A T G$$

```
1 PARSE-G (e1):  
2 begin  
3   if  $t \in \{\mathbf{T\_Plus}, \mathbf{T\_Minus}\}$  then  
4     op := PARSE-A()  
5     e2 := PARSE-T()  
6     return PARSE-G(MK-BINARY(op, e1, e2))  
7   else return e1  
8 end
```

$$T \rightarrow F U$$

```

1 PARSE-T ():
2 begin
3   e1 := PARSE-F()
4   return PARSE-U(e1)
5 end

```

$$U \rightarrow \varepsilon \mid M F U$$

```

1 PARSE-U (e1):
2 begin
3   if t ∈ {T_Star, T_Slash} then
4     op := PARSE-M()
5     e2 := PARSE-F()
6     return PARSE-U(MK-BINARY(op, e1, e2))
7   else return e1
8 end

```

$$A \rightarrow + \mid -$$

```
1 PARSE-A ():  
2 begin  
3   if  $t == \mathbf{T\_Plus}$  then  
4      $t := \text{NEXT-TOKEN}()$   
5     return Add  
6   else if  $t == \mathbf{T\_Minus}$  then  
7      $t := \text{NEXT-TOKEN}()$   
8     return Sub  
9   else ERROR()  
10 end
```

$$M \rightarrow * \mid /$$

```
1 PARSE-M():  
2 begin  
3   if  $t == \mathbf{T\_Star}$  then  
4      $t := \text{NEXT-TOKEN}()$   
5     return Mul  
6   else if  $t == \mathbf{T\_Slash}$  then  
7      $t := \text{NEXT-TOKEN}()$   
8     return Div  
9   else ERROR()  
10 end
```

$$\begin{array}{l}
 F \rightarrow - F \\
 \quad | (E) \\
 \quad | \langle ident \rangle \\
 \quad | \langle num \rangle
 \end{array}$$

```

1  PARSE-F ():
2  begin
3      switch t do
4          case T_Minus: do
5              t := NEXT-TOKEN()
6              e := PARSE-F()
7              return MK-UNARY(Un_minus, e)
8          case T_LParen: do
9              t := NEXT-TOKEN()
10             e := PARSE-E()
11             if t == T_RParen then
12                 t := NEXT-TOKEN()
13                 return e
14             else ERROR()
15         case T_Ident: do
16             e := MK-VAR(last-ident)
17             t := NEXT-TOKEN()
18             return e
19         case T_Number: do
20             e := MK-NUM(last-num)
21             t := NEXT-TOKEN()
22             return e
23         otherwise do ERROR()
24     end
25 end

```

- Pokud funkce končí rekurzivním voláním sebe sama, jako třeba funkce `PARSE-G()`, je možné nahradit tuto rekurzi iterací.
- Funkce `PARSE-E()` a `PARSE-G()` je možné spojit do jediné funkce.
- Podobně ve funkci `PARSE-U()` je možné nahradit rekurzi iterací a funkce `PARSE-T()` a `PARSE-U()` spojit do jediné funkce.

```
1  PARSE-E ():  
2  begin  
3      e1 := PARSE-T()  
4      while t ∈ {T_Plus, T_Minus} do  
5          op := PARSE-A()  
6          e2 := PARSE-T()  
7          e1 := MK-BINARY(op, e1, e2)  
8      end  
9      return e1  
10 end
```

```
1  PARSE-T ():  
2  begin  
3      e1 := PARSE-F()  
4      while t ∈ {T_Star, T_Slash} do  
5          op := PARSE-M()  
6          e2 := PARSE-F()  
7          e1 := MK-BINARY(op, e1, e2)  
8      end  
9      return e1  
10 end
```

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
```

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.)

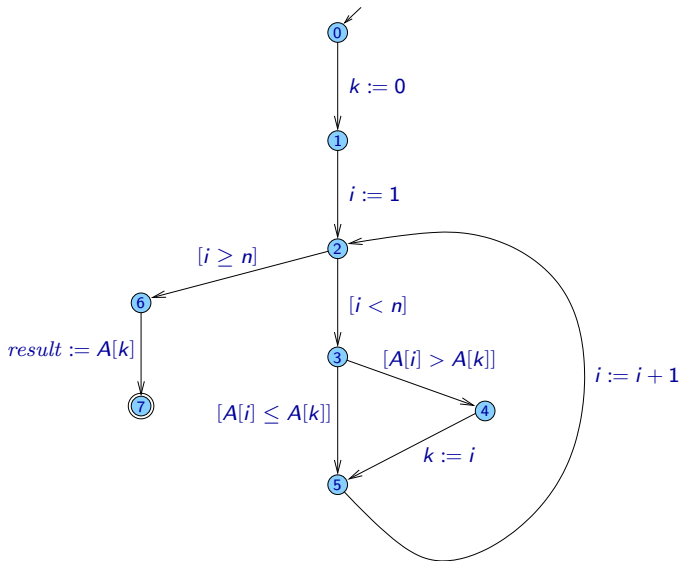
nebo

- čte vstupní data z paměti (např. jsou mu předány jako parametry)
- zapisuje data na do paměti (např. je vrátí jako návratovou hodnotu)

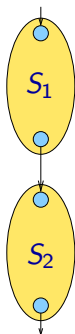
Instrukce lze zhruba rozdělit na dvě skupiny:

- instrukce 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, ...)
 - ...

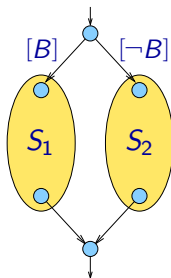
Graf řídicího toku



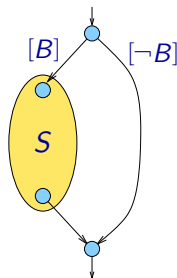
Některé základní konstrukce strukturovaného programování



$S_1; S_2$

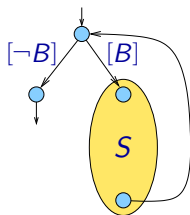


if B then S_1 else S_2

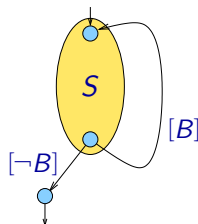


if B then S

Některé základní konstrukce strukturovaného programování

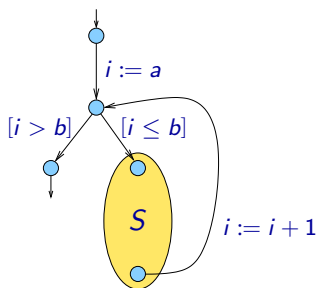


while B **do** S



do S **while** B

Některé základní konstrukce strukturovaného programování



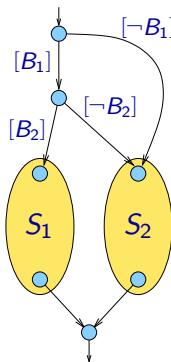
```
 $i := a$   
while  $i \leq b$  do  
     $S$   
     $i := i + 1$   
end
```

for $i := a$ **to** b **do** S

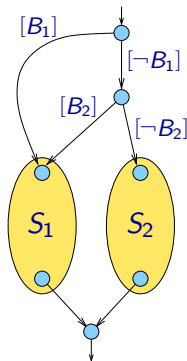
Některé základní konstrukce strukturovaného programování

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:

