Teoretické základy informatiky

26. dubna 2022

Obsah

1 První odstavec

Formální jazyky a jejich hierarchie. Regulární jazyky (definice, uzávěrové vlastnosti). Konečné automaty deterministické a nedeterministické. Regulární výrazy, automaty s epsilon-přechody. Minimalizace konečného deterministického automatu. Pumping lemma. Bezkontextové jazyky a jejich vlastnosti (uzávěrové vlastnosti, jednoznačnost). Zásobníkové automaty a jejich modifikace. Deterministické zásobníkové automaty. Deterministické bezkontextové jazyky.

1.1 Formální jazyky a jejich hierarchie

Zaveďme si pojem *formální jazyk nad množinou všech řetězců* Σ^*. Označme tento jazyk jako L. Pak platí tato tvrzení:

$$\begin{aligned} L &\subseteq \Sigma^* \text{ (každá podmnožina abecedy je jazykem)} \\ L &= \emptyset \text{ (prázdný jazyk)} \\ L &= \{\varepsilon\} \text{ (jazyk s prázdným řetězcem)} \\ &\vdots \end{aligned}$$

Pozor, obecně platí že prázdný jazyk $\neq$ jazyk s prázdným řetězcem.

Formálním jazykem obecně myslíme nějakou množinu *řetězců*, které navíc většinou mají určitou společnou vlastnost, jak uvidíme později.

1.1.1 Operace nad jazyky

1.1.2 Množinové

Množinové operace nad jazyky jsou prakticky totožné operacím na kterýchkoliv jiných množinách. Můžeme tedy použít množinový průnik, sjednocení, komplement (doplněk) nebo rozdíl.

1.1.3 Ostatní

- *Zřetězení* (produkt) množin. Vyjádřeme produkt takto:

  $$L_1 L_2 = \{xy | x \in L_1, y \in L_2\}$$

  Produkt množin není obecně komutativní, ale je asociativní, přičemž prázdná množina tuto operaci anihiluje. Uveďme si rovněž monoid $\langle 2^{\Sigma^*}, \circ, \{\varepsilon\}\rangle$.

- Mocnina jazyka. Mocninu vyjádříme takto:

  $$L^n = \begin{cases} \{\varepsilon\} & \text{pro } n = 0 \\ LL^{n-1} & \text{pro } n \geq 1 \end{cases}$$

- *Kleeneho* uzávěr neboli *iterace*. Tento uzávěr vyjádříme takto:

  $$L^* = \bigcup_{i=0}^{\infty} L^i$$

- *Pozitivní* uzávěr neboli pozitivní iterace. Tento uzávěr vyjádříme takto:

  $$L^+ = \bigcup_{i=1}^{\infty} L^i$$

  Všimněte si podobností mezi těmito dvěma uzávěry. Pozitivní uzávěr vynechává prázdný řetězec.

GRAMATIKY

Jak víme, tak jazyky mohou být *nekonečné* ve smyslu, že obsahují nekonečný počet slov. Nabízí se tedy otázka, jak tyto jazyky rozumně popsat, jak je reprezentovat resp. jak vytvořit *konečnou* sadu pravidel, jejichž aplikace by vedla k opětovné generaci původního jazyka.

Přepisovací pravidla

Pravidlem rozumíme zpravidla každou dvojici:

$$\langle x, y \rangle \in \Sigma^* \times \Sigma^*$$

Pak neformálně tvrdíme, že x *se přepisuje* na y. Nutno dodat, že předchozí zápis lze zapsat i například takto.

$$x \to y, \text{ kde symbol } \to \notin \Sigma \text{ můžeme prohlásit za tzv. metasymbol.}$$

Vlastnosti pravidel

- *Nezkracující* pravidlo je pravidlo, o kterém platí, že $|x| \leq |y|$. Tedy aplikaci tohoto pravidla na vstupní řetězec určitě nevznikne řetězec kratší, než-li jeho předloha.
- ε - pravidlo je pravidlo tvaru $x \to \varepsilon$.

Příklady pravidel

Příklad 1: Mějme zadání abecedy $\Sigma = \{a, b, c\}$. Pravidla s využitím této abecedy by mohla být například tato.

$$\begin{aligned} aa &\to bc \\ bb &\to abba \\ c &\to \varepsilon \end{aligned}$$

Příklad 2: Mějme další zadání abecedy $\Sigma = \{expr, +, \times\}$. Pravidla s využitím této abecedy by mohla být například takto.

$$\begin{aligned} expr &\to expr + expr \\ expr &\to expr \times expr \end{aligned}$$

Přímé odvozování řetězců pomocí pravidel

Uvažujme odvozovací pravidlo $x \to y$ nad abecedou Σ, pak řekneme, že řetězec v *je přímo odvozen* z řetězce u pomocí pravidla $x \to y$, pokud $\exists p, q \in \Sigma^*$ tak, že

$$\begin{aligned} u &= pxq \\ v &= pyq \end{aligned}$$

Značení předchozí operace je následující:

$$u \Rightarrow_{x \to y} v$$

Slovně bychom tento zápis vystihli jako „přímý přepis dle pravidla $x \to y$.“

Řetězec v vznikne přímým přepisem z u pomocí pravidel $P \subseteq \Sigma^* \times \Sigma^*$, pokud $\exists \pi \in P$ tak, že $u \Rightarrow_\pi v$. Značme $u \Rightarrow_P v$. P je množinou užitých pravidel. P i $\Rightarrow_p$ jsou binární relace na Σ^* a $P \subseteq \Rightarrow_p$, tedy „P je podmnožinou šipky p.“ Platí, že $x \to y \in P$ a $x \Rightarrow_{x \to y} y$. Můžeme tedy poněkud nadneseně říci, že množina pravidel P obsahuje jen podmnožinu všech možných pravidel, který je možno sestrojit nad danými terminálními a neterminálními symboly (o těch se dozvíme více informaci později).

Příklad 3: Mějme abecedu $\Sigma = \{a, b, c\}$ a soubor pravidel $P = \{aa \to bc, a \to cab, bb \to \varepsilon\}$. Pak by odvození v jednom kroku mohla vypadat například takto:

$$baaa \to bbca$$
$$bac \to bcabc$$

Definice 1: Definujme pojem *derivace*. Jedná se o posloupnost řetězců ve tvaru:

$$x_0, \ldots, x_k, \text{ kde } k \geq 0 \text{ a kde } \{x_0, \ldots, x_k\} \in \Sigma^*$$

se nazývá *P-derivace délky k*, pokud $x_{i-1} \Rightarrow_p x_i, \forall 1 \leq i \leq k$. Symbolicky totéž $x_0 \Rightarrow_p x_1 \Rightarrow_p \ldots \Rightarrow_p x_k$. Počet odvození tedy značí *délku* derivace.

Pokud pro $u, v \in \Sigma^* \quad \exists$ P-derivace $u = x_0 \ldots x_k = v$, pak říkáme, že v je odvozeno z u pomocí pravidel z P, což značíme například $u \Rightarrow_P^* v$, tímto je pochopitelně myšleno odvození ve více krocích. Platí, že $P \subseteq \Rightarrow_P \subseteq \Rightarrow_P^+$.

Příklad 4: Mějme abecedu $\Sigma = \{a, \ldots, z\}$ a pravidla stejná jako v příkladu **??**. Nyní odvozujeme například takto:

$$b\underline{aa}a, \underline{bb}ca, c\underline{a}, c\underline{cab}$$

Formální gramatiky

Mějme následující entity:

- Σ - abeceda terminálních symbolů (tyto symboly tvoří řetězce daného jazyka).
- N - abeceda neterminálních symbolů (tyto symboly se užívají k řízení průběhu odvozování).

Dodejme, že obě množiny by měly být neprázdné a konečné.

Definice 2: Odvozovací pravidlo $x \to y$ se nazývá *generativní*, pokud x obsahuje alespoň jeden neterminální symbol.

Definice 3: Mějme strukturu $G = \langle N, \Sigma, P, S \rangle$, kde N je abecedou neterminálních symbolů, Σ je abecedou terminálních symbolů, P je množinou odvozovacích pravidel a $S \in N$ je tzv. *počátečním* resp. *startovním* neterminálem. Pak tuto čtveřici nazveme *gramatikou*.

Poznámka 1: Pokud chceme vyjádřit, že z jednoho symbolu odvozujeme několik možných alternativ, tak to zapíšeme místo klasického dlouhé zápisu $y \to x_1, y \to x_2, \ldots$ pomocí zkrácené notace např. $y \to x_1 | x_2 | \ldots$

Příklad 5: Gramatika může vypadat třeba takto:

$$\begin{aligned} N &= \{\varepsilon, S, D, I\} \\ \Sigma &= \{0, \ldots, 9, +, -\} \\ P &= \{S \to -I | + I | I, I \to DI | D, D \to 0 | 1 | \ldots | 9\} \\ G &= \langle N, \Sigma, P, S \rangle \end{aligned}$$

Příklad 6: Nebo takto:

$$\begin{aligned} N &= \{S, X, Y\} \\ \Sigma &= \{a, b, c\} \\ P &= \{S \to XcYcX, X \to aX, X \to bX, X \to cX, X \to \varepsilon, Y \to abY, Y \to ab\} \\ G &= \langle N, \Sigma, P, S\rangle \end{aligned}$$

Definice 4: Každý řetězec $x \in (N \cup \Sigma)^*$, pro který platí $S \to^* x$, je *větná forma* gramatiky $G = \langle N, \Sigma, P, S\rangle$. Větná forma se nazývá *větou*, pokud $x \in \Sigma^*$.

Definice 5: Jazyk generovaný gramatikou definujme jako:

$$L(G) = \{x \in \Sigma^* | S \Rightarrow_G^* x\}$$

Vidíme tedy, že takový jazyk obsahuje *věty*, které lze odvodit ze startovacího neterminálu pomocí pravidel této gramatiky.

Příklad 7: Tento příklad čerpá gramatiku z příkladu **??**.

$$\begin{aligned} S &\Rightarrow_G^* abbccYcX \\ S &\Rightarrow_G^* Xcababababc \\ S &\Rightarrow_G^* cYcbaX \\ S &\Rightarrow_G^* abbccabca \\ S &\Rightarrow_G^* cababababc \end{aligned}$$

Definice 6: Gramatiky G_1 a G_2 jsou *ekvivalentní*, pokud generují stejný jazyk.

1.1.4 Hierarchie gramatik

- *Gramatiky typu 0* – jedná se o gramatiky bez omezení.
- *Gramatiky typu 1* – jedná se o tzv. *kontextové* nebo *kontextově závislé* gramatiky. Ty splňují následující omezení na tvar pravidel. Pro každé pravidlo gramatik tohoto typu platí, že:

  1. Buď je (pravidlo) ve tvaru $pAq \to pxq$, kde $p, q \in (\Sigma \cup N)^*, A \in N, x \in (\Sigma \cup N)^+$, kde p a q se nazývají levým resp. pravým *kontextem*.
  2. Nebo je (pravidlo) ve tvaru $S \to \varepsilon$, kde S je startovní terminál gramatiky, ale pouze za předpokladu, že S se nevyskytuje na pravé straně žádného pravidla.

  Zároveň platí pro každé pravidlo (s výjimkou pravidla $S \to \varepsilon$), že délka odvozeného řetězce je minimálně stejně velká jako délka vstupního řetězce. Gramatika tedy zároveň obsahuje pouze tzv. *nezkracující* pravidla.

- *Gramatiky typu 2* – jedná se o tzv. *bezkontextové* gramatiky, jenž obsahují pravidla ve tvaru:

  $$A \to x, \text{ kde } A \in N, x \in (\Sigma \cup N)^+$$

  Na levých stranách pravidel tedy očekáváme pouze neterminální symbol a na pravé straně očekáváme minimálně jeden symbol (s výjimkou ε-pravidla).

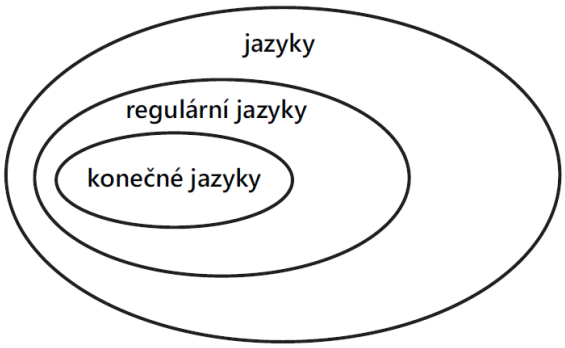
  Očividně platí, že je-li gramatika bezkontextová, pak je i kontextová, nebo $p = \varepsilon, q = \varepsilon$ jsou také kontexty, byť jsou „prázdné“.

Příklad 8: Mějme tuto gramatiku:

$$\begin{aligned} G &= \langle N, \Sigma, P, S \rangle \\ N &= \{A, S\} \\ \Sigma &= \{0, 1\} \\ P &= \{S \to 0A, A \to \varepsilon\} \end{aligned}$$

- *Gramatiky typu 3* – jedná se o tzv. *regulární* resp. *pravolineární* gramatiky, které obsahují pravidla ve třech následujících tvarech:
  1. $A \to bB,$ kde $A, B \in N, b \in \Sigma$
  2. $A \to a$
  3. $S \to \varepsilon$ (přičemž platí stejné podmínky, jako u gramatik typu 1)

Věta 1: Každý konečný jazyk je regulární.



Obrázek 1: Vychodilovo „vajíčko.“

Příklad 9:

$$\begin{aligned} N &= \{S\} \\ \Sigma &= \{a, b\} \\ P &= \{S \to aSb|\varepsilon\} \\ L(G) &= \{a^n b^n | \quad n \geq 0\} \end{aligned}$$

Máme tedy *bezkontextový* jazyk.

Příklad 10:

$$\begin{aligned} N &= \{S\} \\ \Sigma &= \{a, b\} \\ P &= \{S \to SS|aSb|bSa|\varepsilon\} \end{aligned}$$

$L(G)$ je *bezkontextový* jazyk.

Příklad 11:

$$\begin{aligned} N &= \{S, V\} \\ \Sigma &= \{p,), (, \Rightarrow, !\} \\ P &= \{S \to V | (S \Rightarrow S) | !S, V \Rightarrow pV | p\} \end{aligned}$$

$L(G)$ je jazyk všech výrokových formulí.