```
start:   $k := 0$   
         $i := 1$   
        goto  $L3$   
 $L1$ :   if  $A[i] \leq A[k]$  then goto  $L2$   
         $k := i$   
 $L2$ :    $i := i + 1$   
 $L3$ :   if  $i < n$  then goto  $L1$   
        return  $A[k]$ 
```

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

$$\begin{aligned}t_1 &:= i + s \\t_2 &:= 3 * j \\t_2 &:= t_2 + 1 \\t_3 &:= B[t_2] \\t_3 &:= t_3 + x \\t_3 &:= t_3 * y \\t_3 &:= t_3 + 8 \\A[t_1] &:= t_3\end{aligned}$$

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**.

Během výpočtu si stroj musí pamatovat:

- která instrukce 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 zahrnout i místa

- odkud se čtou vstupní data
- kam se zapisují výstupní data

Konfigurace — popis celkového stavu stroje v nějakém okamžiku během výpočtu

Příklad: Konfigurace tvaru

$$(q, mem)$$

kde

- q — aktuální řídicí stav
- mem — představuje aktuální obsah paměti stroje — jaké hodnoty jsou momentálně přiřazeny jednotlivým proměnným.

Příklad obsahu paměti mem :

$$\langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 0, \quad result: ? \rangle$$

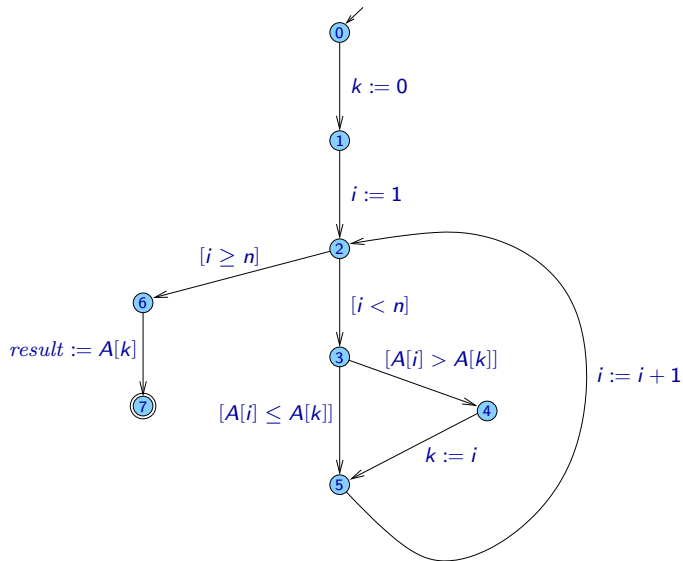
Příklad konfigurace:

$(2, \langle A: [3, 8, 1, 3, 6], \quad n: 5, \quad i: 1, \quad k: 0, \quad result: ? \rangle)$

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, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)

α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)

α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)

α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)

α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)

α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)

α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{13} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{13} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{14} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{13} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{14} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{15} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{13} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{14} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{15} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: ? \rangle$)
 α_{16} : (6, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: ? \rangle$)

Příklad: Výpočet, kde algoritmus **FIND-MAX** zpracovává vstup, kde $A = [3, 8, 1, 3, 6]$ a $n = 5$.

α_0 : (0, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: ?, result: ? \rangle$)
 α_1 : (1, $\langle A: [3, 8, 1, 3, 6], n: 5, i: ?, k: 0, result: ? \rangle$)
 α_2 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_3 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_4 : (4, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 0, result: ? \rangle$)
 α_5 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 1, k: 1, result: ? \rangle$)
 α_6 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_7 : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_8 : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 2, k: 1, result: ? \rangle$)
 α_9 : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{10} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{11} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 3, k: 1, result: ? \rangle$)
 α_{12} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{13} : (3, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{14} : (5, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 4, k: 1, result: ? \rangle$)
 α_{15} : (2, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: ? \rangle$)
 α_{16} : (6, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: ? \rangle$)
 α_{17} : (7, $\langle A: [3, 8, 1, 3, 6], n: 5, i: 5, k: 1, result: 8 \rangle$)

Provedením instrukce l se přejde z konfigurace α do konfigurace α' :

$$\alpha \xrightarrow{l} \alpha'$$

Výpočet může být:

- **Konečný:**

$$\alpha_0 \xrightarrow{l_0} \alpha_1 \xrightarrow{l_1} \alpha_2 \xrightarrow{l_2} \alpha_3 \xrightarrow{l_3} \alpha_4 \xrightarrow{l_4} \dots \xrightarrow{l_{t-2}} \alpha_{t-1} \xrightarrow{l_{t-1}} \alpha_t$$

kde α_t je koncová konfigurace

- **Nekonečný:**

$$\alpha_0 \xrightarrow{l_0} \alpha_1 \xrightarrow{l_1} \alpha_2 \xrightarrow{l_2} \alpha_3 \xrightarrow{l_3} \alpha_4 \xrightarrow{l_4} \dots$$

Výpočet je možné popsat dvěma různými způsoby:

- jako posloupnost konfigurací $\alpha_0, \alpha_1, \alpha_2, \dots$
- jako posloupnost provedených instrukcí $l_0, l_1, l_2, \dots$

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

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.

$$A = [3, 8, 1, 3, 6], \quad n = 5.$$

Pro tuto instanci je výstupem hodnota 8.

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.

Algoritmus *Alg* **není** korektním řešením 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.

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.

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 řídicího toku, můžeme pro **řídicí stav** q (tj. vrchol grafu) specifikovat invarianty, které platí v každé dosažitelné konfiguraci, kde je řídicím stavem q .

Invarianty můžeme zapisovat formulemi predikátové logiky:

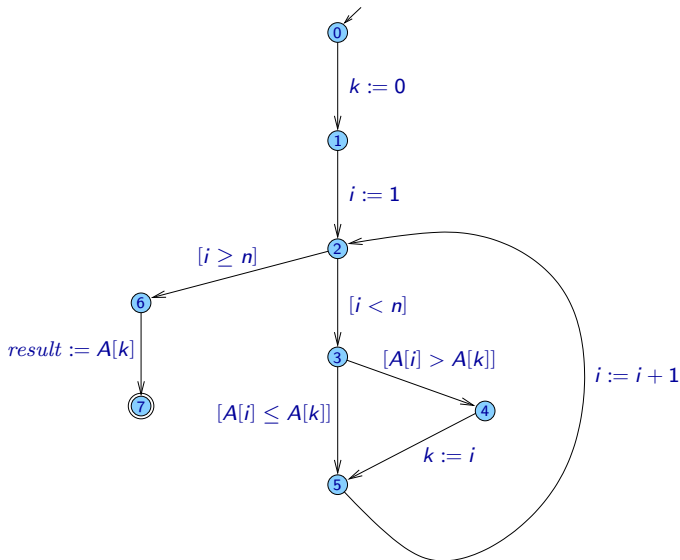
- **volné** proměnné odpovídají proměnným programu
- **valuace** je dána hodnotami proměnných programu v dané konfiguraci

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



Příklady invariantů:

- invariant v řídicím stavu q zapíšeme formulí φ_q

Invarianty v jednotlivých řídicích stavech (zatím jen hypotézy):

- $\varphi_0: (n \geq 1)$
- $\varphi_1: (n \geq 1) \wedge (k = 0)$
- $\varphi_2: (n \geq 1) \wedge (1 \leq i \leq n) \wedge (0 \leq k < i)$
- $\varphi_3: (n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- $\varphi_4: (n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k < i)$
- $\varphi_5: (n \geq 1) \wedge (1 \leq i < n) \wedge (0 \leq k \leq i)$
- $\varphi_6: (n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$
- $\varphi_7: (n \geq 1) \wedge (i = n) \wedge (0 \leq k < n)$

Příklady invariantů:

- invariant v řídicím stavu q zapíšeme formulí φ_q

Invarianty v jednotlivých řídicích stavech (zatím jen hypotézy):

- $\varphi_0: n \geq 1$
- $\varphi_1: n \geq 1, k = 0$
- $\varphi_2: n \geq 1, 1 \leq i \leq n, 0 \leq k < i$
- $\varphi_3: n \geq 1, 1 \leq i < n, 0 \leq k < i$
- $\varphi_4: n \geq 1, 1 \leq i < n, 0 \leq k < i$
- $\varphi_5: n \geq 1, 1 \leq i < n, 0 \leq k \leq i$
- $\varphi_6: n \geq 1, i = n, 0 \leq k < n$
- $\varphi_7: n \geq 1, i = n, 0 \leq k < n$

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 řídicího 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'

Zkontrolování instrukcí, které jsou testy podmínky:

- 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:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z}) (\varphi \wedge B \rightarrow \varphi')$$

Zkontrolování 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 $k := 3 * k + i + 1$:

$$(\forall n \in \mathbb{Z})(\forall i \in \mathbb{Z})(\forall k \in \mathbb{Z})(\forall k' \in \mathbb{Z}) (\varphi \wedge (k' = 3 * k + i + 1) \rightarrow \varphi'')$$

Dokončení ověření toho, že algoritmus pro nalezení maximálního prvku v poli vrací správný výsledek (za předpokladu, že skončí):

- $\psi_0: \varphi_0$
- $\psi_1: \varphi_1 \wedge (\forall j \in \mathbb{N})(0 \leq j < 1 \rightarrow A[j] \leq A[k])$
- $\psi_2: \varphi_2 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- $\psi_3: \varphi_3 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k])$
- $\psi_4: \varphi_4 \wedge (\forall j \in \mathbb{N})(0 \leq j < i \rightarrow A[j] \leq A[k]) \wedge (A[i] > A[k])$
- $\psi_5: \varphi_5 \wedge (\forall j \in \mathbb{N})(0 \leq j \leq i \rightarrow A[j] \leq A[k])$
- $\psi_6: \varphi_6 \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq A[k])$
- $\psi_7: \varphi_7 \wedge (result = A[k]) \wedge (\forall j \in \mathbb{N})(0 \leq j < n \rightarrow A[j] \leq result) \wedge (\exists j \in \mathbb{N})(0 \leq j < n \wedge A[j] = result)$

Často není třeba specifikovat invarianty ve všech řídicích 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.

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 $A[j] \leq A[k]$.

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

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')

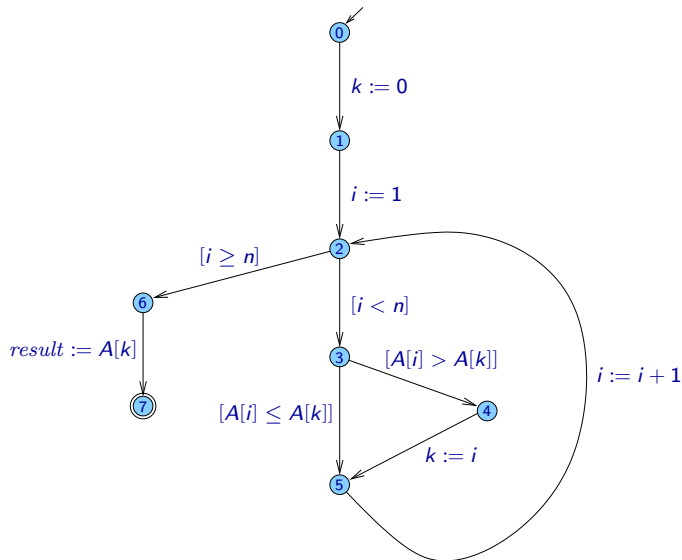
Jako množinu W je možno použít například:

- Množinu přirozených čísel $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ s uspořádáním $\leq$.
- Množinu vektorů přirozených čísel s lexikografickým uspořádáním, tj. s uspořádáním, kde vektor $(a_1, a_2, \dots, a_m)$ je menší než vektor $(b_1, b_2, \dots, b_n)$, 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



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

- 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.

- 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.

- **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).

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 závisí 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 — 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.

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

- 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

- 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

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)

$t(w)$ — doba výpočtu algoritmu Alg nad vstupem w na stroji $\mathcal{M}$

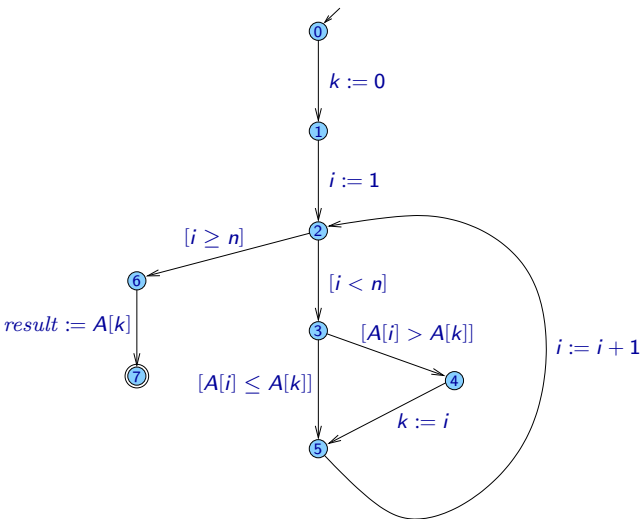
V jakých jednotkách dobu výpočtu udávat? (Uvidíme, že ve skutečnosti to při použití asymptotické notace není podstatné.)

- **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 jeden 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)
 - ...

Řekněme, že máme algoritmus reprezentován ve formě grafu řídicího 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

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 $size : In \rightarrow \mathbb{N}$.)

Definice

Časová složitost algoritmu Alg v nejhorším případě je funkce $T : \mathbb{N} \rightarrow \mathbb{R}^+$, 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 $t(w) = T(n)$.

Z definice vidíme, že časová složitost algoritmu je funkce, jejíž přesné hodnoty závisí 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 $size$).

Někdy se také určuje časová složitost algoritmu v **průměrném případě**:

- Musí se předpokládat určité **pravděpodobnostní 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.

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 $c_0, c_1, \dots, c_8$, které udávají dobu trvání jednotlivých instrukcí — nebudeme počítat s konkrétními čísly.

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

$$t(w) = c_0m_0 + c_1m_1 + \dots + c_8m_8,$$

kde $m_0, m_1, \dots, m_8$ jsou čísla udávající, kolikrát je daná instrukce při výpočtu nad vstupem w provedena.

Časová složitost algoritmu

Instr.	doba	počet provedení	hodnota m_i
$k := 0$	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

Dosažením do

$$t(w) = c_0 m_0 + c_1 m_1 + \dots + c_8 m_8,$$

dostaneme

$$t(w) = d_1 + d_2 \cdot (n - 1) + d_3 \cdot (n - 1 - \ell) + d_4 \cdot \ell,$$

kde

$$d_1 = c_0 + c_1 + c_3 + c_8$$

$$d_3 = c_4$$

$$d_2 = c_2 + c_7$$

$$d_4 = c_5 + c_6$$

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 $[0, 0, \dots, 0]$ nebo třeba $[n, n-1, n-2, \dots, 2, 1]$
- Pokud $d_3 \leq d_4$ — nejhorší jsou případy, kdy má ℓ co největší hodnotu
 $\ell = n-1$ — například vstupy tvaru $[0, 1, \dots, n-1]$

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

$$T(n) = 33n - 8$$

Pokud bychom se spokojili s tím, že časová složitost algoritmu **FIND-MAX** je nějaká funkce tvaru

$$T(n) = an + b,$$

kde by nás ale nezajímal 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 $1\ \mu\text{s}$ (10^{-6} 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	Nárůst
$T_1(n) = n$	10^{15}	$1000\times$
$T_2(n) = n^3$	10^5	$10\times$
$T_3(n) = 2^n$	50	+10

V následujícím se zaměříme na funkce typu $f : \mathbb{N} \rightarrow \mathbb{R}$, 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 $n \geq 5$.

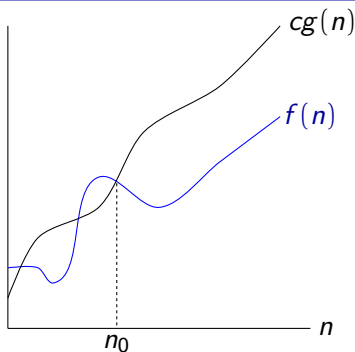
Vezměme si libovolnou funkci $f : \mathbb{N} \rightarrow \mathbb{R}$. Zápisy $O(f)$, $\Omega(f)$ a $\Theta(f)$ označují **množiny funkcí** typu $\mathbb{N} \rightarrow \mathbb{R}$, kde:

- $O(f)$ – 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 $g : \mathbb{N} \rightarrow \mathbb{R}$. Pro funkci $f : \mathbb{N} \rightarrow \mathbb{R}$ platí $f \in O(g)$ právě tehdy, když

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \leq c g(n)).$$

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 $c = 3$.

$$cg(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2$$

Potřebujeme najít takové n_0 , aby pro každé $n \geq n_0$ platilo současně

$$2n^2 \geq 2n^2 \qquad \frac{1}{2}n^2 \geq 3n \qquad \frac{1}{2}n^2 \geq 7$$

Snadno ověříme, že například $n_0 = 6$ vyhovuje těmto požadavkům.

Pak pro každé $n \geq 6$ platí $cg(n) \geq f(n)$:

$$cg(n) = 3n^2 = 2n^2 + \frac{1}{2}n^2 + \frac{1}{2}n^2 \geq 2n^2 + 3n + 7 = f(n)$$

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

$$2n^2 \geq 2n^2$$

$$3n^2 \geq 3n$$

$$7n^2 \geq 7$$

Uvedené vztahy zjevně platí pro $n_0 = 1$, takže pro každé $n \geq 1$ platí $cg(n) \geq f(n)$:

$$cg(n) = 12n^2 = 2n^2 + 3n^2 + 7n^2 \geq 2n^2 + 3n + 7 = f(n)$$

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

$$f(n) = a \cdot n^k \qquad g(n) = b \cdot n^\ell$$

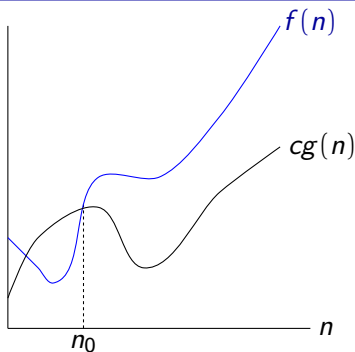
Pro každé takové funkce f a g platí $f \in O(g)$:

Důkaz: Zvolme $c = \frac{a}{b}$.

Vzhledem k tomu, že pro $n \geq 1$ zjevně platí $n^k \leq n^\ell$ (protože $k \leq \ell$), 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 $g : \mathbb{N} \rightarrow \mathbb{R}$. Pro funkci $f : \mathbb{N} \rightarrow \mathbb{R}$ platí $f \in \Omega(g)$ právě tehdy, když

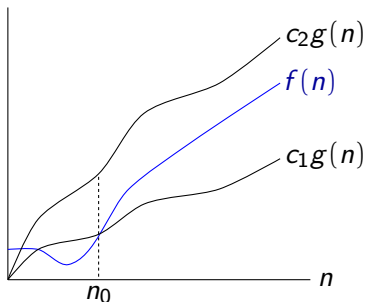
$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(c g(n) \leq f(n)).$$

Není těžké zdůvodnit, že platí následující tvrzení:

Pro libovolné funkce f a g platí:

$$f \in O(g) \quad \text{právě tehdy, když} \quad g \in \Omega(f)$$

Asymptotická notace – symbol Θ



Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \rightarrow \mathbb{R}$. Pro funkci $f : \mathbb{N} \rightarrow \mathbb{R}$ platí $f \in \Theta(g)$ právě tehdy, když

$$(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(c_1 g(n) \leq f(n) \leq c_2 g(n)).$$

Asymptotická notace – symbol Θ

Pro libovolné funkce f a g platí:

$f \in \Theta(g)$ právě tehdy, když $f \in O(g)$ a $f \in \Omega(g)$

$f \in \Theta(g)$ právě tehdy, když $f \in O(g)$ a $g \in O(f)$

$f \in \Theta(g)$ právě tehdy, když $g \in \Theta(f)$

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 $f \in \Theta(h)$

Příklady:

$$n \in O(n^2)$$

$$1000n \in O(n)$$

$$2^{\log_2 n} \in \Theta(n)$$

$$n^3 \notin O(n^2)$$

$$n^2 \notin O(n)$$

$$n^3 + 2^n \notin O(n^2)$$

$$n^3 \in O(n^4)$$

$$0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2)$$

$$n^3 - n^2 \log_2^3 n + 1000n - 10^{100} \in \Theta(n^3)$$

$$n^3 + 1000n - 10^{100} \in O(n^3)$$

$$n^3 + n^2 \notin \Theta(n^2)$$

$$n! \notin O(2^n)$$

- Existují dvojice funkcí f a g takové, že

$$f \notin O(g) \quad \text{a} \quad g \notin O(f),$$

například

$$f(n) = n \quad g(n) = n^{1+\sin(n)}.$$

- $O(1)$ je množina všech **omezených** funkcí, tj. funkcí jejichž funkční hodnoty jsou shora omezeny nějakou konstantou.

- 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 $f_1 \cdot g_1 \in \Theta(f_2 \cdot g_2)$

- 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.

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 $f \in O(g)$.

Řekněme, že bychom chtěli analyzovat časovou složitost $T(n)$ nějakého algoritmu, který se skládá z instrukcí $l_1, l_2, \dots, l_k$:

- Pokud $m_1, m_2, \dots, m_k$ jsou počty provedení jednotlivých instrukcí pro nějaký vstup w (tj. pro vstup x se instrukce l_i provede m_i krát), tak celkový počet instrukcí provedených pro vstup w je

$$T(n) = c_1 m_1 + c_2 m_2 + \dots + c_k m_k.$$

- Vezměme si funkce $f_1, f_2, \dots, f_k$, kde $f_i : \mathbb{N} \rightarrow \mathbb{R}$, přičemž $f_i(n)$ je maximum z počtu provedení instrukce l_i pro všechny vstupy velikosti n .
- Zjevně platí, že pro libovolnou funkci f_i je $T \in \Omega(f_i)$.
- Zjevně také platí $T \in O(f_1 + f_2 + \dots + f_k)$.

- 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

$$T \in \Theta(f_i).$$

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.

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  $i \geq 0$  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 $\Theta(T(n))$.

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
$$m = 1 + 2 + \dots + (n - 1) = (1 + (n - 1)) \cdot \frac{n-1}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$
- Celková časová složitost algoritmu **INSERTION-SORT** v nejhorším případě je tedy $\Theta(n^2)$.

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.

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 $S : \mathbb{N} \rightarrow \mathbb{N}$, 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	

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.

- 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.

Polynom — funkce tvaru

$$a_k n^k + a_{k-1} n^{k-1} + \dots + a_2 n^2 + a_1 n + a_0$$

kde $a_0, a_1, \dots, a_k$ jsou konstanty.

Příklady polynomů:

$$4n^3 - 2n^2 + 8n + 13$$

$$2n + 1$$

$$n^{100}$$

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:

$$O(n)$$

$$O(n \log n)$$

$$O(n^2)$$

$$O(n^5)$$

$$O(\sqrt{n})$$

$$O(n^{100})$$

Složitost algoritmů

Funkce jako 2^n nebo $n!$ polynomiální nejsou — pro libovolně velkou konstantu k platí

$$2^n \in \Omega(n^k)$$

$$n! \in \Omega(n^k)$$

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

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.

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
```

- 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 $a_0, a_1, \dots, a_{n-1}$, kde

$$a_i = a_0 + i \cdot d,$$

kde d je nějaká konstanta nezávislá na i .

Poznámka: V aritmetické posloupnosti tedy pro všechna i platí $a_{i+1} = a_i + d$.

Součet aritmetické posloupnosti:

$$\sum_{i=0}^{n-1} a_i = a_0 + a_1 + \dots + a_{n-1} = \frac{1}{2}n(a_{n-1} + a_0)$$

Příklad:

$$1 + 2 + \dots + n = \frac{1}{2}n(n+1) = \frac{1}{2}n^2 + \frac{1}{2}n = \Theta(n^2)$$

Konkrétně například pro $n = 100$ je

$$1 + 2 + \dots + 100 = 50 \cdot 101 = 5050.$$

Poznámka: Stačí si umědomit, že

$$1 + 2 + \dots + 100 = (1 + 100) + (2 + 99) + \dots + (50 + 51),$$

kde sčítáme celkem 50 dvojic, kde součet každé dvojice je 101.

Geometrická posloupnost — číselná řada $a_0, a_1, \dots, a_n$, kde

$$a_i = a_0 \cdot q^i,$$

kde q je nějaká konstanta nezávislá 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 $q \neq 1$):

$$\sum_{i=0}^n a_i = a_0 + a_1 + \dots + a_n = a_0 \frac{q^{n+1} - 1}{q - 1}$$

Příklad:

$$1 + q + q^2 + \dots + q^n = \frac{q^{n+1} - 1}{q - 1}$$

Speciálně pro $q = 2$:

$$1 + 2^1 + 2^2 + 2^3 + \dots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2 \cdot 2^n - 1 = \Theta(2^n)$$

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 $\Theta(n \log n)$.