1.2 Regulární jazyky (definice, uzávěrové vlastnosti)

Gramatiky typu 3 – jedná se o tzv. *regulární* resp. *pravolineární* gramatiky, které obsahují pravidla ve třech následujících tvarech:

1. $A \to bB$, kde $A, B \in N, b \in \Sigma$
2. $A \to a$
3. $S \to \varepsilon$ (přičemž platí stejné podmínky, jako u gramatik typu 1)

Věta 2: Každý konečný jazyk je regulární.

1.2.1 Vztah regulárních jazyků a konečných automatů

Regulární jazyky jsou rozpoznatelné KDA (implikace zleva)

Věta 3: Pro každou regulární gramatiku $G = \langle N, \Sigma, P, S \rangle$ existuje konečný deterministický automat A tak, že jazyk generovaný gramatikou je totéž, jako jazyk rozpoznatelný automatem, tj. $L(G) = L(A)$

V případě, že $\varepsilon \in L(G)$, rozšíříme automat následovně, jednou ze tří možností:

1. Přidáme S do množiny koncových stavů.
2. Přidáme # mezi počáteční stavy.[1]
3. Zavedeme nový stav, který bude počáteční a zároveň koncový a nevedou z něj žádné přechody jinam.

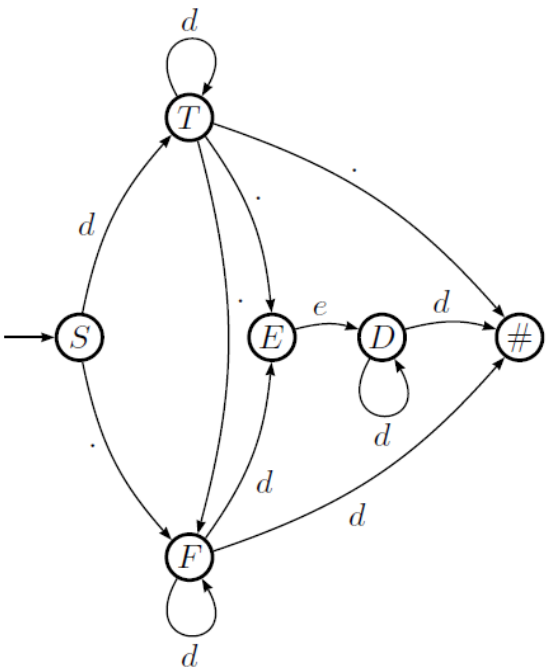
Poznámka 2: Nyní zbývá automat pouze determinizovat.

Příklad 12: Máme gramatiku G.

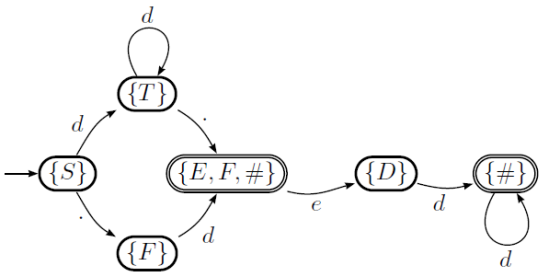
$$\begin{aligned} G &= \langle N, \Sigma, P, S \rangle \\ \Sigma &= \{e, d, .\} \\ P &= \{S \to .F | dT, T \to .E | .F | dT | ., D \to dD | d, E \to eD, F \to dE | dF | d\} \end{aligned}$$

Automat rozpoznávající jazyk, generovaný gramatikou G, bude vypadat následovně:

[1] Též nazývána jako „Konyho finta“.



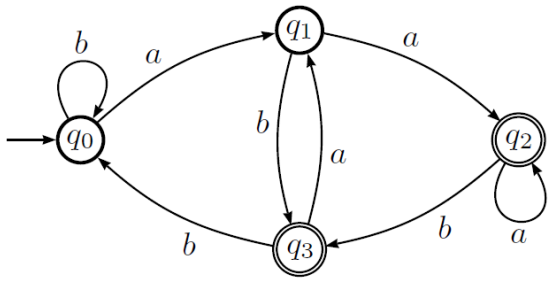
Když tento automat zdeterminizujeme, dostaneme následující automat:



1.3 Jazyky rozpoznatelné KDA jsou regulární (implikace zprava)

Věta 4: Pro každý konečný deterministický automat $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ existuje regulární gramatika G tak, že $L(A) = L(G)$.

Příklad 13: Mějme abecedu $\Sigma = \{a, b\}$ a automat zadaný diagramem:



Odvozovací pravidla gramatiky, generující tento jazyk budou:

$$\begin{aligned} q_0 &\rightarrow aq_1 \mid bq_0 \\ q_1 &\rightarrow aq_2 \mid a \mid bq_3 \mid b \\ q_2 &\rightarrow aq_2 \mid a \mid bq_3 \mid b \\ q_3 &\rightarrow aq_1 \mid bq_0 \end{aligned}$$

1.3.1 Regulární gramatiky

Co jsou to regulární gramatiky a jaké podmínky jejich odvozovací pravidla splňují již víme, ale můžeme si je ještě rozdělit na dva druhy, právě podle tvaru odvozovacích pravidel.

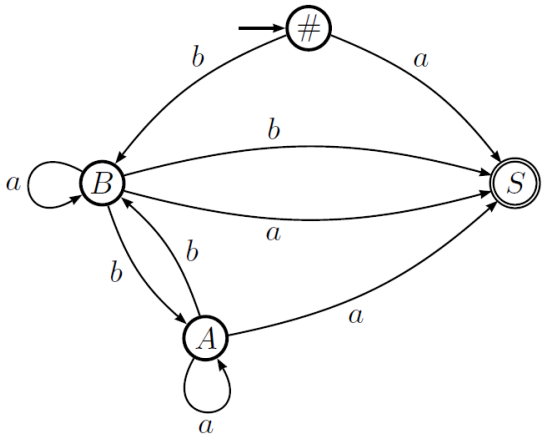
1. *Zprava regulární gramatiky:* Obsahují pravidla ve tvaru $A \rightarrow bB$, tedy neterminál je napravo od terminálního symbolu.
2. *Zleva regulární gramatiky:* Obsahují pravidla ve tvaru $A \rightarrow Bb$. Analogicky se neterminál nachází vlevo od terminálního symbolu.

Věta 5: Pro každou zleva regulární gramatiku $G = \langle N, \Sigma, P, S \rangle$ existuje konečný deterministický automat A tak, že *L(A)=L(G)*.

Příklad 14: Máme gramatiku G s následovně definovanými pravidly.

$$\begin{aligned} S &\rightarrow Aa|Ba|Bb|a \\ A &\rightarrow Aa|Bb \\ B &\rightarrow Ab|Ba|b \end{aligned}$$

Automat rozpoznávající jazyk generovaný touto gramatikou bude vypadat následovně:



Věta 6: Pro každý konečný deterministický automat A existuje zleva regulární gramatika taková, že *L(A)=L(G)*

Příklad 15: Vezmeme KDA z příkladu Odvozovací pravidla budou vypadat takto:

$$
\begin{aligned}
q_0 &\rightarrow q_0 b \mid b \mid q_3 b \\
q_1 &\rightarrow q_0 a \mid a \mid q_3 a \\
q_2 &\rightarrow q_1 a \mid q_2 a \\
q_3 &\rightarrow q_1 b \mid q_2 b \\
S &\rightarrow q_1 a \mid q_1 b \mid q_2 a \mid q_2 b
\end{aligned}
$$

Definice 7: Regulární jazyky jsou jazyky, generované zprava (zleva) regulárními gramatikami, tj. jsou rozpoznatelné konečnými ne/deterministickými automaty.

Poznámka 3: Pravidla zprava a zleva nelze míchat.

1.3.2 Uzávěrové vlastnosti regulárních jazyků

Základní uzávěrové vlastnosti

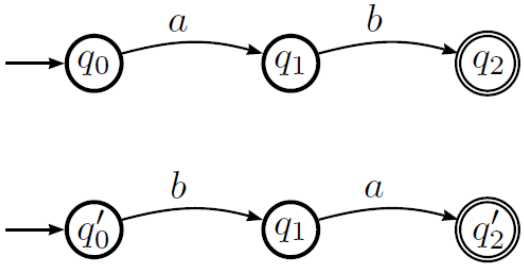
- *Komplement*

  Pokud je L regulární, pak je $\Sigma^* \backslash L$ regulární.

- *Sjednocení*

  Jsou-li L_1 a L_2 regulární, pak je $L_1 \cup L_2$ regulární.

  Příklad 16: Nyní si uvedeme protipříklad, co by se stalo, kdyby množiny stavů nebyly disjunktní.



  První automat přijímá řetězec ab, druhý automat přijímá řetězec ba, čili od jejich sjednocení očekáváme, že bude přijímat ab i ba. Jelikož množiny stavů nejsou disjunktní (q_1 je společný pro oba), sjednocení těchto automatů může stejně dobře přijímat i řetězce aa nebo bb, což je nežádoucí.

- *Průnik*

  S použitím De Morganových zákonů, dostáváme:

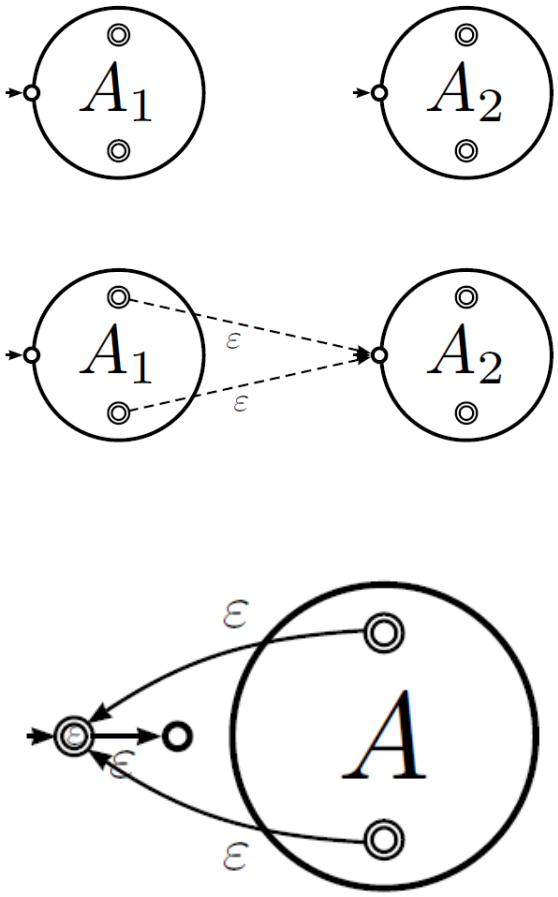
$$L_1 \cap L_2 = \Sigma^* \backslash (\Sigma^* \backslash L_1 \cup \Sigma^* \backslash L_2)$$

  tj. $L_1 \cap L_2$ je regulární.

- *Produkt (zřetězení)*

  $L_1 \cdot L_2$

  Předpokládáme existenci automatů A_1 a A_2 takových, že $L_1 = L(A_1)$ a $L_2 = L(A_2)$.



Sestavíme ε-KNA

$$A = \langle \Sigma, Q_1 \cup Q_2, \delta, \{q_{01}\}, F_2 \rangle$$

s přechodovou funkcí $\delta : (Q_1 \cup Q_2) \times (\Sigma \cup \{\varepsilon\}) \to 2^{Q_1 \times Q_2}$

$$\delta(q,a) = \begin{cases} \{\delta_1(q,a)\} & \text{pokud} \quad q \in Q_1 \\ \{\delta_2(q,a)\} & \text{pokud} \quad q \in Q_2 \end{cases}$$

$$\delta(q,\varepsilon) = \begin{cases} \{q_{02}\} & \text{pokud} \quad q \in F_1 \\ \emptyset & \text{pokud} \quad q \notin F_1 \end{cases}$$

- *Kleeneho uzávěr*

  Pokud je L regulární, pak je L^* regulární.

  Předpokládáme, že existuje automat A tak, že $L = L(A)$

  Poznámka 4: Automat rozpoznávající L^* musí mít možnost dostat se z koncového stavu zpět na začátek.

  Pro automat $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ je potřeba zavést nový počáteční stav $q_T \notin Q$. Konstruovaný ε-KNA bude vypadat následovně:

  $$A' = \langle \Sigma, Q \cup \{q_T\}, \delta', \{q_T\}, F \cup \{q_T\} \rangle$$

s přechodovou funkcí δ':

$$\begin{array}{lll} \delta'(q,a) = \{\delta(q,a)\} & \text{pokud} & q \in Q \\ \delta'(q,\varepsilon) = \emptyset & \text{pokud} & q \in Q \backslash F \\ \delta'(q,\varepsilon) = \{q_T\} & \text{pokud} & q \in F \\ \delta'(q_T,\varepsilon) = \{q_0\} & & \\ \delta'(q_T,a) = \emptyset & & a \in \Sigma \end{array}$$

1.4 Další uzávěrové vlastnosti

- *Množinový rozdíl*

  Jsou-li L_1 a L_2 regulární, pak $L_1 \backslash L_2 = L_1 \cap (\Sigma^* \backslash L_2)$ je regulární.

- *Kleeneho pozitivní uzávěr*

  Je-li L regulární, pak $L^+ = (L^* \backslash \{\varepsilon\}) \cup L$ je regulární.

- *N-tá mocnina jazyka*

  L^n ... plyne z uzavření na produkt

- *Jazyk reverzních řetězců*

  $L^R = \{w^R \mid w \in L\}$

- *Jazyk sufixů*

  $Sfx(L) = \bigcup_{w \in L} Sfx(w)$

  Poznámka 5: Všechny stavy jsou označeny za počáteční, aby měl automat možnost skočit do libovolné fáze výpočtu a tím "uhádnout"vynechané znaky řetězce, jehož sufix zkoumáme.

- *Jazyk prefixů*

$$\begin{aligned} Pfx(L) &= \bigcup_{w \in L} Pfx(w) \\ Pfx(L) &= (Sfx(L^R))^R \end{aligned}$$

  Důkaz plyne z uzavření na Sfx a reverzní řetězec.

- *Jazyk infixů*

$$Ifx(L) = Pfx(Sfx(L))$$

  Důkaz je taktéž zřejmý.

1.5 Konečné automaty deterministické a nedeterministické

1.5.1 Konečné deterministické automaty

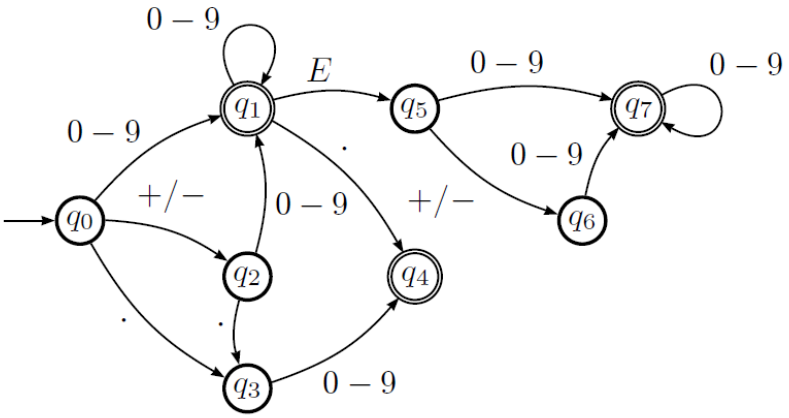
KDA realizují analytický (resp. výpočetní) formalismus a představují jakési „jednoduché počítače“. KDA a jejich činnost jsou charakterizovány několika prvky:

1. Vstupem jsou řetězce nad určitou abecedou, která se obvvykle značí Σ.
2. Řídící jednotka automatu se skládá z konečně mnoha stavů (aby byla reprezentovatelná v počítačích).
3. Počátek činnosti automatu představuje okamžik, kdy na vstupu automatu je celý vstupní řetězec a řídící jednotka je v iniciálním stavu.

4. Automat vykonává svou primární činnost tím, že přečte ze vstupu aktuální symbol a na základě stavu, ve kterém se právě nachází se řídící jednotka přepne do jiného stavu a odebere onen přečtený symbol ze vstupního řetězce.

5. Konec činnosti automatu nastává v případě, že na vstupu již není co číst. Byl tedy zpracován celý vstupní řetězec. V tomto okamžiku automat nachází v určitém stavu, podle typu stavu následně řekneme, že automat řetězec přijímá nebo zamítá. Existují tedy přijímací (neboli koncové) stavy a nepřijímací (neboli nekoncové) stavy.

Příklad 17: Nyní si ukažme, jak vypadá typický KDA.



Dvojitě kroužkovaný stav označuje stav koncový. Šipky mezi jednotlivými stavy označují přechody, přičemž popisek u jednotlivých přechodů indikuji symbol, který se čte, pokud je tento přechod využit. Automat na obrázku by přijímal například slovo $13.6E + 2$, obecně jakékoliv číslo kodované v tradiční fixní a pohyblivé notaci s desetinnou tečkou.

Definice 8: Konečný deterministický automat s úplnou přechodovou funkcí je struktura:

$$A = \langle \Sigma, Q, \delta, q_0, F \rangle$$

Nyní si popišme jednotlivé členy automatu:

$$\begin{aligned} \Sigma &= \text{vstupní abeceda} \\ Q &= \text{konečná množina stavů} \\ q_0 \in Q &= \text{počáteční stav} \\ F \subseteq Q &= \text{množina koncových stavů} \\ \delta &= \text{zobrazení } \delta: \quad Q \times \Sigma \to Q, \text{ neboli přechodová funkce} \end{aligned}$$

U přechodové funkce δ uvažujeme zápis například $\delta(r, a) = q$, který čteme: „Při vstupním symbolu $a \in \Sigma$ a aktuálním stavu r přejde automat do nového stavu q.“

Předpokládáme, že Q je konečná a δ je zobrazení.

Příklad 18: Automat s nadefinovanými přechody dle δ by mohl vypadat například takto:

$$\begin{aligned} A &= \langle \Sigma, Q, \delta, q_0, F \rangle \\ \Sigma &= \{a, b, c\} \\ Q &= \{q_0, q_1, q_2, q_3, q_4\} \\ F &= \{q_4\} \\ \delta &= \{< q_0, a, q_0 >, < q_0, b, q_0 >, < q_0, c, q_1 > \\ &\quad < q_1, a, q_2 >, < q_1, b, q_0 >, < q_1, c, q_1 > \\ &\quad < q_2, a, q_0 >, < q_2, b, q_3 >, < q_2, c, q_1 > \\ &\quad < q_3, a, q_2 >, < q_4, b, q_0 >, < q_3, c, q_4 > \\ &\quad < q_4, a, q_4 >, < q_4, b, q_4 >, < q_4, c, q_4 >\} \end{aligned}$$

1.5.2 Reprezentace KDA

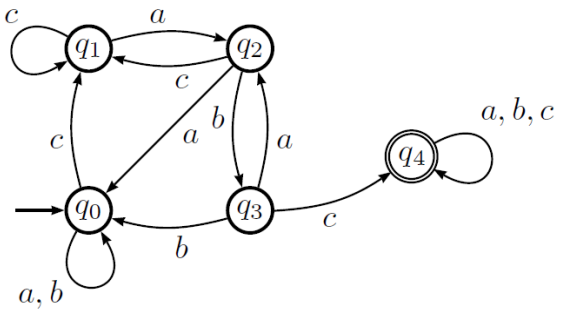
Reprezentace přechodovou tabulkou

Řádky tabulky představují stavy a sloupce jednotlivé symboly vstupní abecedy. Znak * označuje, že stav je koncový.

	a	b	c
q_0	q_0	q_0	q_1
q_1	q_2	q_0	q_1
q_2	q_0	q_3	q_1
q_3	q_2	q_0	q_4
q_4^*	q_4	q_4	q_4

Tabulka 1: Přechodová tabulka pro KDA

1.5.3 Reprezentace přechodovým diagramem



Obrázek 2: Přechodový diagram pro KDA

1.5.4 Konfigurace a výpočet KDA

Konfgurace jednoznačně určuje aktuální fázi výpočtu a jsou to dvojice $\langle q, w \rangle, q \in Q, w \in \Sigma^*$, následně samotná konfigurace $Q \times \Sigma^*$.

Počáteční konfigurace zahrnuje obvykle počáteční stav q_0, tedy například $\langle q_0, w\rangle$.

Koncová konfigurace je konfigurací, kdy jsme přečetli celý vstupní řetězec, tedy $\langle q, \varepsilon\rangle$. Nicméně je třeba rozlišit dva stavy:

1. Koncová přijímací konfigurace, pro případ, kdy $q \in F$.
2. Koncová zamítací konfigurace, pro případ, kdy $q \notin F$.

Definice 9: Mějme automat $A = \langle \Sigma, Q, \delta, q_0, F\rangle$ a řetězec $w \in \Sigma^*$. Pak posloupnost konfigurací r_i, w_i pro $i \in \{0, 1, \ldots, n\}$ splňující podmínky:

1. r_0 je počáteční stav automatu A.
2. $w_0 = w$
3. $w_n = \varepsilon$
4. $w_i = a_i w_{i+1}$ a $\delta(r_i, a_i) = r_{i+1}$ pro každé $i \in \{0, 1, \ldots, n-1\}$

nazveme *výpočet automatu A délky n pro řetězec w.*

Popíšme si nyní jeden krok výpočtu, tj. obecně:

$$\langle r_i, w_i\rangle = \langle r_{i+1}, w_{i+1}\rangle$$

tj. w_{i+1} vzniknlo z w_i odebráním prvního symbolu, který si označme a_i. Do stavu r_{i+1} jsme se dostali ze stavu r_i při vstupním symbolu a_i.

Příklad 19: Typický automatový výpočet vypadá například takto:

$$\begin{gathered}\langle q_0; accbcabca\rangle\\ \langle q_0; ccbcabca\rangle\\ \langle q_1; cbcabca\rangle\\ \langle q_1; bcabca\rangle\\ \langle q_0; cabca\rangle\\ \langle q_1; abca\rangle\\ \langle q_2; bca\rangle\\ \langle q_3; ca\rangle\\ \langle q_4; c\rangle\\ \langle q_4; \varepsilon\rangle\end{gathered}$$

Věta 7: KDA A má pro řetězec délky n jednoznačný výpočet rovněž délky n.

1.5.5 Rozšířená přechodová funkce pro KDA

Rozšířená přechodová funkce má rekurzivní předpis:

$$\delta^* : \quad Q \times \Sigma^* \to Q$$

$$\delta^*(q, u) = \begin{cases} q & \text{pokud } u = \varepsilon \\ \delta^*(\delta(q, a), v) & \text{pokud } u = av, a \in \Sigma, \end{cases}$$

Věta 8: Pro každé $u, v \in \Sigma^*$ platí, že $\delta^*(q, uv) = \delta^*(\delta^*(q, u), v)$

Poznámka 6: Automat A přijímá řetězec w, pokud $\delta^*(q_0, w) \in F$.

Věta 9: Řetězec w je přijat automatem, právě když existuje přijímací výpočet pro w.

Jazyk rozpoznaný konečným deterministickým automatem lze formulovat jako:

$$L(A) = \{w \in \Sigma^* | \quad \delta^*(q_0, w) \in F\}$$

1.5.6 KDA s neúplnou přechodovou funkcí

Mějme KDA $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ a upřesněme δ, které se liší.

δ je neúplná přechodová funkce tvaru $\delta \subseteq Q \times \Sigma \times Q$, přičemž také platí, že pokud $\langle q, a, r_1 \rangle \in \delta$ a $\langle q, a, r_2 \rangle \in \delta$, pak $r_1 = r_2$ a $\delta : \quad Q \times \Sigma \to Q$.

Z toho plynou dvě situace:

1. $\delta(q, a)$ je definována, $\langle q, a, r \rangle \in \delta$ pro nějaké r takové, že $\delta(q, a) = r$.
2. $\delta(q, a)$ není definováno, pokud $\langle q, a, r \rangle \notin \delta$ pro žádná r.

   V grafu se toto projeví tak, že ze stavu q_i nevede žádná hrana pro symbol a_i.

   Výpočet se liší v bodu **??** $w_i = a_i w_{i+1}$ a $\delta(r_i, a_i)$ je definována a $\delta(r_i, a_i) = r_{i+1}$ pro každé $i \in \{0, 1, \ldots, n\}$.

Řetězec w je přijat automatem A, právě tehdy, když existuje výpočet A pro w, kteý je přijímající.

Následně jazyk rozpoznaný automatem A je $L(A) = \{w \in \Sigma^* | \quad \delta^*(q_0, w) \in F\}$.

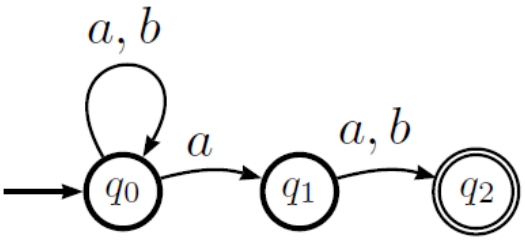
Věta 10: Ke každému KDA s neúplnou přechodovu funkcí δ existuje KDA s úplnou přechodovu funkcí δ, který rozpoznává stejný jazyk.

1.6 Implementace KDA

1. δ jako dvourozměrné pole - „ďábelsky rychlé“, paměťově náročné
2. δ jako seznam/hashovací tabulka - vhodné když $|\delta| <<< |Q \times \Sigma|$
3. δ jako orientovaný graf

1.6.1 Konečné nedeterministické automaty

Příklad 20: Mějme automat:



Vstupní řetězce: *abba* (nepřijat), *baba* (nepřijat), *baab* (přijat), *bbaa* (přijat).

V případě řetězce *baab* máme dokonce 3 možnosti výpočtu:

1. $\langle q_0, baab\rangle, \langle q_0, aab\rangle, \langle q_0, ab\rangle, \langle q_0, b\rangle, \langle q_0, \varepsilon\rangle$ – končí neúspěchem.
2. $\langle q_0, baab\rangle, \langle q_0, aab\rangle, \langle q_1, ab\rangle, \langle q_2, b\rangle$ – končí neúspěchem.
3. $\langle q_0, baab\rangle, \langle q_0, aab\rangle, \langle q_0, ab\rangle, \langle q_1, b\rangle, \langle q_2, \varepsilon\rangle$ – končí úspěchem.

Předchozí zápisy můžeme pojmenovat také jako „nedeterministický výpočet."

Jiným zápisem téhož může být také ten následující.

$$\langle \{q_0\}, baab\rangle, \langle \{q_0\}, aab\rangle, \langle \{q_0, q_1\}, ab\rangle, \langle \{q_0, q_1, q_2\}, b\rangle, \langle \{q_0, q_2\}, \varepsilon\rangle$$

Definice 10: Strukturu $A = \langle \Sigma, Q, \delta, I, F\rangle$ nazvěme *konečným nedeterministickým automatem* nad abecedou Σ. Pro tuto strukturu následně platí tato tvrzení:

- Σ, Q a F jsou stejné jako u konečného deterministického automatu.
- $I \subseteq Q$ označuje množinu počátečních stavů, která by měla být obecně neprázdná.
- δ označuje přechodovou funkci ve tvaru $\delta : Q \times \Sigma \to 2^Q$, tedy $\delta(q, a) = \{r_1, \ldots, r_k\}$. Totéž slovně: „Automat může při stavu q při symbolu a přejít do kteréhokoliv stavu z $\{r_1, \ldots, r_k\}$."

Příklad 21:

$$\begin{aligned} \Sigma &= \{a, b\} \\ P &= \{q_0, q_1, q_2, q_3\} \\ I &= \{q_0, q_3\} \\ F &= \{q_2\} \end{aligned}$$

Následně přechodová funkce:

$$\begin{aligned} \delta = \quad & \{\langle q_0, a, \{q_0, q_1\}\rangle, \langle q_0, b, \{q_0\}\rangle, \langle q_1, a, \{q_2\}\rangle, \langle q_1, b, \{q_2\}\rangle, \\ & \langle q_2, a, \emptyset\rangle, \langle q_2, b, \emptyset\rangle, \langle q_3, a, \emptyset\rangle, \langle q_3, b, \{q_1\}\rangle\} \end{aligned}$$

1.6.2 Reprezentace KNA

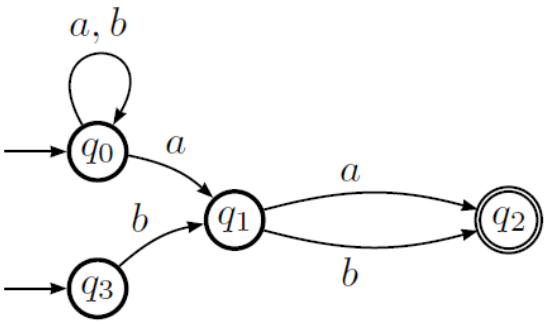
Předchozí příklad číslo **??** lze reprezentovat několika způsoby:

1. *Přechodová tabulka*, která ve svém těle obsahuje množiny stavů.

	a	b
$\to q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2*	$\emptyset$	$\emptyset$
$\to q_3$	$\emptyset$	$\{q_1\}$

Tabulka 2: Přechodová tabulka s množinami stavů

2. *Diagram*, který automat demonstruje v grafičtější podobě.