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 $\Theta(\log n)$.

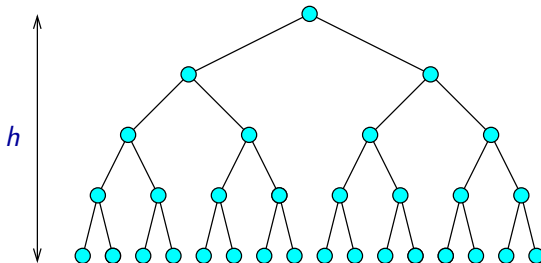
- Pomocí n bitů je možno reprezentovat čísla od 0 do $2^n - 1$.
- Minimální počet bitů potřebných pro uložení přirozeného čísla x reprezentovaného binárně je

$$\lceil \log_2(x + 1) \rceil.$$

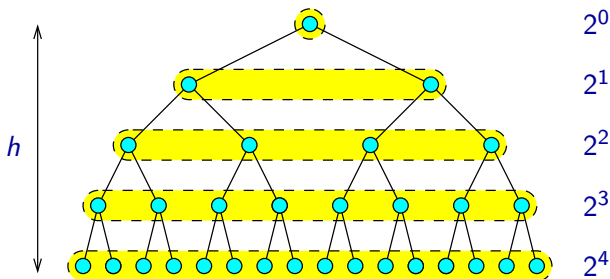
- Dokonale vyvážený binární strom o výšce h má $2^{h+1} - 1$ vrcholů, z čehož 2^h jsou listy.
- Dokonale vyvážený binární strom o n vrcholech má výšku zhruba $\log_2 n$.

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.

Dokonale vyvážený binární strom výšky h :



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.

34	42	58	61
----	----	----	----

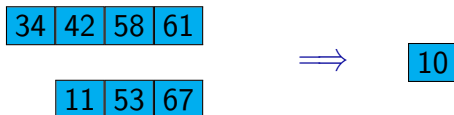


10	11	53	67
----	----	----	----

Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

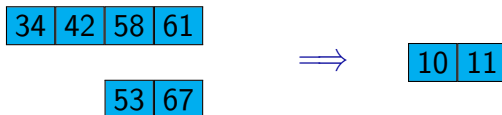
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

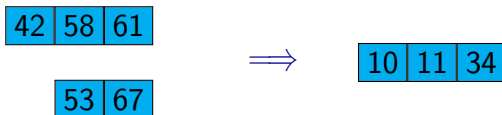
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

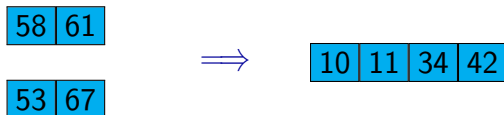
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

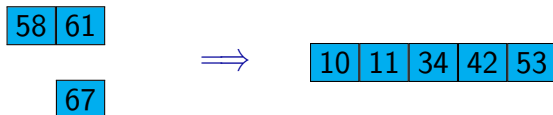
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

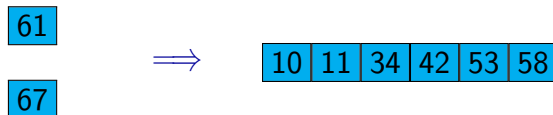
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Příklad: Algoritmus **MERGE-SORT**.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.

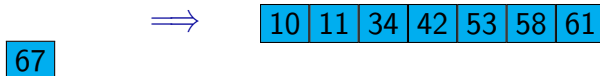
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



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.



10	11	34	42	53	58	61	67
----	----	----	----	----	----	----	----

Algoritmus 4: Merge sort

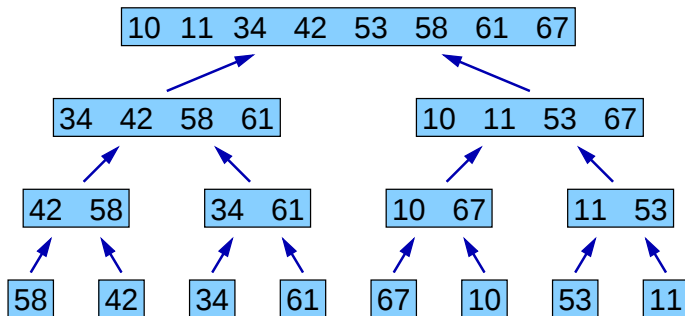
```
1 MERGE-SORT ( $A, p, r$ ):  
2 begin  
3   if  $r - p > 1$  then  
4      $q := \lfloor (p + r) / 2 \rfloor$   
5     MERGE-SORT( $A, p, q$ )  
6     MERGE-SORT( $A, q, r$ )  
7     MERGE( $A, p, q, r$ )  
8   end  
9 end
```

Pro setřídění pole A , které obsahuje prvky $A[0], A[1], \dots, A[n-1]$, zavoláme $\text{MERGE-SORT}(A, 0, n)$.

Poznámka: Procedura $\text{MERGE}(A, p, q, r)$ spojí setříděné posloupnosti uložené v $A[p \dots q-1]$ a $A[q \dots r-1]$ do jedné posloupnosti uložené v $A[p \dots r-1]$.

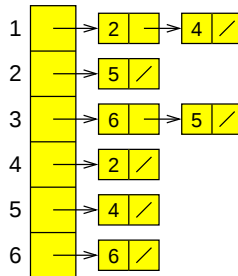
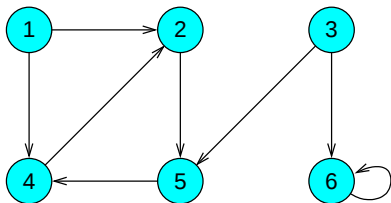
Složitost algoritmů

Vstup: 58, 42, 34, 61, 67, 10, 53, 11



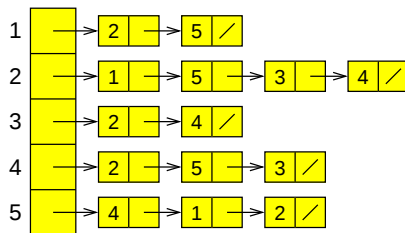
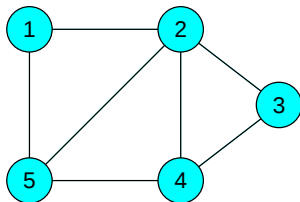
Strom rekurzivních volání má $\Theta(\log n)$ úrovní. Na každé úrovni se provede $\Theta(n)$ operací. Časová složitost algoritmu **MERGE-SORT** je $\Theta(n \log n)$.

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

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

Nalezení nejkratší cesty v 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 $\Theta(n + m)$.

Algoritmus 5: Prohledávání do šířky

```
1 BFS (  $G, s$  ):  
2 begin  
3   BFS-INIT(  $G, s$  )  
4   ENQUEUE(  $Q, s$  )  
5   while  $Q \neq \emptyset$  do  
6      $u :=$  DEQUEUE(  $Q$  )  
7     for each  $v \in \text{edges}[u]$  do  
8       if  $\text{color}[v] = \text{WHITE}$  then  
9          $\text{color}[v] := \text{GRAY}$   
10         $d[v] := d[u] + 1$   
11         $\text{pred}[v] := u$   
12        ENQUEUE(  $Q, v$  )  
13      end  
14    end  
15     $\text{color}[u] := \text{BLACK}$   
16  end  
17 end
```

Algoritmus 6: Prohledávání do šířky — inicializace

```
1  BFS-INIT ( $G, s$ ):  
2  begin  
3    for each  $u \in V - \{s\}$  do  
4       $color[u] := \text{WHITE}$   
5       $d[u] := \infty$   
6       $pred[u] := \text{NIL}$   
7    end  
8     $color[s] := \text{GRAY}$   
9     $d[s] := 0$   
10    $pred[s] := \text{NIL}$   
11    $Q := \emptyset$   
12 end
```

Problém „Prvočíselnost“

Vstup: Přirozené číslo x .

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

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

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

Problémům, kde množina výstupů je $\{A_{NO}, N_E\}$ se říká **rozhodovací problémy**.

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

Příklad:

Problém „Prvočíselnost“

Vstup: Přirozené číslo x .

Otázka: Je x prvočíslo?

Testování prvočíselnosti

Jednoduchý algoritmus řešící problém „Prvočíselnost“ může vypadat například takto:

- Otestovat triviální případy (např. zda $x \leq 2$ nebo zda je x sudé).
- Vyzkoušet postupně x dělit všemi lichými čísly v intervalu $3, \dots, \lfloor \sqrt{x} \rfloor$.

Označme n počet bitů v zápise čísla x , tj. $n = \lceil \log_2(x + 1) \rceil$.
Tuto hodnotu n budeme brát jako velikost vstupu.

Všimněme si, že číslo $\lfloor \sqrt{x} \rfloor$ má zhruba $n/2$ bitů.

Lichých čísel v intervalu $3, \dots, \lfloor \sqrt{x} \rfloor$ je tedy zhruba $2^{(n/2)-1}$, takže tento jednoduchý algoritmus má časovou složitost $2^{\Theta(n)}$.

Testování prvočíselnosti

Tento jednoduchý algoritmus s exponenciální složitostí (resp. i jeho různá vylepšení) je nepoužitelný pro čísla, která mají tisíce bitů.

Testování prvočíselnosti takto velkých čísel hraje ale důležitou roli například v **kryptografii**.

Teprve od roku 2003 je znám polynomiální algoritmus. Původní verze algoritmu měla složitost $O(n^{12+\epsilon})$, později o něco zlepšen ($O(n^{7.5})$). Aktuálně má nejrychlejší známý algoritmus složitost $O(n^6)$.

V praxi se v kryptografii používají pro testování **randomizované algoritmy**:

- Solovay–Strassen
- Miller–Rabin

(Složitost obou těchto algoritmů je $O(n^3)$.)

Randomizovaný algoritmus:

- Během výpočtu používá generátor náhodných čísel.
- Pro tentýž vstup může daný algoritmus v různých bězích dávat různé výstupy.
- Výstup nemusí být vždy korektní, ale pravděpodobnost toho, že výstup není korektní, je možné nějakým způsobem omezit.

Například oba výše uvedené randomizované algoritmy pro testování prvočíselnosti se chovají následovně:

- Pokud je x je prvočíslo, vrátí vždy odpověď **ANO**.
- Pokud x není prvočíslo, je pravděpodobnost odpovědi **NE** minimálně 50%, ale existuje až 50% šance, že program vrátí chybnou odpověď **ANO**.

Program můžeme volat opakovaně (k krát):

- Pokud program vrátí alespoň jednou odpověď **NE**, víme na 100%, že x není prvočíslo.
- Pokud program vrátí pokaždé **ANO**, je pravděpodobnost toho, že x není prvočíslo, nejvýše $\frac{1}{2^k}$.