1.6.3 Nedeterministický výpočet

Nyní si popišme *nedeterministický výpočet*, který je definován následujícími věcmi:

- Konfigurace, což je dvojice ve tvaru $\langle stav, řetězec \rangle$.
- Počáteční konfigurace ve tvaru $\langle q, w \rangle$ kde $q \in I$.
- Koncová konfigurace ve tvaru $\langle q, \varepsilon \rangle$.
- Koncová přijímací konfigurace $\langle q, \varepsilon \rangle$ kde $q \in F$.

Definice 11: Mějme $A = \langle \Sigma, Q, \delta, I, F \rangle$ a $w \in \Sigma^*$. Pak posloupnost konfigurací $\langle r_i, w_i \rangle$ pro $i = \{0, \ldots, n\}$ splňující podmínky:

$$r_0 \in I \tag{1}$$
$$w_0 = w \tag{2}$$
$$w_n = \varepsilon \tag{3}$$
$$w_i = a_i w_{i+1} \text{ a } r_{i+1} \in \delta(r_i, a_i) \text{ pro } i = \{0, \ldots, n-1\} \tag{4}$$

nazveme *nedeterministický výpočet*.

1.6.4 Rozšířená přechodová funkce

Definice 12: Rozšířená přechodová funkce má tvar:

$$\delta^* : 2^Q \times \Sigma^* \to 2^Q$$
$$\delta^*(R, w) = \left\{ \begin{array}{ll} R & \text{pokud } w = \varepsilon \\ \delta^*(\bigcup\limits_{q \in R} \delta(q, a), u) & \text{pokud } w = au, \text{ kde } a \in \Sigma, u \in \Sigma^* \end{array} \right\}$$

Věta 11: Platí $\delta^*(R, w) = \delta^*(\delta^*(R, u), v), \forall R \subseteq Q, uv \in \Sigma^*$.

Věta 12: Platí následující tvrzení:

$$\delta^*(\bigcup_{i=1}^{k} R_i, w) = \bigcup_{i=1}^{k} \delta^*(R_i, w) \text{ pro každé} R_i \subseteq Q, w \in \Sigma^*$$

1.6.5 Řetězce přijímané KNA

KNA A přijímá řetězec w, pokud $\delta^*(I, w) \cap F \neq \emptyset$. Navíc jazyk, přijímaný KNA A si definujme jako $L(A) = \{w \in \Sigma^* | \delta^*(I, w) \cap F \neq \emptyset\}$.

Věta 13: Platí, že $w \in L(a)$ právě tehdy, když KNA A má přijímací výpočet pro w.

1.7 Regulární výrazy

Definice 13: Nechť je dána $\Sigma = \{a_1, \ldots, a_k\}$. Pak regulární výraz na Σ je:

1. $\emptyset$
2. ε
3. symboly $a_1, \ldots, a_k$
4. pokud R_1, R_2 jsou RV, pak $(R_1|R_2)$ je RV
5. pokud R_1, R_2 jsou RV, pak (R_1R_2) je RV
6. pokud R je RV, pak R^* je RV

Příklad 22: Podívejme se na následující výrazy a rozhodněme, zda-li jsou regulární:

- $((ab)c)^*$, $((a|b)c)^*$ – jsou RV
- $a^*b)$, $a||b$ – nejsou RV

Poznámka 7: Priorita $|, \circ, *$ je stejná jako (po řadě) $+, \cdot, ^{-1}$ v aritmetických výrazech.

1.7.1 Jazyky generované regulárními výrazy

Definice 14: Nechť R je RV nad Σ. Pak $L(R) \subseteq \Sigma^*$. Pak také platí:

1. $L(R) = \emptyset$, pokud $R = \emptyset$.
2. $L(R) = \{\varepsilon\}$, pokud $R = \varepsilon$.
3. $L(R) = \{a_i| \quad \text{pokud } R = a_i\}$.
4. $L(R) = L(R_1) \cup L(R_2)$, pokud $R = \{R_1|R_2\}$.
5. $L(R) = L(R_1) \circ L(R_2)$, pokud $R = \{R_1 \circ R_2\}$.
6. $L(R) = L(R_1)^*$, pokud $R = R_1^*$.

Věta 14: Každý regulární výraz lze převést na konečný automat.

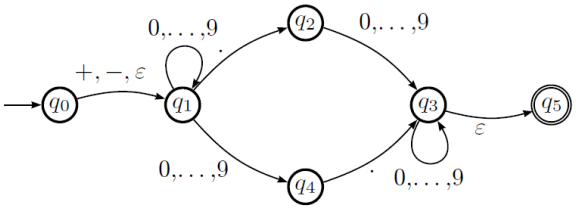
Věta 15: Každý jazyk generovaný regulárním výrazem je regulární.

Věta 16: Každý regulární jazyk lze generovat regulárním výrazem.

1.8 Automaty s epsilon-přechody

Značíme ε-KNA.

Příklad 23: Zde je jeden motivační příklad na úvod.



Definice 15: Nedeterministický konečný automat s ε-přechody je struktura $\langle \Sigma, Q, \delta, I, F \rangle$, kde Σ, Q, δ, I, F mají stejný význam jako u KNA. δ je přechodová funkce $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.

Fakt $\delta(q, a) = \{r_1, \ldots, r_k\}$ čteme: „automat A při čtení symbolu a přejde ze stavu q do některého ze stavů $r_1, \ldots, r_k$.“

Fakt $\delta(q, \varepsilon) = \{r_1, \ldots, r_k\}$ čteme: „automat A přejde samovolně ze stavu q do některého ze stavů $r_1, \ldots, r_k$.“

1.9 Reprezentace ε-KNA

1. *Přechodová tabulka*, vypadá stejně jako u KNA s tím, že přidáme jeden sloupec, ve kterém budeme zaznamenávat ε-přechody.

   Takto bude vypadat předchozí příklad číslo **??**, reprezentovaný pomocí tabulky.

	+,-	.	$0, \ldots, 9$	ε
$\to q_0$	$\{q_1\}$	$\emptyset$	$\emptyset$	$\{q_1\}$
q_1	$\{q_2\}$	$\emptyset$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\emptyset$	$\emptyset$	$\{q_3\}$	$\{q_5\}$
q_4	$\emptyset$	$\{q_3\}$	$\emptyset$	$\emptyset$
q_5^*	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Tabulka 3: Přechodová tabulka pro **??**. příklad

2. *Přechodový diagram*, u kterého mohou být některé hrany ohodnoceny ε. Viz příklad **??**.

1.9.1 Nedeterministický výpočet

Pojem *konfigurace* pro nás zůstává stejný, jedná se stále o dvojici $\langle$*stav,řetězec*$\rangle$.

Definice 16: Mějme automat $A = \langle \Sigma, Q, \delta, I, F \rangle$ a řetězec $w \in \Sigma^*$. Posloupnost konfigurací $\langle r_i, w_i \rangle$ pro $i = 0, \ldots, n$ splňující podmínky:

1. $r_0 \in I, w_0 = w$
2. $w_n = \varepsilon$
3. pro každé $i = 0, \ldots, n$
   (a) $w_i = a w_{i+1}, r_{i+1} \in \delta(r_i, a)$
   (b) $w_i = w_{i+1}, r_{i+i} \in \delta(r_i, \varepsilon)$

1.9.2 ε-uzávěry množin stavů

Je dána množina stavů $R \subseteq Q$, jež se nazývá ε-uzavřená, pokud $\delta(q, \varepsilon) \subseteq R$ pro každý stav $q \in R$.

Příklad 24: Lze ukázat na našem příkladě **??**.
$\{q_0\}$ není ε-uzavřená protože $\delta(q_0, \varepsilon) = \{q_1\} \not\subseteq \{q_0\}$
$\{q_0, q_1\}$ je ε-uzavřená

1.9.3 Tvorba ε-uzávěru

1. Mějme ε-uzávěr R.
2. Prohlašme, že $E_0 = R$ a následně pro $i \geq 1$ tvrďme, že:
   - $E_i = E_{i-1} \cup \{\delta(q, \varepsilon) | q \in E_{i-1}\}$
   - $E_A(R) = \bigcup_{i=0}^{\infty} E_i$
   - $E_0 \subseteq E_1 \subseteq E_2 \subseteq \ldots \subseteq E_A(R)$

Poznámka 8: Vzhledem ke konečnosti množiny $E_A(R)$ musí existovat index k, pro který platí $E_k = E_{k+1} = \ldots = E_A(R)$

Můžeme pozorovat, že $E_A(R)$ není jen ε-uzavřená, ale je také *nejmenší* ε-uzavřená. Z toho můžeme vyvodit, že

$$E_A : 2^Q \to 2^Q$$

je *uzávěrový* operátor.

1.9.4 Rozšířená přechodová funkce

Pro automat $A = \langle \Sigma, Q, \delta, I, F \rangle$ je rozšířená přechodová funkce definovaná jako
$\delta^* : 2^Q \times \Sigma^* \to 2^Q$

$$\delta^*(R, w) = \begin{cases} E_A(R) & \text{pokud} \quad w = \varepsilon \\ \delta^*(E_A(\bigcup_{q \in E_A(R)} \delta(q, a), u)) & \text{pokud} \quad w = au, a \in \Sigma, u \in \Sigma^* \end{cases}$$

Věta 17: Pro libovolné množiny $R_i \subseteq Q \quad (i = 1, \ldots, k)$ platí:
$\bigcup_{i=1}^{k} E_A(R_i) = E_A(\bigcup_{i=1}^{k} R_i)$

1.9.5 Přijímaný řetězec ε-KNA

Nechť $A = \langle \Sigma, Q, \delta, I, F \rangle$ je ε-KNA. Řetězec $w \in \Sigma^*$ nazýváme řetězec *přijímaný* A, pokud
$\delta^*(I, w) \cap F \neq \emptyset$
jinak w nazýváme řetězec *zamítaný* A.
Jazyk přijímaný A: $L(A) = \{w \in \Sigma^* | \delta^*(I, w) \cap F \neq \emptyset\}$

Ekvivalence s KDA

Věta 18: Ke každému ε-KNA $A = \langle \Sigma, Q, \delta, I, F \rangle$ existuje KNA $A^S = \langle \Sigma, Q, \delta^S, I^S, F \rangle$ takový, že $L(A) = L(A^S)$.

1.10 Minimalizace konečného deterministického automatu

Poznámka 9: Pro regulární jazyk L existuje více, než jeden automat A tak, že $L = L(A)$ a navíc můžeme mít A_1, A_2 tak, že $L(A_1) = L(A_2)$, ale $|Q_1| < |Q_2|$.

1.10.1 Zprava invariantní ekvivalence

Definice 17: Předpokládejme, že máme $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ a relaci ekvivalence $\Theta \subseteq Q \times Q$ nazveme *zprava invariantní ekvivalencí* vzhledem k δ, pokud platí, že $\langle q, r \rangle \in \Theta$ a $a \in \Sigma$, pak $\langle \delta(q, a), \delta(r, a) \rangle \in \Theta$.

Pravá invariance reprezentuje přirozenou vlastnost, kterou musí mít relace nerozlišitelnosti stavů.

Mezními případy invariantních relací zprava jsou:

1. identita $\Theta = \{\langle q, q \rangle | \quad q \in Q\}$
2. $\Theta = Q \times Q$

1.10.2 Faktorizace automatu

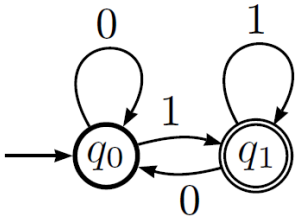
Definice 18: Mějme KDA $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ a zprava invariantní ekvivalenci Θ vzhledem k δ, pak zavedeme $A/\theta = \langle \Sigma, Q/\Theta, \delta^{A/\Theta}, [q_0]_\Theta, F^{A/\Theta} \rangle$, kde

$$\delta^{A/\Theta} = ([q]_\Theta, a) = [\delta(q, a)]_\Theta \text{ a } F^{A/\Theta} = \{[q]_\Theta | \quad q \in F\}$$

Věta 19: Automat A je dobře definovaný KDA.

Obecně $L(A/\Theta) \neq L(A)$. Snažíme se najít co největší Θ, tak aby tato rovnost platila.

Příklad 25: $\Theta = Q \times Q$



Věta 20: Platí, že $(\delta^{A/\Theta})^*([q]_\Theta, x) = [\delta^*(q, x)]_\Theta$ pro každý $x \in \Sigma^*$.

Definice 19: Mějme KDA $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ s úplnou přechodovou funkcí. Pro stavy $q, r \in Q$ položme $q \equiv_A r$, pokud pro každý řetězec $x \in \Sigma*$ platí, že $\delta^*(q, x) \in F$, právě když $\delta^*(r, x) \in F$.

Věta 21: $\equiv_A$ je zprava invariantní operace.

Věta 22: Pro $A/\equiv_A$ a stav $q \in Q$ platí, že $q \in F$ právě, když $[q]_{\equiv_A} \in F^{A/\equiv_A}$.

Důsledek 1: KDA A nazveme redukovaný, pokud je $\equiv_A$ identita.

Věta 23: KDA $A/\equiv_A$ je redukovaný.

Věta 24: $L(A) = L(A/\equiv_A)$, což plyne užitím vět **??** a **??**.

Důsledek 2: Obecně platí, že $L(A) \subseteq L(A/\Theta)$.

Věta 25: Pokud je každý stav automatu A dosažitelný, pak má $A/\equiv_A$ také každý stav dosažitelný.

1.10.3 Algoritmus pro hledání redukovaného automatu

1. Označíme $A/\equiv_A$ jako A^R, konstruujeme posloupnost rozkladů na Q tak, že $\varphi_1, \varphi_2, \ldots, \varphi_i = \varphi_{i+1}$, kde $\varphi_1 = \{F, Q \setminus F\}$
2. Při odvození rozkladů φ_i z φ_{i-1} postupujeme následovně:

   Pro $\forall R \in \varphi_{i-1}$ provedeme:

   (a) Vezmeme libovolný stav $r \in R$.

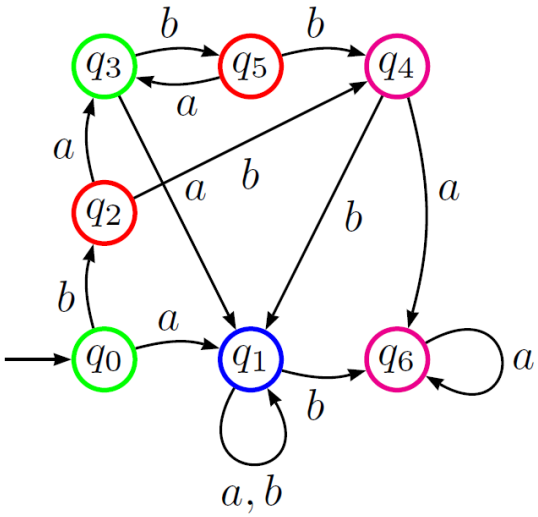
   (b) Položíme $S = \{s \in R |$ pro každé $a \in \Sigma$ platí také, že $\delta(S, a) \in [\delta(R, a)]\}$.

   (c) Pokud $S = R$, pak vložíme R do φ_i, pokud $S \subseteq R$, pak vložíme S a $R \setminus S$ do φ_i.

Věta 26: Korektnost algoritmu pro nalezení $\equiv_A$: pokud $\varphi_i = \varphi_{i+1}$, pak $\varphi_i = Q/\equiv_A$.

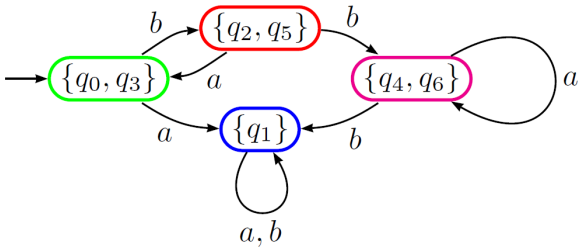
Příklad 26: Mějme výraz $b(ab)^*ba^*$. Dále

$$\begin{aligned}
\varphi_1 &= \{\{q_0, q_1, q_2, q_3, q_5\}, \{q_4, q_6\}\} \\
\varphi_2 &= \{\{q_0, q_1, q_3\}, \{q_2.q_5\}, \{q_4, q_6\}\} \\
\varphi_3 &= \{\{q_0, q_3\}, \{q_1\}, \{q_2.q_5\}, \{q_4, q_6\}\}
\end{aligned}$$



Obrázek 3: q_4 a q_6 jsou přijímací (šipka mezi q_1 a q_6 má být obráceně)

Následně rozklady:



	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_0	*	X	X	*	X	X	X
q_1	-	*	X	X	X	X	X
q_2	-	-	*	X	X	*	X
q_3	-	-	-	*	X	X	X
q_4	-	-	-	-	*	X	*
q_5	-	-	-	-	-	*	X
q_6	-	-	-	-	-	-	*

Tabulka 4: Tabulkový popis rozkladů stavů

V tabulce **??** se objevují znaky X na pozicích, pro které platí, že:

$$\begin{array}{lll} T[q,x] & \text{kde platí} & q \in F, x \notin F \\ & \text{nebo} & q \notin F, x \in F \end{array}$$

Projdeme prázdná místa, $T[q,r]$ = „prázdno, pokud“ $\exists a \in \Sigma$ tak, že:

- buď $T[\delta(q,a), \delta(r,a)] = X$
- nebo $T[\delta(r,a), \delta(q,a)] = X$

Pokud ano, tak „X“ na pozici $[q,r]$.

1.10.4 Izomorfismus automatů

Pro KDA A_1 a A_2 mějme zobrazení

$$h: \quad Q_1 \to Q_2$$

a toto zobrazení označme jako *izomorfismus*, pokud platí:

1. Zobrazení h je bijektivní.
2. Počáteční stav automatu A_1 e zobrazí na počáteční stav automatu A_2.
3. Pro všechna $q \in Q$ platí, že $Q \in F_1$ právě, pokud $q \in F_2$.
4. Zobrazení h je kompatibilní s přechodovou funkcí.

Věta 27: Jsou-li dva automaty izomorfní, pak $L(A_1) = L(A_2)$.

Definice 20: Mějme regulární jazyk L, pak KDA s úplnou přechodovou funkcí je *minimálním automatem* pro L, pokud $L(A) = L$ a pro každý KDA B takový, že $L(B) = L$ platí, že B má tolik stavů jako A.

Věta 28: Je-li automat A minimální pro jazyk L, pak je *redukovaný*a nemá nedosažitelné stavy.

Věta 29: Pokud jsou automaty A, B KDA bez nedosažitelných stavů a pokud jsou tyto automaty navíc redukované a existuje stejný jazyk, který generují, pak jsou izomorfní.

Věta 30: A je minimální automat pro jazyk L právě tehdy, když A neobsahuje nedosažitelné stavy a je redukovaný.

1.11 Pumping lemma

Poznámka 10: Jen drobné upozornění na začátek: jedná se o tvrzení ve tvaru když $\rightarrow$ pak, tj. "Pokud je L regulární, pak ..."

Věta 31: Nechť L je regulární jazyk nad Σ. Pak existuje $n \in \mathbb{N}$ tak, že pro každý řetězec $w \in L$ délky alespoň n platí, že existují $x, y, z \in \Sigma^*$ tak, že jsou splněny podmínky:

1. $w = xyz$
2. $|xy| \leq n$
3. $y \neq \varepsilon$
4. pro každé $i \geq 0$ platí, že $xy^i z \in L$

Příklad 27: Jazyk $L = \{a^n b^n |\quad \text{kde } n \in N\}$ není regulární.

1.12 Bezkontextové jazyky a jejich vlastnosti (uzávěrové vlastnosti, jednoznačnost)

V této části se vrátíme k problematice bezkontextových gramatik. Před dalším čtením je potřeba ovládat základní pojmy, zejména

- Bezkontextové gramatiky
- Bezkontextové jazyky
- P-derivace
- Odvozování řetězců
- Věty a větné formy

Navíc si zavedeme duální pojem k derivaci - *redukci.*

Definice 21: Řetězec v lze redukovat na řetězec u, pokud $u \Rightarrow_G^* v$. Značíme $v \Leftarrow_G^* u$.

V následujícím příkladě si ukážeme problém nejednoznačnosti bezkontextových gramatik.

Příklad 28: Mějme gramatiku:

$G = \langle N, \Sigma, P, S \rangle$
$\Sigma = \{a, b, c, 0, 1, +, -, *, /, (,)\}$
$N = \{S, E, C, V\}$
$P = \{\quad S \rightarrow E,$
$\qquad E \rightarrow E * E|E/E|E + E|E - E| - E|C|V|(E),$
$\qquad C \rightarrow 0C|1C|0|1,$
$\qquad V \rightarrow aV|bV|cV|a|b|c \quad \}$

Uvažujme větu $w = ac * 1 - c$. K ní se lze dostat buď:

$S \Rightarrow_G E \Rightarrow_G E * E \Rightarrow_G V * E \Rightarrow_G V * E - E \Rightarrow_G V * E - V \Rightarrow_G aV * E - V \Rightarrow_G ac * E - V \Rightarrow_G ac * C - V \Rightarrow_G ac * C - c \Rightarrow_G ac$*1-c*

nebo

$S \Rightarrow_G E \Rightarrow_G E * E \Rightarrow_G V * E \Rightarrow_G aV * E \Rightarrow_G ac * E \Rightarrow_G ac * E - E \Rightarrow_G ac * C - E \Rightarrow_G ac * 1 - E \Rightarrow_G ac * 1 - V \Rightarrow_G ac$*1-c*

Ačkoliv odvozujeme stejnou větu, můžeme zde pozorovat jakousi nejednoznačnost, způsobenou tím, že neterminály derivujeme v libovolném pořadí.

Tento problém bychom mohli vyřešit použitím tzv. lineární bezkontextové gramatiky, tedy takové, jejíž odvozovací pravidla obsahují pouze jeden neterminální symbol na pravé straně.

Příklad 29: Lineární gramatika může vypadat např. takto:

$G = \langle N, \Sigma, P, S \rangle$

$P = \{ \quad S \to abB,$
$\qquad A \to aaBb|\varepsilon,$
$\qquad B \to bbAa \quad \}$

$L(G) = \{ab(bbaa)^n bba(ba)^n \mid n \geq 0\}$

Ovšem k nejednoznačnosti může stejně dojít, pokud by se z různých neterminálních symbolů daly odvodit stejná slova.

Příklad 30: Příklad nejednoznačné lineární gramatiky:

$G = \langle N, \Sigma, P, S \rangle$

$P = \{ \quad S \to aA \mid aB,$
$\qquad A \to bA|a,$
$\qquad B \to bB|a \quad \}$

Uvažujme slovo $w = abba$, ke kterému lze dojít buď:

$S \Rightarrow_G aA \Rightarrow_G abA \Rightarrow_G abbA \Rightarrow_G abba$

nebo:

$S \Rightarrow_G aB \Rightarrow_G abB \Rightarrow_G abbB \Rightarrow_G abba$

Mějme bezkontextovou gramatiku $G = \langle N, \Sigma, P, S \rangle$.

Definice 22: P-derivace $x_0, \ldots, x_k$ se nazývá *nejlevější* P-derivace, pokud pro každé $i \in \{1, \ldots, k\}$ platí, že x_{i-1} je ve tvaru uAv, kde $u \in \Sigma^*, A \in N, v \in (\Sigma \cup N)^*$ a x_i je ve tvaru uyv, kde $A \to y \in P$.

Poznámka 11: Řetězci u se říká uzavřená forma a řetězci Av otevřená forma (větné formy uAv). Odvození pomocí nejlevější derivace značíme $u \Rightarrow_{G,l} v$.

Věta 32: Mějme $v \in \Sigma^*$ a $X \in N$. Pokud existuje P-derivace $X, \ldots, v$, pak existuje nejlevější P-derivace $X, \ldots, v$ používající stejnou množinu pravidel jako výchozí P-derivace.

Příklad 31: Uvažujme gramatiku z příkladu **??**

$S \Rightarrow^*_G E + C$ tady problém není
$S \Rightarrow^*_{G,l} E + C$ tento výraz už smysl nedává. Pomocí nejlevější derivace bychom zákonitě museli nejdříve derivovat E na levé straně výrazu $E + E$

Poznámka 12: Lze zavést duálně nejpravější derivaci.

Příklad 32: Zase se odkážeme na příklad **??**.

Výraz $10 + (ca * 110)$ má jedinou nejlevější derivaci

Naopak $a + 10 * c$ jich má několik:

$S \Rightarrow_{G,l} E \Rightarrow_{G,l} E + E \Rightarrow_{G,l} V + E \Rightarrow_{G,l} a + E \Rightarrow_{G,l} a + E * E \Rightarrow_{G,l} a + C * E \Rightarrow_{G,l} a + 1C * E \Rightarrow_{G,l} a + 10 * E \Rightarrow_{G,l} a + 10 * V \Rightarrow_{G,l} a + 10 * c$

nebo:

$S \Rightarrow_{G,l} E \Rightarrow_{G,l} E * E \Rightarrow_{G,l} \ldots \Rightarrow_{G,l} a + 10 * c$ (už ve druhém odvození můžeme pozorovat rozdíl)

Je to způsobeno jinou nejednoznačností než tou, kterou jsme eliminovali pomocí nejlevější derivace.

1.12.1 Derivační stromy

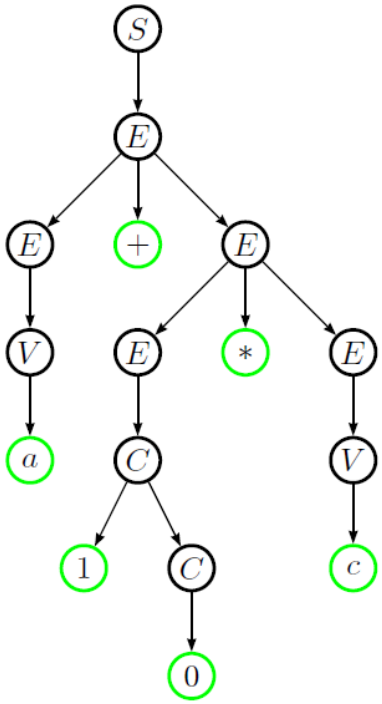
Slouží ke grafickému znázornění nejlevějších derivací. Mějme gramatiku $G = \langle N, \Sigma, P, S \rangle$.

Definice 23: Derivačním stromem slova $x \in (\Sigma \cup N)^*$ z $X \in N$ podle pravidel z G je každý strom splňující:

1. každý vnitřní uzel je ohodnocen neterminálem
2. kořen je ohodnocen X
3. pokud je vnitřní uzel označený $A \in N$ a jeho potomci jsou zleva doprava označeni $X_1, \ldots, X_k \in N \cup \Sigma$ pak existuje pravidlo $A \to X_1 \cdots X_k \in P$
4. zřetězením hodnot v listech při průchodu stromu do hloubky a zleva doprava získáme řetězec x

Příklad 33: Opět pracujeme s gramatikou z příkladu **??**.

$S \Rightarrow^*_{G,l} a + 10 * c$ Derivační strom bude vypadat následovně:



Věta 33: Pokud $X \Rightarrow^*_{G,l} x$ pak existuje derivační strom x z X podle G.

Věta 34: Pokud existuje derivační strom x z X podle G, pak $X \Rightarrow^*_{G,l} x$

Věta 35: $S \Rightarrow^*_{G,l} u$ právě když existuje derivační strom u z S podle G.

1.12.2 Jednoznačné a nejednoznačné bezkontextové gramatiky

Definice 24: Bezkontextová gramatika $G = \langle N, \Sigma, P, S \rangle$ se nazývá nejednoznačná, pokud existuje věta $x \in L(G)$, která má více než jeden derivační strom z S podle G. V opačném případě se nazývá jednoznačná bezkontextová gramatika.