Pro dostatečně velké k je pravděpodobnost takové chyby zanedbatelně malá.

Poznámka: Například pro $k = 100$ je pravděpodobnost této chyby menší než pravděpodobnost toho, že počítač bude během dané mikrosekundy zasažen meteoritem (za předpokladu, že každých 1000 let je meteoritem zničeno alespoň 100 m^2 zemského povrchu).

Následující problém vypadá na první pohled velmi podobně jako testování prvočíselnosti:

Problém „Faktorizace“

Vstup: Přirozené číslo x , kde $x > 1$.

Výstup: Prvočísla $p_1, p_2, \dots, p_m$ taková, že $x = p_1 \cdot p_2 \cdot \dots \cdot p_m$.

Ve skutečnosti je tento problém podstatně obtížnější než testování prvočíselnosti.

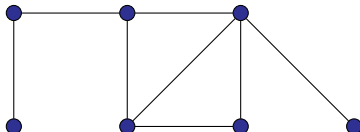
Není znám žádný efektivní (polynomiální) algoritmus (ani randomizovaný).

Další příklady problémů

Problém nezávislé množiny (IS — independent set)

Vstup: Neorientovaný graf G , číslo k .

Otázka: Existuje v grafu G nezávislá množina velikosti k ?



$k = 4$

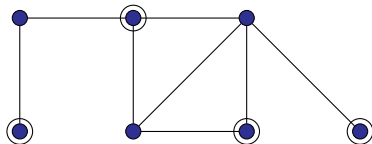
Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

Další příklady problémů

Problém nezávislé množiny (IS — independent set)

Vstup: Neorientovaný graf G , číslo k .

Otázka: Existuje v grafu G nezávislá množina velikosti k ?

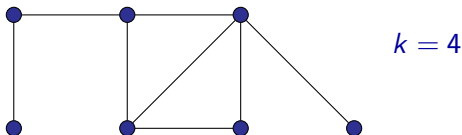


$k = 4$

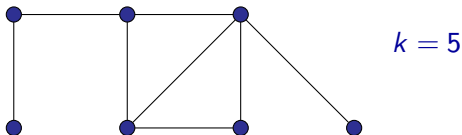
Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

Další příklady problémů

Příklad instance, kde je odpověď **ANO**:



Příklad instance, kde je odpověď **NE**:



Další příklady problémů

- Množina obsahující n prvků má 2^n podmnožin.

Algoritmus řešící daný problém **hrubou silou**, kdy testuje příslušnou podmínku pro všechny podmnožiny dané množiny.

- Stačí se omezit na podmnožiny velikosti k . Těchto podmnožin je

$$\binom{n}{k}$$

Pro některé hodnoty k však celkový počet těchto podmnožin není o moc menší než 2^n :

Dá se například poměrně snadno ukázat, že

$$\binom{n}{\lfloor n/2 \rfloor} \geq \frac{2^n}{n}.$$

Další příklady problémů

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 $2^{\Theta(n)}$.

Nerozhodnutelné 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).

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

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

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

Halting Problem

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

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

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

Halting problem

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

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

Halting Problem

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

Mohli bychom tedy vytvořit podprogram H , deklarovaný jako

$\text{Bool } H(\text{String kod}, \text{String vstup})$

kde $H(P, x)$ vrátí:

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

Poznámka: Řekněme, že podprogram $H(P, x)$ by vracel **false** v případě, že P není syntakticky správný kód programu.

Halting Problem

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

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

`loop: goto loop`

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

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

Halting Problem

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

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

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

Převody mezi problémy

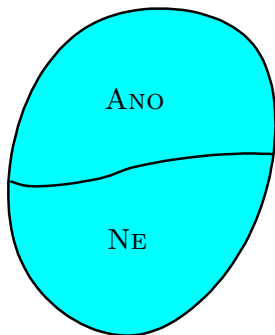
Pokud máme o nějakém (rozhodovacím) problému dokázáno, že je nerozhodnutelný, můžeme ukázat nerozhodnutelnost dalších problémů pomocí redukci (převodů) mezi problémy.

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

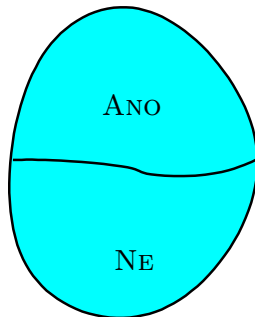
- Jako vstup může dostat libovolnou instanci problému P_1 .
- K instanci problému P_1 , kterou dostane jako vstup (označme ji w), vyprodukuje jako svůj výstup instanci problému P_2 (označme ji $Alg(w)$).
- Platí, že pro vstup w je v problému P_1 odpověď **ANO** právě tehdy, když pro vstup $Alg(w)$ je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

Převody mezi problémy

vstupy problému P_1



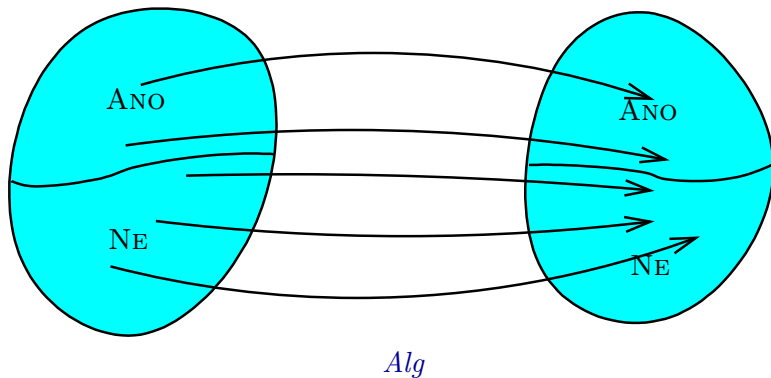
vstupy problému P_2



Převody mezi problémy

vstupy problému P_1

vstupy problému P_2



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

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

Řešení problému P_1 pro vstup x :

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

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

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

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

Pro účely různých důkazů a převodů mezi problémy je výhodné mít jazyk $\mathcal{L}$ a příslušný stroj vykonávající programy v tomto jazyce co nejjednodušší:

- co nejmenší počet různých druhů instrukcí
- co nejprimitivnější instrukce
- co nejprimitivnější typ dat, se kterými se pracuje
- není třeba se ohlížet na to, jak jednoduše nebo složitě se v takovém jazyce programuje (může být uživatelsky velmi nepřívětivý)

Na druhou stranu, takový jazyk (resp. stroj) ale musí být dostatečně obecný na to, ale se do něj dal přeložit program napsaný v libovolném jiném programovacím jazyce.

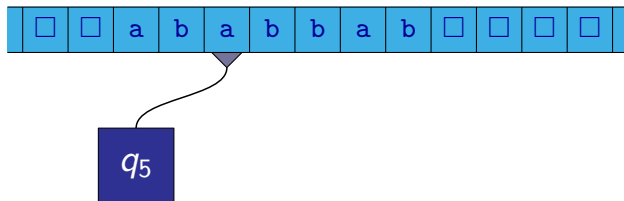
Takovým jazykům (resp. strojům), které jsou dostatečně obecné na to, aby se do nich (resp. do jejich instrukcí) daly přeložit programy napsané v libovolném jiném programovacím jazyce se říká **Turingovsky úplné**.

Příklady některých Turingovsky úplných **výpočetních modelů** (jazyků či strojů) často používaných pro účely důkazů:

- Turingův stroj (Alan Turing)
- lambda kalkulus (Alonzo Church)
- Minského stroj (Marvin Minsky)
- ...

Turingův stroj:

- Rozšíříme deterministický konečný automat následujícím způsobem:
 - pohyb čtecí hlavy oběma směry
 - možnost zápisu na pásku
 - prodloužení pásky do nekonečna



Churchova-Turingova teze

Každý algoritmus je možné realizovat nějakým Turingovým strojem.

Není to věta, kterou by bylo možno dokázat v matematickém smyslu – není formálně definováno, co je to algoritmus.

Tezi formulovali nezávisle na sobě v polovině 30. let 20. století Alan Turing a Alonzo Church.

Halting problem

Pro účely důkazů se Halting problem nejčastěji používá v následující podobě:

Halting problem

Vstup: Popis Turingova stroje M a slovo w .

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

Další nerozhodnutelné problémy

S následujícím 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

Vstupem je množina typů kachliček, jako třeba:

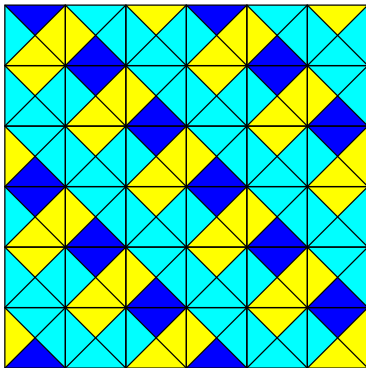


Otázka je, zda je možné použitím daných typů kachliček pokrýt každou libovolně velkou konečnou plochu tak, aby všechny kachličky spolu sousedily stejnými barvami.

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:

abb	a	bab	baba	aba
bbab	aa	ab	aa	a

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

a	abb	abb	baba	abb	aba
aa	bbab	bbab	aa	bbab	a

Nahoře i dole vznikne slovo **aabbabbbabaabbaba**.

Další nerozhodnutelné problémy

Redukcí z předchozího problému se dá snadno ukázat nerozhodnutelnost některých dalších problémů z oblasti bezkontextových gramatik:

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?

Problém

Vstup: Uzavřená formule predikátové logiky (prvního řádu), ve které mohou být použity jako predikátové symboly pouze $=$ a $<$, jako funkční symboly pouze $+$ a $*$ a jako konstantní symboly pouze 0 a 1 .

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

Příklad vstupu:

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

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 je nerozhodnutelný už velmi omezený případ:

Desátý Hilbertův problém

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

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

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 prvního řádu.

Otázka: Platí $\models \varphi$?

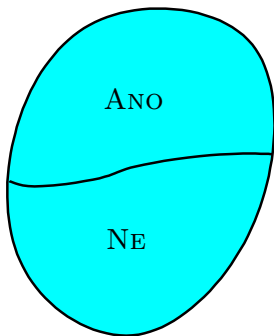
Poznámka: Zápis $\models \varphi$ znamená, že formule φ je logicky platná, tj. pravdivá v každé interpretaci.

NP-úplné problémy

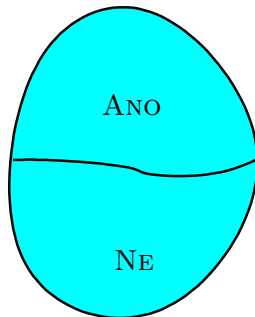
Problém P_1 je **polynomiálně převeritelný** na problém P_2 , jestliže existuje algoritmus Alg s polynomiální časovou složitostí, který převádí problém P_1 na problém P_2 .