Definice 25: Bezkontextový jazyk L se nazývá jednoznačný pokud existuje jednoznačná gramatika G tak, že $L = L(G)$.

Definice 26: Bezkontextový jazyk L se nazývá *inherentně* nejednoznačný, pokud neexistuje žádná jednoznačná gramatika G taková, že $L = L(G)$.

Věta 36: Každý regulární jazyk je jednoznačný.

Příklad 34: $L = \{a^n b^n c^p d^p \mid n, p \geq 0\}$ je inherentně nejednoznačný.

1.13 Uzávěrové vlastnosti bezkontextových jazyků

- *Sjednocení*

  Mějme dány bezkontextové jazyky L_1, L_2 a korespondující bezkontexotvé gramatiky G_1 a G_2. Lze předpokládat, že množiny stavů obou gramatik jsou disjunktní (tj. $N_1 \cap N_2 = \emptyset$).

  Dále uvažujeme gramatiku $G = \langle N, \Sigma, P, S \rangle$, kde
  $N = N_1 \cup N_2 \cup \{S\}$
  $\Sigma = \Sigma_1 \cup \Sigma_2$
  $P = P_1 \cup P_2 \cup \{S \to S_1 \mid S_2\}$

  Platí, že $L(G) = L(G_1) \cup L(G_2)$.

- *Produkt*

  L_1 a L_2 jsou bezkontextové jazyky, G_1 a G_2 jsou odpovídající bezkontextové gramatiky.

  Zavedeme gramatiku $G = \langle N, \Sigma, P, S \rangle$, kde
  $N = N_1 \cup N_2 \cup \{S\}$
  $\Sigma = \Sigma_1 \cup \Sigma_2$
  $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$

  Platí, že $L(G) = L(G_1) \cdot L(G_2)$.

- *Kleeneho uzávěr*

  Mějme bezkontextový jazyk L a k němu odpovídající gramatiku G.

  Vytvoříme novou gramatiku $G' = \langle N \cup \{S'\}, \Sigma, P \cup \{S' \to \varepsilon \mid SS'\}, S' \rangle$.

- *Pozitivní uzávěr*

  Podobné jako u Kleeneho uzávěru, akorát výsledná gramatika se bude lišit v množině pravidel.

  $G' = \langle N \cup \{S'\}, \Sigma, P \cup \{S' \to S \mid SS'\}, S' \rangle$

- *Průnik*

  $\mathcal{L}_2$ není uzavřená na průnik.

Příklad 35: Uvedeme si příklad, který nám tvrzení potvrdí.
$\Sigma = \{a, b, c\}$
$L_1 = \{a^n b^n c^* \mid n \geq 0\}$
$L_2 = \{a^* b^n c^n \mid n \geq 0\}$

$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$

Z toho také plyne, že není uzavřená ani na Komplement či Množinový rozdíl (De Morganovy zákony).

Bezkontextové gramatiky v programátorské praxi *BNF*(Backus-Naurova forma): Vytvořili ji John Backus a Peter Naur. Zápis se řidí několika pravidly:

- neterminální symboly se zapisují do $\langle \ldots \rangle$
- používá se ::= místo $\rightarrow$
- terminální symboly se píší do uvozovek (" $*$ ")
- pravidla se oddělují pomocí |

Příklad 36: Uvedeme si příklad zápisu gramatiky pomocí BNF.
$\langle expr \rangle$ ::= "("$\langle expr \rangle$")" | $\langle expr \rangle \langle op \rangle \langle expr \rangle$ | $\langle number \rangle$
$\langle op \rangle$ ::= " + " | " − " | " $*$ " | "/"
$\langle number \rangle$ ::= $\langle digit \rangle \langle number \rangle$ | $\langle digit \rangle$
$\langle digit \rangle$ ::= "0" | ... | "9"

Extended BNF: Zavedl Niklaus Wirth v roce 1977. Odpovídá BNF, pouze zjednodušuje její notaci a to tak že:

- každé pravidlo je ukončeno znakem ";"
- terminální symboly se píší do uvozovek, neterminální ne
- [,] značí volitelnou část výrazu
- {, } indukují možnost opakování
- (,) jsou použity pro shlukování výrazů
- místo ::= se používá =
- terminální a neterminální symboly se oddělují čárkou

Příklad 37: Gramatiku z předchozího příkladu můžeme zapsat pomocí EBNF takto:
$expr$ = "(", $expr$, ")" | $expr, op, expr$ | $number$;
op = " + " | " + " | " − " | " $*$ " | "/";
$number$ = $[signum], digit, \{digit\}$;
$digit$ = "0" | ... | "9";
$signum$ = " + " | " − ";

1.14 Zásobníkové automaty a jejich modifikace

1.14.1 Nedeterministický zásobníkový automat

Hledáme silnější analytický aparát, než jsou KDA, protože ty v určitých situacích selhávají.

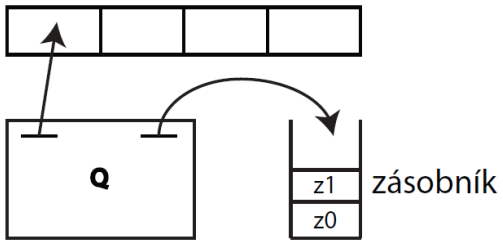
Příklad 38: Mějme jazyk

$$L = \{x \in \Sigma^* | \quad x \text{ je korektně uzávorkovaný výraz}\}$$

Následně podrobněji

$$L = \{x \in \Sigma^* | \Sigma = \{(,),[,]\} \quad x \text{ je korektně uzávorkovaný výraz}\}$$

Do tohoto jazyka patří například řetězce $()[], ([]), \ldots$, ale nepatří do něj řetězce $())$, $][[, \ldots$



Obrázek 4: Náčrt zásobníkového automatu.

Definice 27: Jako *nedeterministický zásobníkový automat* označme následující strukturu:

$$A = \langle \Sigma, \Gamma, Q, \delta, q_0, z_0, F \rangle$$

Nyní si popišme význam nám dosud neznámých prvků tohoto automatu:

Γ	–	abeceda zásobnákových symbolů.
z_0	–	počáteční zásobníkový symbol.
Q	–	konečná množina stavů.
δ	–	přechodová funkce ve tvaru $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$

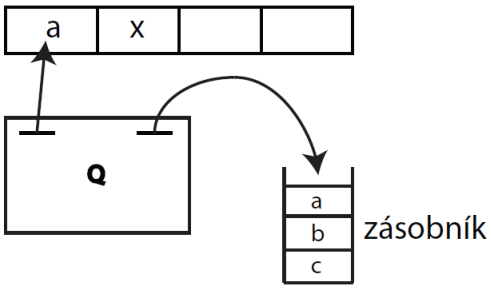
Navíc $\langle \sigma, abc \rangle \in \delta(q, a, d)$ znamená slovně: „Ze stavu q po přečtení a ze vstupu a symbolu d ze zásobníku přejde automat A do stavu σ a na zásobník zapíše řetězec "abc"." Mějme na paměti, že tento řetězec je na zásobník zapsán „obráceně".

1.14.2 Reprezentace NZA

Přechodová tabulka V přechodové tabulce řádky odpovídají stavům a sloupce odpovídají prvkům $(\Sigma \cup \{\varepsilon\}) \times \Gamma$.

Příklad 39:

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{a, b\} \\ \Gamma &= \{0, 1\} \end{aligned}$$

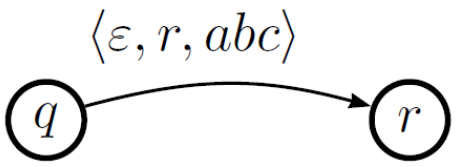


Obrázek 5: Vkládání hodnot na zásobník u NZA.

	$\langle a,0\rangle$	$\langle b,0\rangle$	$\langle c,0\rangle$	$\langle a,1\rangle$	$\langle b,1\rangle$	$\langle c,1\rangle$
$\rightarrow q_0$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
q_1	$\cdots$	$\{\langle r,w\rangle\}$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Tabulka 5: Ukázka přechodové tabulky pro NZA

Přechodový diagram Přechodové diagramy NZA jsou velmi podobné diagramům KA.



Definice 28: Konfigurací NZA je každý prvek z množiny $Q \times \Sigma^* \times \Gamma^*$, tedy $\langle q, abc, 001\rangle \in Q \times \Sigma^* \times \Gamma^*$ znamená, že NZA je ve stavu q, na vstupu zbývá přečíst řetězec „abc“ a na zásobníku je řetězec „001“.

Pro konfigurace $\langle q_i, w_i, x_i\rangle$ a $\langle q_j, w_j, x_j\rangle$ klademe $\langle q_i, w_i, x_i\rangle \vdash \langle q_j, w_j, x_j\rangle$ právě, když platí:

- $x_i = zy$,kde $z \in \Gamma, y \in \Gamma^*$
- $w_i = aw_j$,kde $a \in (\Sigma \cup \{\varepsilon\}), w_j \in \Sigma^*$
- $\langle q_j, x\rangle \in \delta(q_i, a, z)$

Přičemž $x_j = xy$.

Reflexivní a tranzitivní uzávěr $\vdash$ označíme $\vdash^*$ a platí:

$$\langle q_i, w_i, x_i\rangle \vdash^* \langle q_j, w_j, x_j\rangle \text{ právě, když}$$
$$\langle q_i, w_i, x_i\rangle \vdash \cdots \vdash \langle q_j, w_j, x_j\rangle$$

Pokud $\langle q_i, w_i, x_i\rangle \vdash^* \langle q_j, w_j, x_j\rangle$ tak říkáme, že existuje výpočet NZA, kterým přejde tento NZA z $\langle q_i, w_i, x_i\rangle$ do $\langle q_j, w_j, x_j\rangle$.

1.14.3 Jazyky rozpoznávané NZA

1. Jazyk rozpoznávaný pomocí koncových stavů automatu, tedy:

$$L(A) = \{w \in \Sigma^* | \quad \langle q_0, w, z_0\rangle \vdash^* \langle q, \varepsilon, x\rangle, q \in F, x \in \Gamma^*\}$$

2. Jazyk rozpoznávaný vyprazdňováním zásobníků, tedy:

$$N(A) = \{w \in \Sigma^* | \quad \langle q_0, w, z_0\rangle \vdash^* \langle q, \varepsilon, \varepsilon\rangle, q \in Q\}$$

1.14.4 Rozpoznání jazyk způsobem přechodu od koncových stavů k vyprázdnění zásobníku

Věta 37: Ke každému NZA A existuje NZA A', pro který platí $L(A) = N(A') = L(A')$

Uvažujme

$$A' = \langle \Sigma, \Gamma \cup \{z'_0\}, Q \cup \{q'_0, q'_\#\}, \delta', q'_0, z'_0, \{q'_\#\}\rangle$$

kde δ' je přechodová funkce, jež vznikne rozšířením původní přechodové funkce o následující přechody:

$$\delta'(q'_0, \varepsilon, z'_0) = \{\langle q_0, z_0, z'_0\rangle\}$$
$$\delta'(q, \varepsilon, z) = \delta(q, \varepsilon, z) \cup \{\langle q'_\#, \varepsilon\rangle\} \text{ pokud } q \in F, a \in \Gamma \cup \{z'_0\}$$
$$\delta'(q'_\#, \varepsilon, z) = \{\rangle q'_\#, \varepsilon\langle\} \text{ pro } z \in \Gamma \cup \{z'_0\}$$

1.14.5 Rozpoznání jazyk způsobem přechodu od vyprázdnění zásobníku k přijímání koncovými stavy

Věta 38: Ke každému NZA $A = \langle \Sigma, \Gamma, Q, \delta, q_0, z_0, F\rangle$ existuje NZA A' tak, že $N(A) = L(A') = N(A')$.

1.14.6 Automaty pracující s celým zásobníkem

Pro automat pracující s celým zásobníkem má platit, že $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \to 2^{Q \times \Gamma^*}$ s omezením, že přechodová funkce je pro každé $q \in Q, a \in (\Sigma \cup \{\varepsilon\})$ definována pouze pro konečně mnoho řetězců $z \in \Gamma^*$ a navíc musí platit, že pro každý z konečně mnoha řetězců je $\delta(q, a, z)$ konečná.

Platí také:

$$\langle q_i, w_i, z_i\rangle \vdash \langle q_j, w_j, z_j\rangle$$

1. $z_i = zy, z \in \Gamma^*, y \in \Gamma^*$
2. $w_i = aw_j, a \in (\Sigma \cup \{\varepsilon\})$
3. $\langle q_j, x\rangle \in \delta(q_i, a, z)$
4. $z_j = xy$

Věta 39: Pro každý NZA A pracující s celým zásobníkem existuje NZA A' pracující pouze s vrcholem zásobníku.

Důsledek 3: Třídy automatu jsou ekvivalentní.

1.14.7 Poznámky o vlastnostech NZA

1. Pokud $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$ pak $\langle q, uw, x\rangle \vdash^* \langle p, vw, y\rangle$ pro libovolný řetězec $w \in \Sigma^*$.
2. Pokud $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$ pak $\langle q, u, xz\rangle \vdash^* \langle p, v, yz\rangle$ pro libovolný řetězec $w \in \Gamma^*$.
3. Pokud $\langle q, uw, x\rangle \vdash^* \langle p, vw, y\rangle$ pak $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$.

1.+2. Pokud $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$ pak $\langle q, uw, xz\rangle \vdash^* \langle p, vw, yz\rangle$ pro všechny řetězce $w \in \Sigma^*$ a $z \in \Gamma^*$.

1.+3. $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$ právě když $\langle q, uw, x\rangle \vdash^* \langle p, vw, y\rangle$.

Poznámka 13: Opak 2. vlastnosti neplatí: Pokud $\langle q, u, xz\rangle \vdash^* \langle p, v, yz\rangle$ pak obecně nelze odvodit, že $\langle q, u, x\rangle \vdash^* \langle p, v, y\rangle$.
Speciálně: $x = y = \varepsilon$

Poznámka 14: $\langle q, u, z\rangle \vdash^* \langle p, \varepsilon, \varepsilon\rangle$: automat je schopen přejít ze stavu q do p, přitom zkonzumuje řetězec u ze vstupu a řetězec z ze zásobníku, ale bez toho, aniž by se někdy během činnosti dostal pod úroveň řetězce z na zásobníku.