Polynomiální převody mezi problémy

vstupy problému P_1



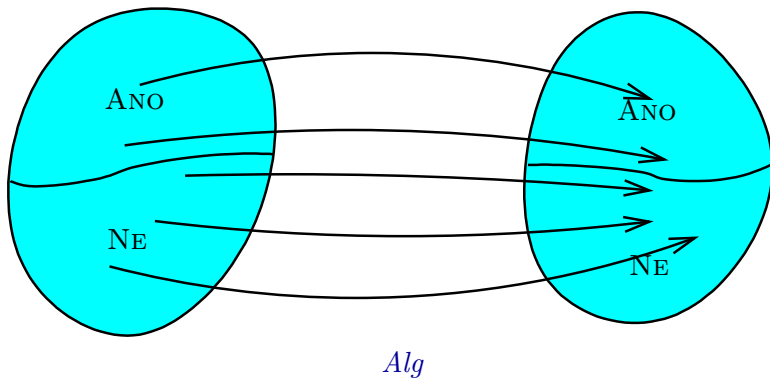
vstupy problému P_2



Polynomiální převody mezi problémy

vstupy problému P_1

vstupy problému P_2



Polynomiální převody mezi problémy

Řekněme, že problém P_1 je polynomiálně převeditelný na problém P_2 , tj. existuje (polynomiální) algoritmus Alg realizující tento převod.

Pokud pro problém P_2 existuje polynomiální algoritmus, pak i pro problém P_1 existuje polynomiální algoritmus.

Řešení problému P_1 pro vstup w :

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

Z toho plyne:

Pokud neexistuje polynomiální algoritmus pro problém P_1 , tak neexistuje ani polynomiální algoritmus pro problém P_2 .

Polynomiální převody mezi problémy

Existuje velká skupina algoritmických problémů označovaných jako **NP-úplné** problémy, které:

- jsou algoritmicky řešitelné v exponenciálním čase
- 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
- jsou všechny navzájem polynomiálně převeditelné

Poznámka: Toto není definice NP-úplných problémů. Ta bude uvedena později.

Typickým příkladem NP-úplného problému je problém SAT:

SAT (splnitelnost booleovských formulí)

Vstup: Booleovská formule φ .

Otázka: Je φ splnitelná?

Příklad:

Formule $\varphi_1 = x_1 \wedge (\neg x_2 \vee x_3)$ je splnitelná:

např. při ohodnocení v , kde $v(x_1) = 1$, $v(x_2) = 0$, $v(x_3) = 1$, je formule φ_1 pravdivá.

Formule $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ není splnitelná: je nepravdivá při každém ohodnocení v .

3-SAT je varianta problému SAT, ve které se omezujeme na formule určitého speciálního typu:

3-SAT

Vstup: Formule φ v konjunktivní normální formě, kde každá klauzule obsahuje právě 3 literály.

Otázka: Je φ splnitelná?

Problém 3-SAT

Připomenutí některých pojmů:

- **Literál** je formule tvaru x nebo $\neg x$, kde x je atomický výrok.
- **Klauzule** je disjunkce literálů.

Příklady: $x_1 \vee \neg x_2$ $\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ x_6

- Formule je v **konjunktivní normální formě (KNF)**, jestliže je konjunkcí klauzulí.

Příklad: $(x_1 \vee \neg x_2) \wedge (\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}) \wedge x_6$

V případě problému 3-SAT tedy vyžadujeme, aby formule φ byla v KNF a navíc, aby každá klauzule obsahovala právě tři literály.

Příklad:

$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Následující formule je splnitelná:

$$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

Je pravdivá např. při ohodnocení v , kde

$$v(x_1) = 0$$

$$v(x_2) = 1$$

$$v(x_3) = 0$$

$$v(x_4) = 1$$

Naproti tomu následující formule není splnitelná:

$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

Ukážeme si příklad polynomiálního převodu problému 3-SAT na problém nezávislé množiny (IS).

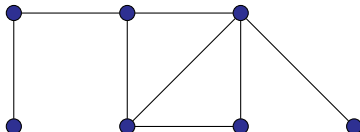
Poznámka: Jak 3-SAT, tak IS jsou příklady NP-úplných problémů.

Problém nezávislé množiny (IS)

Problém nezávislé množiny (IS)

Vstup: Neorientovaný graf G , číslo k .

Otázka: Existuje v grafu G nezávislá množina velikosti k ?



$k = 4$

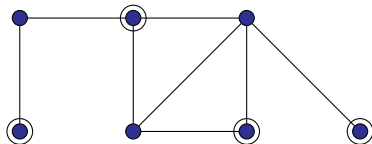
Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

Problém nezávislé množiny (IS)

Problém nezávislé množiny (IS)

Vstup: Neorientovaný graf G , číslo k .

Otázka: Existuje v grafu G nezávislá množina velikosti k ?

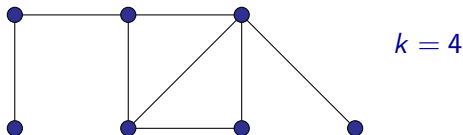


$k = 4$

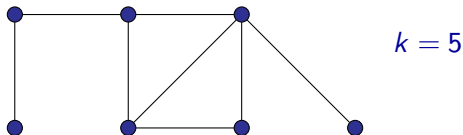
Poznámka: **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

Problém nezávislé množiny (IS)

Příklad instance, kde je odpověď **ANO**:



Příklad instance, kde je odpověď **NE**:



Popíšeme (polynomiální) algoritmus, který bude mít následující vlastnosti:

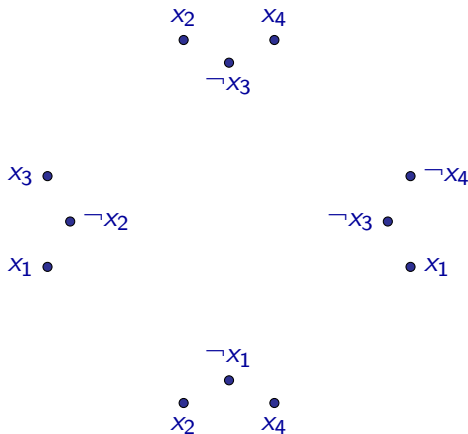
- **Vstup:** Libovolná instance problému 3-SAT, tj. formule φ v konjunktivní normální formě, kde každá klauzule obsahuje právě tři literály.
- **Výstup:** Instance problému IS, tj. neorientovaný graf G a číslo k .
- Navíc bude pro libovolný vstup (tj. pro libovolnou formuli φ ve výše uvedeném tvaru) zaručeno následující:
V grafu G bude existovat nezávislá množina velikosti k právě tehdy, když formule φ bude splnitelná.

Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

Převod 3-SAT na IS

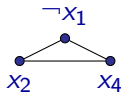
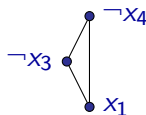
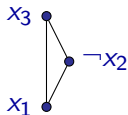
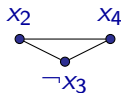
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



Pro každý výskyt literálu přidáme do grafu jeden vrchol.

Převod 3-SAT na IS

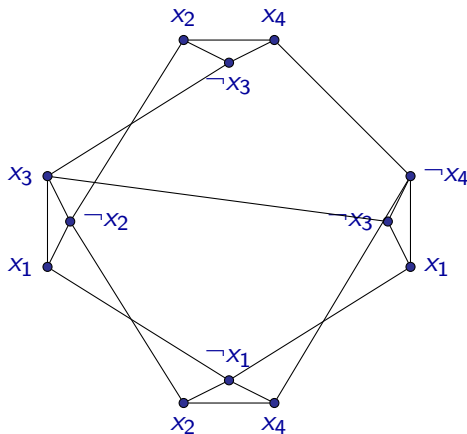
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



Vrcholy odpovídající výskytům literálů patřícím do stejné klauzule spojíme hranami.

Převod 3-SAT na IS

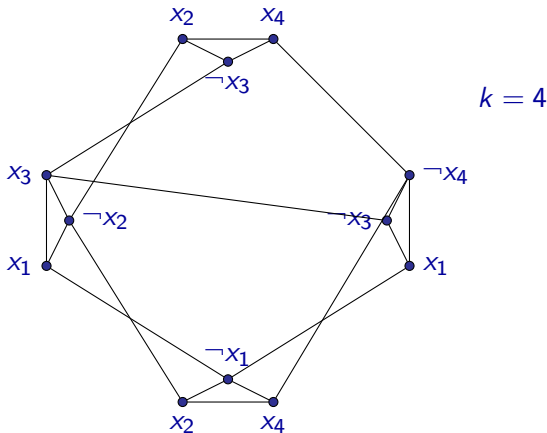
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



Dvojice vrcholů odpovídající literálům x_i a $\neg x_i$ spojíme hranami.

Převod 3-SAT na IS

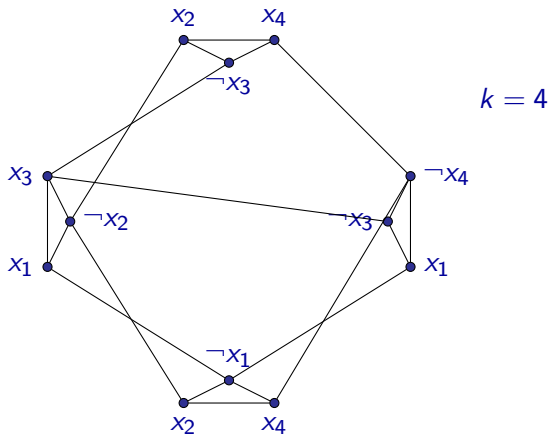
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



Číslo k položíme rovno počtu klauzulí.

Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



Vytvořený graf a číslo k vydá algoritmus jako výstup.

Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

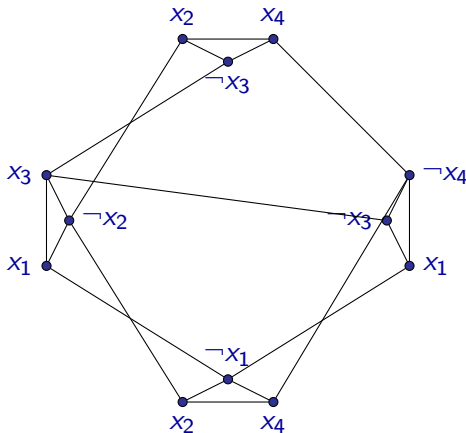
$$v(x_1) = 1$$

$$v(x_2) = 1$$

$$v(x_3) = 0$$

$$v(x_4) = 1$$

$$k = 4$$



Jestliže je formule φ splnitelná, existuje ohodnocení v , při kterém má v každé klauzuli alespoň jeden literál hodnotu 1.

Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

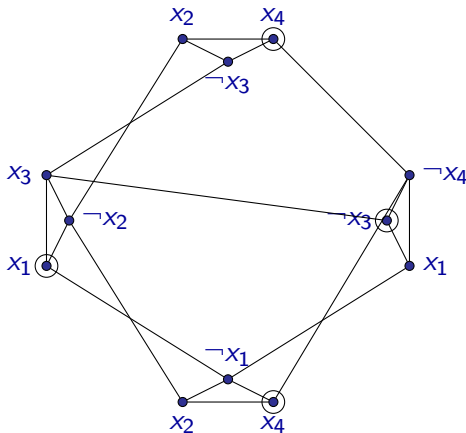
$$v(x_1) = 1$$

$$v(x_2) = 1$$

$$v(x_3) = 0$$

$$v(x_4) = 1$$

$$k = 4$$



Z každé klauzule vybereme jeden literál, který má při ohodnocení v hodnotu **1**, a do nezávislé množiny přidáme odpovídající vrchol.

Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

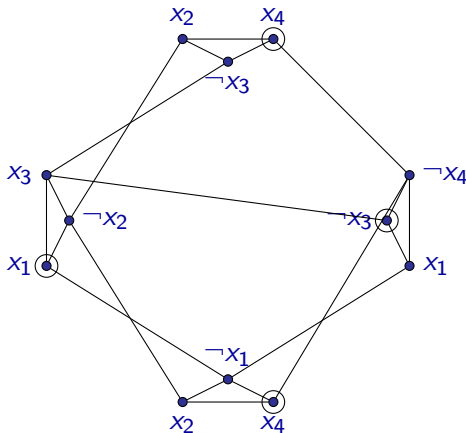
$$v(x_1) = 1$$

$$v(x_2) = 1$$

$$v(x_3) = 0$$

$$v(x_4) = 1$$

$$k = 4$$



Lehce ověříme, že vybrané vrcholy tvoří nezávislou množinu.

Vybrané vrcholy tvoří nezávislou množinu, protože:

- Z každé trojice vrcholů odpovídající jedné klauzuli byl vybrán jen jeden vrchol.
- Nemohly být současně vybrány vrcholy označené x_i a $\neg x_i$.
(Při daném ohodnocení v má hodnotu 1 jen jeden z nich.)

Na druhou stranu, pokud v grafu G existuje nezávislá množina velikosti k , musí určitě splňovat následující vlastnosti:

- Z každé trojice vrcholů odpovídající jedné klauzuli musí být vybrán nejvýše jeden vrchol.

Protože je ale klauzulí k a je vybráno k vrcholů, musí být z každé takové trojice vybrán právě jeden.

- Nemohly být současně vybrány vrcholy označené x_i a $\neg x_i$.

Ohodnocení tedy zvolíme podle vybraných vrcholů, protože z předchozího vyplývá, že nehrozí, že by neexistovalo.

(Zbylým proměnným přiřadíme libovolné hodnoty.)

Při daném ohodnocení má formule φ určitě hodnotu 1, neboť v každé klauzuli má hodnotu 1 alespoň jeden literál.

Popsaný algoritmus je určitě polynomiální:

Graf G a číslo k je možné zkonstruovat k formuli φ v čase $O(n^2)$, kde n je velikost formule φ .

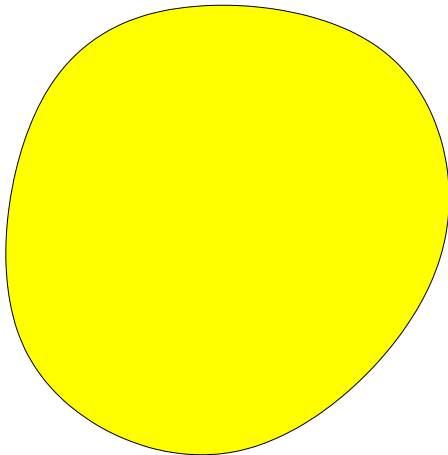
Navíc jsme viděli, že ve zkonstruovaném grafu G existuje nezávislá množina velikosti k právě tehdy, když formule φ je splnitelná.

Popsaný algoritmus tedy ukazuje, že problém 3-SAT je polynomiálně převeditelný na problém IS.

- **PTIME** — třída všech algoritmických problémů, pro které existuje (deterministický) algoritmus s polynomiální časovou složitostí
- **NPTIME** — třída všech algoritmických problémů, pro které existuje **nedeterministický** algoritmus s polynomiální časovou složitostí

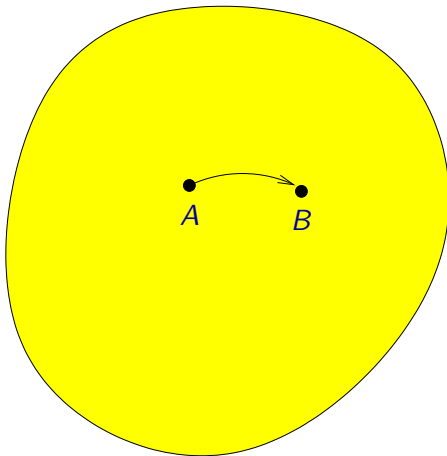
NP-úplné problémy

Vezměme si množinu všech možných rozhodovacích problémů.



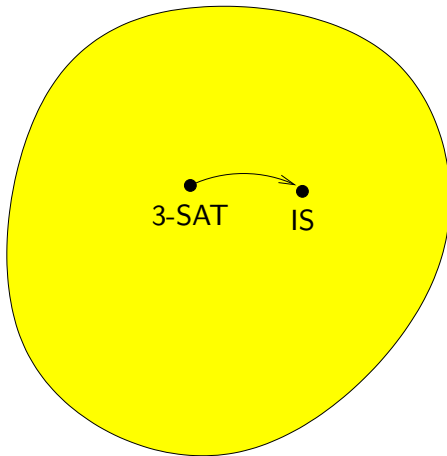
NP-úplné problémy

Šipkou si znázorníme, že problém A je polynomiálně převoditelný na problém B .



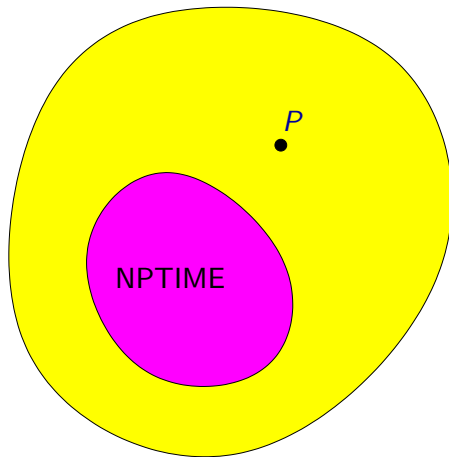
NP-úplné problémy

Například problém 3-SAT je polynomiálně převeditelný na problém IS.



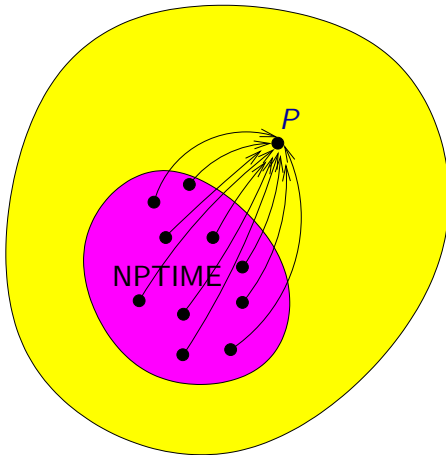
NP-úplné problémy

Vezměme si nyní třídu **NPTIME** a nějaký problém P .



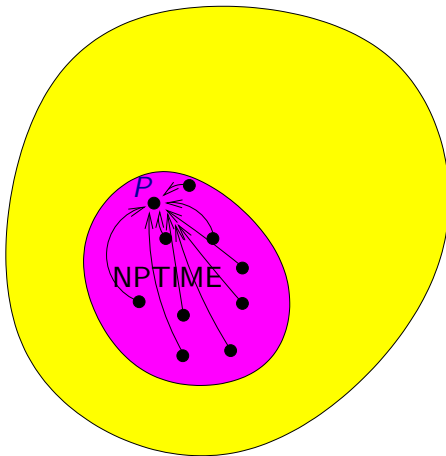
NP-úplné problémy

Problém P je **NP-těžký**, jestliže každý problém z NPTIME je polynomiálně převoditelný na P .



NP-úplné problémy

Problém P je **NP-úplný**, jestliže je NP-těžký a navíc sám patří do třídy **NPTIME**.



Pokud bychom pro nějaký NP-těžký problém P našli polynomiální algoritmus, získali bychom tím polynomiální algoritmus pro každý problém P' z **NPTIME**:

- Na vstup problému P' bychom nejprve aplikovali algoritmus realizující polynomiální převod z P' na P .
- Na vytvořenou instanci problému P bychom aplikovali polynomiální algoritmus řešící problém P a výsledek bychom vrátili jako odpověď pro danou instanci problému P' .

V takovém případě by tedy platilo **PTIME = NPTIME**, neboť pro každý problém z **NPTIME** by existoval polynomiální (deterministický) algoritmus.

Na druhou stranu, pokud existuje alespoň jeden problém z **NPTIME**, pro který neexistuje polynomiální algoritmus, tak z předchozího plyne, že pro žádný **NP**-těžký problém nemůže existovat polynomiální algoritmus.

Zda platí první nebo druhá možnost, je otevřený problém.

Není těžké si rozmyslet následující:

Pokud je problém A polynomiálně převeditelný na problém B a problém B je polynomiálně převeditelný na problém C , pak problém A je polynomiálně převeditelný na problém C .

Pokud tedy o nějakém problému P víme, že je NP-těžký a že P je polynomiálně převeditelný na problém P' , pak víme, že i problém P' je NP-těžký.

Věta

Problém SAT je NP-úplný.

Dá se ukázat, že SAT je polynomiálně převeditelný na 3-SAT a viděli jsme, že 3-SAT je polynomiálně převeditelný na IS.

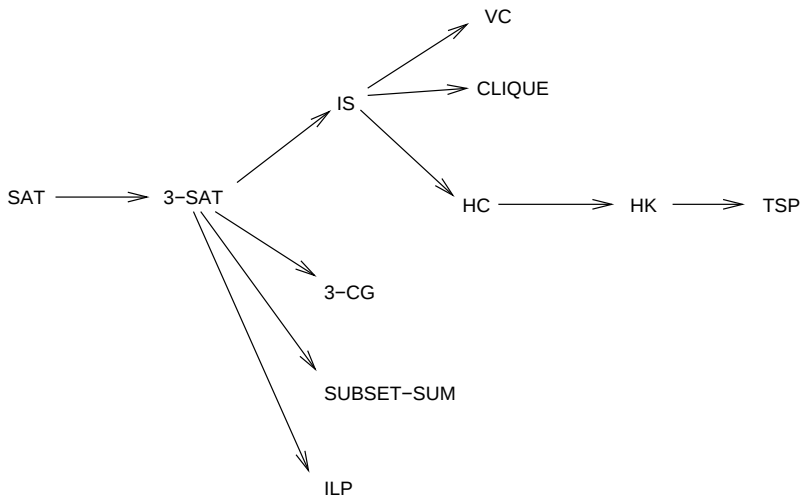
Z toho plyne, že problémy 3-SAT a IS jsou NP-těžké.