1.15 Deterministické zásobníkové automaty, Deterministické bezkontextové jazyky

Definice 29: Řekneme, že zásobníkový automat $A = \langle \Sigma, \Gamma, Q, \delta, q_0, z_0, F\rangle$ je *deterministický* (DZA), pokud jsou pro každé $q \in Q$ a $z \in \Gamma$ splněny následující podmínky:

1. pro všechny $a \in \Sigma \cup \{\varepsilon\}$ platí $|\delta(q, a, z)| \leq 1$
2. pokud $\delta(q, \varepsilon, z) \neq \emptyset$ pak $\delta(q, a, z) = \emptyset$ pro každé $a \in \Sigma$

Nabízí se otázka, jaké třídy jazyků, jsou vlastně rozpoznatelné deterministickým zásobníkový automatem pomocí buď koncových stavů, nebo vyprázdněním zásobníku.

Definice 30: Bezkontextový jazyk L se nazývá *deterministický*, pokud existuje deterministický zásobníkový automat A tak, že $L = L(A)$.

Definice 31: Bezkontextový jazyk L má *prefixovou vlastnost*, pokud L neobsahuje dva různé řetězce x, y tak, že x je prefixem y.

Věta 40: Pro libovolný bezkontextový jazyk L jsou následující tvrzení ekvivalentní:

1. $L = N(A)$ pro nějaký DZA A
2. $L = L(A')$ pro nějaký DZA A' a L má prefixovou vlastnost

Důsledek 4: Deterministický bezkontextový jazyk má prefixovou vlastnost, právě když je přijímán nějakým DZA pomocí vyprázdnění zásobníku.

Věta 41: Každý regulární jazyk je deterministický.

Příklad 40: Následující jazyk je deterministický:

$$L = \{a^n b^n \mid n \geq 1\}$$

Poznámka 15: Mnoho regulárních jazyků *nemá* prefixovou vlastnost.

Věta 42: Pokud $L = N(A)$ pro DZA A, pak je L jednoznačný.

Věta 43: Pokud $L = L(A)$ pro DZA A, pak je L jednoznačný. Deterministické bezkontextové jazyky jsou jednoznačné.

2 Druhý odstavec

Turingův stroj (TS), nedeterministický TS. Jazyk přijímaný TS, jazyk rozhodovaný TS. Church-Turingova teze, varianty TS. Částečně rekurzivní a rekurzivní jazyky, jazyky a rozhodovací problémy. Vztah rekurzivních a částečně rekurzivních jazyků. Uzávěrové vlastnosti jazyků TS. Riceova věta. Vztah jazyků TS k jazykům Chomského hierarchie. Věta o rekurzi.

2.1 Turingův stroj (TS)

- Alan Turing, 1936
- účel: porozumět omezením mechanického výpočtu

TS se skládá z

- řídící jednotky, která se vždy nachází v jednom z konečného množství stavů
- zleva omezené nekonečné pásky rozdělené na políčka, v každém políčku je zapsán jeden symbol
- čtecí / zapisovací hlavy která je vždy umístěna nad jedním políčkem pásky

Definice: Turingův stroj je struktura $\langle Q, \Sigma, \Gamma, \delta, q_{start}, q_+, q_- \rangle$ dána:

- Neprázdnou konečnou množinou stavů Q
- Vstupní abecedou Σ t.ž. $\textvisiblespace \notin \Sigma$
- páskovou abecedou Γ t.ž. $\Sigma \subset \Gamma$, $\textvisiblespace \in \Gamma$
- přechodovou funkcí $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- počátečním stavem $q_{start} \in Q$
- přijímacím stavem $q_+ \in Q$ a zamítacím stavem $q_- \in Q$. $q_+ \neq q_-$

Program TS lze chápat jako množinu elementárních instrukcí ve tvaru:

„Pokud je řídící jednotka ve stavu q a čtecí/zapisovací hlava čte symbol a, tak změň stav řídící jednotky na q', na pásku zapiš a' a posuň čtecí/zapisovací hlavu o jedno políčko směrem d.“

Takováto instrukce se zapisuje jako $\delta(q, a) = (q', a', d)$ a nazýváme ji přechod. Celý program, tedy množinu takovýchto instrukcí, pak nazýváme přechodovou funkcí TS

Konfigurace TS je uspořádaná trojice $(q, \alpha, n) \in Q \times \Gamma^* \times N_0$, která zachycuje aktuální status všech tří komponent.

- q je aktuální stav řídící jednotky
- α je obsah pásky
- n je pozice hlavy

Krok výpočtu TS je definován jako binární relace na množině konfigurací: Nechť $(q, a_0 \dots a_n, i)$ je taková konfigurace T, kde $q \neq q_\pm, n \in N_0, a_0, \dots, a_n \in \Gamma, i \leq n$.

- Je-li $1 \leq i \leq na\delta(q, a_i) = (q', b, L), pak$

$$(q, a_0 \dots a_n, i) \vdash (q', a_0 \dots a_{i-1} b a_{i+1} \dots a_n, i-1)$$

- Je-li $\delta(q, a_0) = (q', b, L), pak$

$$(q, a_0 \dots a_n, 0) \vdash (q', b a_1 \dots a_n, 0)$$

- Je-li $\delta(q, a_i) = (q', b, R), pak$

$$(q, a_0 \dots a_n, i) \vdash (q', a_0 \dots a_{i-1} b a_{i+1} \dots a_n, i+1)$$

2.2 Nedeterministický TS

Obdobný rozdíl jako u konečných deterministických a konečných nedeterministických automatů. Přechodová funkce ve tvaru:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Definice: Nedeterministický TS je struktura $\langle Q, \Sigma, \Gamma, \delta, q_{start}, q_+, q_- \rangle$ dána:

- Neprázdnou konečnou množinou stavů Q
- Vstupní abecedou Σ t.ž. $\llcorner \notin \Sigma$
- páskovou abecedou Γ t.ž. $\Sigma \subset \Gamma$, $\llcorner \in \Gamma$
- přechodovou funkcí $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$
- počátečním stavem $q_{start} \in Q$
- přijímacím stavem $q_+ \in Q$ a zamítacím stavem $q_- \in Q$. $q_+ \neq q_-$

Ke každému deterministickému existuje ekvivalentní nedeterministický automat a ke každému nedeterministickému existuje ekvivalentní deterministický automat.

2.3 Jazyk přijímaný TS

Množinu všech slov $\omega \in \Sigma^*$, které TS T Přijímá značíme $L(T)$ a nazýváme ***jazykem Turingova stroje***, t.j.

$$L(T) = \{\omega | \omega \in \Sigma^*, (\omega, q_0, 0) \vdash^* C_+\}$$

Jazyk $L(T)$ nazýváme *jazyk přijímaný TS* T.
Říkáme, že TS T *přijímá jazyk*$L(T)$.

Jazyk $L \subseteq \Sigma^*$ nazveme *jazyk přijímaný TS*, pokud existuje TS T který jej přijímá.

2.4 Jazyk rozhodovaný TS

Pokud navíc platí, že TS T zamítá každé $\omega \notin L(T)$, nazýváme jazyk L(T) *jazyk rozhodovaný* TS T.
Říkáme, že TS T *rozhoduje jazyk* $L(T)$. (to znamená že nikdy necyklí)

Jazyk $L \subseteq \Sigma^*$ nazveme *jazyk rozhodovaný TS*, pokud existuje TS T který jej rozhoduje.

2.5 Churg-Turingova teze

Intuitivní pojem algoritmu = algoritmus TS.
(to je všechno co k tomu máme)

Wikipedie: Teze může znít například takto: *„Ke každému algoritmu existuje ekvivalentní Turingův stroj.“* Kvůli neformální definici pojmu algoritmus nemůže být tato teze nikdy dokázána, lze ji ale vyvrátit, podaří-li se sestrojit stroj, který bude umět řešit problémy, které Turingův stroj řešit neumí (např. problém zastavení).

Jelikož každý počítačový program lze přeložit do jazyka Turingova stroje a obvykle i naopak, lze tezi ekvivalentně formulovat pro kterýkoli běžně používaný programovací jazyk. Přesněji, programovací jazyk potřebuje jednu z následujících konstrukcí (kromě jiných), aby byl turingovsky úplný (tj. ekvivalentní Turingovu stroji):

- cyklus while-do
- neomezenou (alespoň teoreticky) rekurzi
- podmíněný skok

Běžné programovací jazyky mívají všechny tři tyto konstrukce. Mezi jazyky, které turingovsky úplné nejsou, patří SQL (myšleno bez vložených procedur) a Charity.

2.6 Varianty TS

- TS, který se nikdy nepokusí přejet levý okraj pásky
  - Rozumíme TS, u kterého při výpočtu nad jakýmkoli slovem nedojde k tomu že, je v konfiguraci $(q, a\omega, 0)$ a existuje přechod $\delta(q, a) = (q', b, L)$. Tedy nikdy nenastane situace že by byla hlava nad nejlevějším políčkem pásky a přechodová funkce by určovala pohyb vlevo.
- TS, který nikdy nezapíše na pásku $\llcorner$
  - Roziníme TS, který nemá žádný přechod ve tvaru $\delta(q, a) = (q', \llcorner, D)$
- TS, který „ po sobě uklízí“
  - Roziníme TS, který zastaví pouze v konfiguraci $\langle q_+, \epsilon, 0\rangle$ nebo $\langle q_-, \epsilon, 0\rangle$. Tzn. smaže obsah pásky.
- (předchozí 3 se nejspíš nepočítají mezi varianty TS)
- **TS s instrukcí stop** — struktura $\langle Q, \Sigma, \Gamma, \delta, q_{start}, q_+, q_-\rangle$ dána:
  - Neprázdnou konečnou množinou stavů Q
  - Vstupní abecedou Σ t.ž. $\llcorner \notin \Sigma$
  - páskovou abecedou Γ t.ž. $\Sigma \subset \Gamma$, $\llcorner \in \Gamma$
  - přechodovou funkcí $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$
  - počátečním stavem $q_{start} \in Q$
  - přijímacím stavem $q_+ \in Q$ a zamítacím stavem $q_- \in Q$. $q_+ \neq q_-$

  Odpovídajícím způsobem se upraví definice kroku výpočtu, vlatně se přidá bod:
  – Je-li $\delta(q, a_i) = (q', b, S)$, pak

$$(q, a_0 \dots a_n, i) \vdash (q', a_0 \dots a_{i-1} b a_{i+1} \dots a_n, i)$$

- **TS s oboustranně nekonečnou páskou** — definice je totožná jako u klasického TS; rozdíl je v definici konfigurace (nevystačíme si s $\langle q, \omega, i\rangle \in Q \times \Gamma^* \times N_0$)a výpočtu (není zarážení o levý okraj)
  - **Konfigurace:** Alternativně zapisujeme jako řetězec $\alpha q \beta \in \Gamma^* Q \Gamma^*$. Konfigurace $\alpha q \beta$ představuje status stroje, který má na pásce zapsán řetězec $\alpha\beta$, hlava je nad prvním symbolem řetězce β, řídící jednotka je ve stavu q.
    U TS s oboustranně nekonečnou páskou považujeme $\alpha q \beta$ za totožnou s $\alpha q \beta \llcorner$a s $\llcorner \alpha q \beta$.

– **Krok výpočtu TS (s oboustranně nekonečnou páskou)** je definován jako binární relace na množině konfigurací: Nechť $\alpha aqb\beta$ je taková konfigurace T, kde $q \neq q, \alpha, \beta \in \Gamma^*, a, b \in \Gamma$.

∗ Je-li $\delta(q, a) = (q', x, L)$, pak

$$\alpha aqb\beta \vdash \alpha q'ax\beta$$

.

∗ Je-li $\delta(q, a_1) = (q', x, R)$, pak

$$\alpha aqb\beta \vdash \alpha axq'\beta$$

.

- **TS s více páskami** — je struktura $\langle Q, \Sigma, \Gamma, \delta, q_0, q_+, q_- \rangle$ dána:
  - Neprázdnou konečnou množinou stavů Q
  - Vstupní abecedou Σ t.ž. $\llcorner \notin \Sigma$
  - páskovou abecedou Γ t.ž. $\Sigma \subset \Gamma$, $\llcorner \in \Gamma$
  - přechodovou funkcí $\delta : Q \times \Gamma^k \to Q \times [\Gamma \times \{L, R\}]^k$
  - počátečním stavem $q_{start} \in Q$
  - přijímacím stavem $q_+ \in Q$ a zamítacím stavem $q_- \in Q$. $q_+ \neq q_-$
- **Lineárně ohraničený TS (LBA)** — klasický nedeterministický TS ale na pásce se pohybuje jen mezi dvěma zarážkami které nikdy nepřejede a nepřepíše (jako TS který nikdy nepřejede levý okraj)

Ke každé variantě TS vždy existuje ekvilalentní klasický TS a obráceně.