Není také těžké ukázat, že 3-SAT i IS patří do třídy NPTIME.

Problémy 3-SAT i IS jsou NP-úplné.

NP-úplné problémy

Polynomiálními převody z již známých NP-úplných problémů se dá ukázat NP-obtížnost celé řady různých dalších problémů:



Příklady některých NP-úplných problémů

Ze zatím uvedených problémů jsou NP-úplné následující tři problémy:

- SAT (splnitelnost booleovských formulí)
- 3-SAT
- IS — problém nezávislé množiny (independent set)

Na následujících slidech jsou uvedeny některé další NP-úplné problémy:

- CG — vrcholové barvení grafu (pozn.: je NP-úplný i ve speciálním případě, kdy máme právě 3 barvy)
- VC — vrcholové pokrytí grafu (vertex cover)
- CLIQUE — problém klíky
- HC — problém Hamiltonovského cyklu
- HK — problém Hamiltonovské kružnice
- TSP — problém obchodního cestujícího (traveling salesman problem)
- SUBSET-SUM
- ILP — celočíselné lineární programování (integer linear programming)

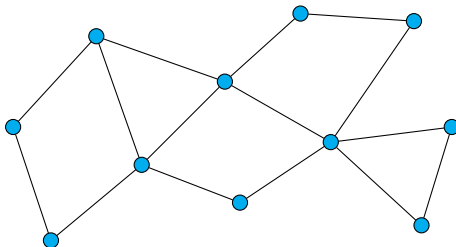
Barvení grafu

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: $k = 3$



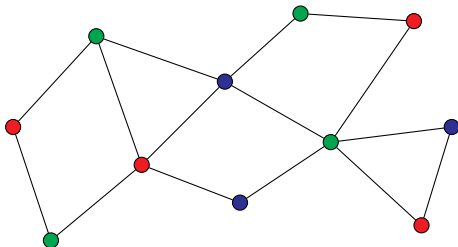
Barvení grafu

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: $k = 3$



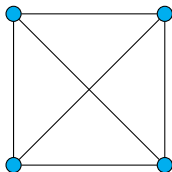
Odpověď: ANO

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: $k = 3$

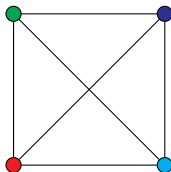


Barvení grafu

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: $k = 3$



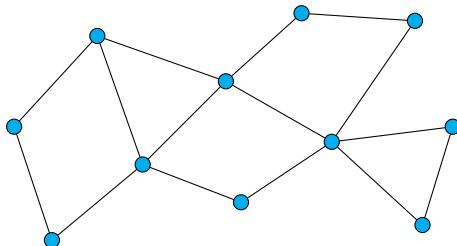
Odpověď: NE

VC – vrcholové pokrytí (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: $k = 6$



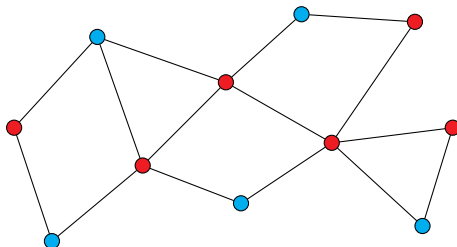
VC – Vrcholové pokrytí

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: $k = 6$



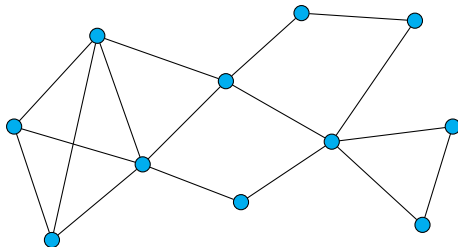
Odpověď: ANO

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: $k = 4$

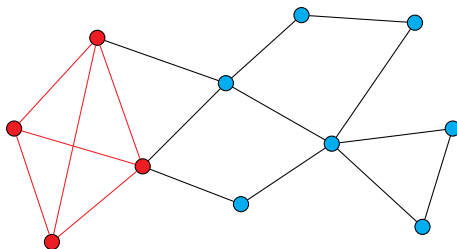


CLIQUE – problém kliky

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

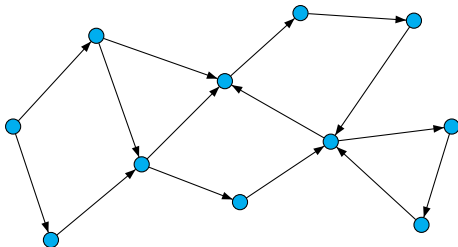
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G .

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



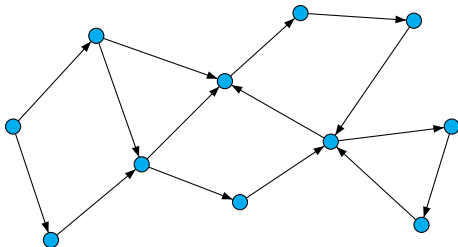
Hamiltonovský cyklus

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

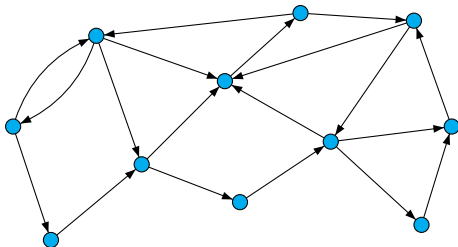
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G .

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



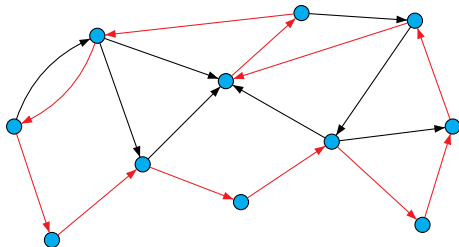
Hamiltonovský cyklus

HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf G .

Otázka: Existuje v grafu G Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



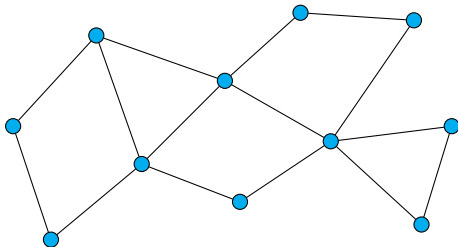
Odpověď: ANO

HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf G .

Otázka: Existuje v grafu G Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:

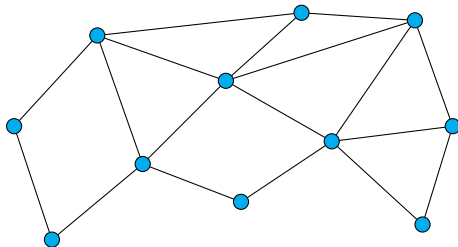


HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf G .

Otázka: Existuje v grafu G Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

Příklad:



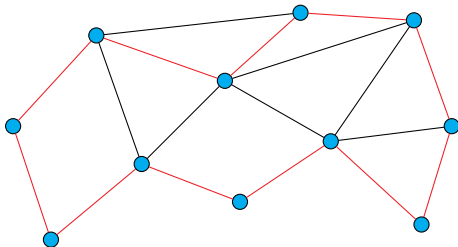
Hamiltonovská kružnice

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

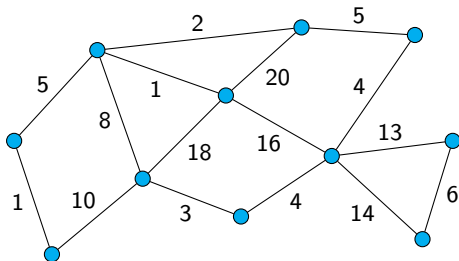
Problém obchodního cestujícího

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: $k = 70$



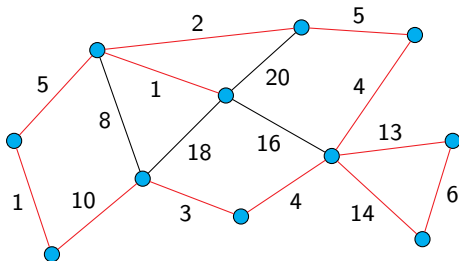
Problém obchodního cestujícího

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.

Problém SUBSET-SUM

Vstup: Sekvence přirozených čísel $a_1, a_2, \dots, a_n$ a přirozené číslo s .

Otázka: Existuje množina $I \subseteq \{1, 2, \dots, n\}$ 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.

Poznámka:

Pořadí čísel $a_1, a_2, \dots, a_n$ na vstupu není důležité.

Všimněte si však určitého rozdílu oproti tomu, kdybychom problém formulovali tak, že vstupem je množina $\{a_1, a_2, \dots, a_n\}$ a číslo s — v množině se čísla neopakují, zatímco v sekvenci se může totéž číslo vyskytnout vícekrát.

Problém SUBSET-SUM je speciálním případem **problému batohu** (knapsack problem):

Knapsack problem

Vstup: Sekvence dvojic přirozených čísel

$(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ a dvě přirozená čísla s a t .

Otázka: Existuje množina $I \subseteq \{1, 2, \dots, n\}$ taková, že $\sum_{i \in I} a_i \leq s$ a $\sum_{i \in I} b_i \geq t$?

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 $I \subseteq \{1, 2, \dots, n\}$, 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.

To, že SUBSET-SUM je speciálním případem problému batohu, vidíme z následující jednoduché konstrukce:

Řekněme, že $a_1, a_2, \dots, a_n, s_1$ je instance problému SUBSET-SUM. Je očividné, že pro instanci problému batohu, kde máme sekvenci $(a_1, a_1), (a_2, a_2), \dots, (a_n, a_n)$, $s = s_1$ a $t = s_1$, je odpověď stejná jako pro původní instanci 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.

Problém ILP (celočíslné lineární programování)

Vstup: Celočíselná matice A a celočíselný vektor b .

Otázka: Existuje celočíselný vektor x , takový že $Ax \leq b$?

Příklad instance problému:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \quad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

Ptáme se tedy, zda existuje celočíselné řešení následující soustavy nerovnic:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

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**.

Poznámka: Analogický problém, kdy se pro danou soustavu lineárních nerovnic ptáme, zda existuje její řešení v oboru **reálných** čísel, je možné řešit v polynomiálním čase.