2.7 Částečně rekurzivní a rekurzivní jazyky, Jazyky a rozhodovací problémy, Vztah rekurzivních a částečne rekurzivních jazyků

- Jazykům rozhodovaným TS říkáme **rekurzivní jazyky (R)**
- Jazykům přijímaným TS říkáme **částečně rekurzivní jazyky (ČR)**
- R $\subseteq$ ČR

Jazyky, které jsou částečně rekurzivní ale nejsou rekurzivní:

- Univezální jazyl L_U definován takto :

$$L_U = \{[T, w] | T \text{ je } TS, \text{který přijímá } w\}$$

- Jazyk L_{HALT} definován takto:

$$L_{\text{HALT}} = \{[T, w] | T \text{ je } TS, \text{který zastaví pro } w\}$$

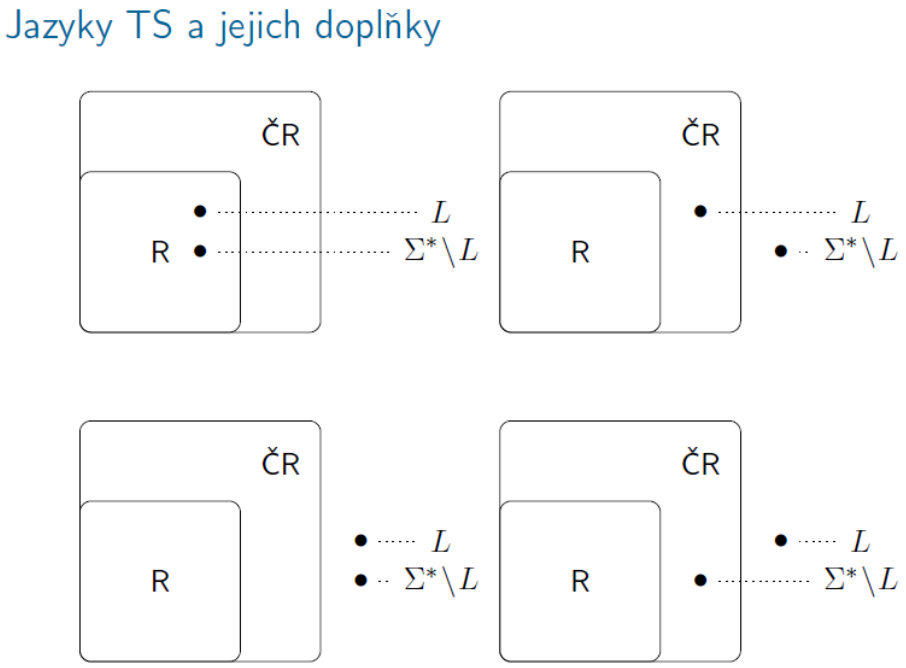
- Jazyk L_{NE} definován takto:

$$L_{\text{NE}} = \{[T] | L(T) \neq \emptyset\} \ldots \text{(zastaví alespoň pro jedno slovo)}$$

Jazyky, které nejsou částečně rekurzivní:

- Diagonální jazyk L_d

$$L_d = \{[T] | T \text{ je } TS, \text{který nepřijímá svůj kód}\}$$



Obrázek 6: Jiné stavy nastat nemůžou !!!

- Jazyk strojů přijímajících prázdný jazyk

$$L_\emptyset = \{[T] | T \text{ je } TS \text{ a } L(T) = \emptyset\} \ldots \text{(všechny slova zamítne?)}$$

- Jazyk strojů, které se nezacyklí

$$L_{noncycle} = \{[T] | T \text{ je } TS \text{ a necyklí pro žádné slovo } w\}$$

JAZYK $\approx$ PROBLÉM

- $L_U \rightarrow$ problém přijetí
- $L_{HALT} \rightarrow$ problém zastavení
- $L_{NE} \rightarrow$ problém neprázdnosti jazyka

2.8 Uzávěrové vlastnosti jazyků TS

Věty:

- Pokud L je rekurzivní pak, $\Sigma^* \setminus L$ je rekurzivní (staší sestrojit TS který rozhoduje L a prohodit přijímací a zamítací stavy).
- Nechť $L_1, L_2 \in R$ jsou rekurzuvní jazyky, pak $L_1 \cup L_2 \in R$ a $L_1 \cap L_2 \in R$.
- Nechť $L_1, L_2 \in R$, pak $L_1 \circ L_2 \in R$

- Pokud $L \in R$, pak $L^* \in R \ldots (L^*$ je kleenyho uzávěr- zřetězení slov z L)
- Nechť $L_1, L_2 \in$ ČR pak:
  - $L_1 \cup L_2 \in$ ČR
  - $L_1 \cap L_2 \in$ ČR
  - $L_1 \circ L_2 \in$ ČR
  - $L^* \in$ ČR
- Pokud $L \in$ ČR a $\Sigma^* \setminus L \in$ ČR, pak $L \in R$

2.9 Riceova věta :

Nechť L je jazyk, jehož slova jsou kódy TS, a platí:

- pokud $L(T_1) = L(T_2)$, pak $[T_1] \in L \leftrightarrow [T_2] \in L$
- $\exists T_1, T_2$ t.ž. $[T_1] \in L, [T_2] \notin L$

pak L není rekurzivní.

Věta slovně: *Pokud se rovnají jazyky strojů, pak jejich kódydo jazyku buďto oba patří nebo oba nepatří a dále existuje alespoň jeden stoj jehož kód do tohoto jazyka nepatří a alespoň jeden který do něj patří, pak není rekurzivní. (zjevně vyplývá třetí podmínka že slova těchto jazyků musí být kódy TS)*

Tvzdení ve tvaru implikace né ekvilalence! Tzn. že lze použít pouze k důkazu, že nějaký jazyk není rekurzivní a nelze použít k důkazu že nějaký jazyk je rekurzivní.

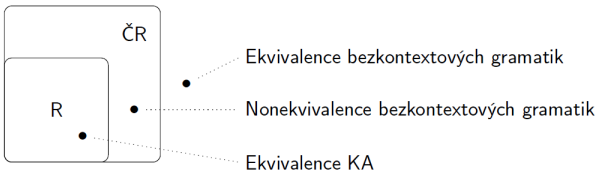
2.10 Vztah jazyků TS k jazykům Chomského hierarchie

$$\text{Typ } 3 \subset \text{Typ } 2 \subset \text{Typ } 1 \subset \text{R} \subset \text{Typ } 0 = \text{ČR}$$

- regulární jazyky = jazyky KA
- bezkontextové jazyky = jazyky nedeterministických ZA
- kontextové jazyky = jazyky LBA (lineárně ohraničený TS)
- jazyky gen. gramatikami bez omezení = jazyky přijímané TS

Věty:

- Třídy jazyků generovaných gramatikami typu 0 = částečně rekurzivní jazyky.
- Jazyky generované kontextově závislými gramatikamy = jazyky přijímané LBA
- Existuje rekurzivní jazyk, který není generovaný kontextovou gramatikou.



2.11 Věta o rekurzi

TS *vyčísluje funkci* f, pokud pro každé slovo $w \in \Sigma^*$ zapíše na pásku $f(w)$ a skončí.

Věta o rekurzi: Nechť T je TS, který vyčísluje $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. Existuje TS R, který počítá funkci $r : \Sigma^* \to \Sigma^*$, kde pro každé w platí

$$r(w) = t(\langle R \rangle, w)$$

Tvrzení věty říká, že abychom vytvořili TS R, který pracuje s vlastním kódem, stačí umět udělat TS T, který pracuje stejně, ale dostává kód $[R]$ na vstup.

Self-Reference

Nejdříve sestavíme stroj, který vypisuje svůj vlastní kód (při lib. vstupu).
Pomocné tvrzení:

Lemma

Existuje vyčíslitelná funkce $q : \Sigma^* \to \Sigma^*$, která jakýkoli řetězec w zobrazí na $[P_w]$, kde P_w je TS, který zapíše na pásku w a pak zastaví.

Důkaz.

Následující stroj Q vyčisluje $q(w)$:

TS Q pro slovo w:

1. *Sestav TS P_w který pro lib. vstup provádí:*
   - *smaž vstup,*
   - *zapiš w na pásku,*
   - *zastav.*
2. *Vypiš $[P_w]$.*

□

TS $SELF$, který bude vypisovat sám sebe rozdělíme na dva moduly A,B.
Nejříve bude spuštěn A, pak B.
Budeme brát, že kód $SELF$ je kombinací kódů A a B.

$$[SELF] = [AB]$$

A bude rovno $P_{[B]}$, což je $q([B])$ (z předchozího lematu a jeho důkazu).

B vypočítá $\langle A \rangle$: definovali jsme A jako $P_{[B]}$, takže v okamžiku, kdy se spustí B (tedy po skončení činnosti A), bude na pásce zapsáno $[B]$. B tedy má k dispozici svůj vlastní kód a může $[A]$ vypočítat jako $q([B])$, pak má k dispozici $[A]$ i $[B]$, může tedy vypsat $[SELF]$.

... podrobněji

TS $A = P_{[B]}$

TS B pro $[M]$, kde M je modul TS

- Vypočítej $q([M])$.
- Zkombinuj výsledek z předchozícho kroku s $[M]$ abys vytvořil celkový popis TS.
- Výsledek z předchozího kroku vytiskni.

TODO OBRAZEK
Všimněte si, že B nepotřebuje znát kód A, nedochází k žádné definici kruhem.

Aplikace věty o rekurzi – Věta o pevném bodě: Nechť $t : \Sigma^* \to \Sigma^*$ je vyčíslitelná funkce. Pak existuje TS F, t.ž. $t([F])$ je kód ekvivalentního TS.

3 Třetí odstavec

Složitost algoritmu (časová a paměťová). Třída P, třída NP, důvody jejich zavedení, jejich vzájemný vztah. NP- úplné problémy. Cook-Levinova věta. Příklady NP-úplných problémů, dokazovaní NP-úplnosti. Třída PSPACE, její vztah k třídám P a NP, PSPACE-úplné problémy. Třídy N a NL a NL-úplné problémy.

3.1 Složitost algoritmu(časová a paměťová)

Časová složitost = funkce $f : N \to N_0$, kde $f(n)$ je maximální počet kroků při výpočtu nad jakýmkoli vstupem délky n.

Paměťová složitost = funkce $f : N \to N$, kde $f(n)$ je maximální počet políček při výpočtu nad jakýmkoli vstupem délky n.

$f, g : N \to R$
$g(n)$ je **asymptotická hosní hranice** $f(n)$, když existují $c, n_0 \in N$, t.ž. pro každé $n \geq n_0$:

$$f(n) \leq c \cdot g(n)$$

Notace: $f(n) = O(g(n))$

3.2 Třída P, NP, důvody jejich zavedení, vzájemný vztah

turingův stroj

$$TIME(t(n)) = \{L | L \text{rozhodovaný TS v čase} O(t(n))\}$$
$$SPACE(s(n)) = \{L | L \text{rozhodovaný TS v paměti} O(s(n))\}$$

nedeterministický turingův stroj

NTS časová složitost t(n) ... TS časová složitost $2^{O(t(n))}$

NTS paměťová složitost t(n) ... TS časová složitost $O(t^2(n))$

$$NTIME(t(n)) = \{L | L \text{rozhodovaný NTS v čase} O(t(n))\}$$
$$NSPACE(s(n)) = \{L | L \text{rozhodovaný NTS v paměti} O(s(n))\}$$

Třídy P a NP:

$$P = \bigcup \{TIME(n^k) | k \in N\}$$
$$NP = \bigcup \{NTIME(n^k) | k \in N\}$$

Víme že

$$P \subseteq NP$$

co ale nevíme je jestli

$$NP \subseteq P \text{ tzn. (P=NP) nebo jestli } NP \nsubseteq P$$

3.3 NP–úplné problémy

Funkce $f : \Sigma^* \to \Sigma^*$ je **funkce vyčíslitelná v polynomiálním čase**, pokud existuje TS pracující v polynomiálním čase, který pro každé $w \in \Sigma^*$ zastaví a na pásce má zapsáno w.

Jazyk A je **redukovatelný v polynomiálním čase** na jazyk B, načeno $A \leq_P B$, pokud existuje redukce v polynomiálním čase, tj. $r : \Sigma^* \to \Sigma^*$, t.ž.

$$w \in A \text{ p. k. } f(w) \in B$$

Jazyk B je **NP-úplný**, pokud

- B je v NP
- B je NP-těžký. Tj. pro každý $A \in NP$ platí $A \leq_P B$

Věty:

- Pokud B je NP-úplný a $B \in P$, pak $P = NP$
- Pokud B je NP-úplný a $B \leq_P C$ pro $C \in$ NP, pak je NP-úplný
- SAT je NP-úplný
- 3SAT je NP-úplný

Hlavní důvod, proč jsou NP-úplné úlohy tak zajímavé, je právě jejich velmi obtížná řešitelnost. Díky ní nacházejí uplatnění v moderní kryptografii, kde musíme být schopni rychle ověřovat správnost řešení, ale jeho nalezení musí trvat dlouho. Obtížnost výpočtu ovšem záleží i na konkrétních datech, pro speciální množinu vstupů může být úloha polynomiální, například řešíme-li obarvení třemi barvami pro jednoduché grafy (cesty).

3.4 Cook–Levinova věta

Věta: SAT je NP-úplný ... (vic k tomu nemáme a důkaz psat nebudu stejně byste se na to vysrali a mě už ten LaTeXpěkně mrdá)

Na wikipedii je ta věta: „Existuje NP-úplný problém“. Víc nic.

3.5 Příklady NP–úplných problémů

- SAT je NP-úplný
- 3SAT je NP-úplný
- hledání hamiltonovské kružnice v grafu
- vrcholové pokrytí
- trojrozměrné párování

3.6 Dokazování NP–úplnosti

Věta: Pokud B je NP-úplný a $B \leq_P C$ pro $C \in$ NP, pak C je NP-úplný

3.7 Třída PSPACE, její vztah k třídám P a NP

PSPACE je třída jazyku, které jsou rozhodnutelné v polynomickém čase na (deterministickém) TS, tedy

$$\text{PSPACE} = \bigcup_k \text{SPACE}(n^k)$$

(ve slajdech je napsáno v polynomickém čase, ale podle mě tam má být polynomická paměť)
Ze Savitchovy věty plyne že

$$\text{PSPACE} = \text{NPSPACE}$$

Celkově tedy máme:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}$$

kde $\text{EXPTIME} = \bigcup_k \text{TIME}(2^n)$
Alespoň jedna z těchto inkluzí je vlastní: ví se, že P $\neq$ EXPTIME.
Věří se, že všechny ty jsou vlastní.

3.8 PSPACE–úplné problémy

Jazyk B je **PSPACE-úplný problém**, pokud

- $B \in$ PSPACE
- B je PSPACE-těžký, tj. každý $A \in$ PSPACE je redukovatelný v polynomickém čase na B

Grafová hra: Mějme orientovaný graf G s vyznačeným počátečním uzlem u_1

- Aktuální uzel je na začátku u_1
- Hráči se střídají v tazích
- Hráč na tahu zvolí souseda u_1, ten se stává aktuálním uzlem, nelze vybrat uzel, který už byl předtím navštíven
- Hráč, který uvízne prohrává

$$GG = \{[G, b] | \text{ hráč 1 má vítěznou strategii pro G a poč. b } \}$$

GG je PSPACE-úplný.

3.9 Třídy L, NL a NL–úplné problémy

$$\text{L} = \text{SPACE}(log(n))$$
$$\text{NL} = \text{NSPACE}(log(n))$$

Omezujeme se na logaritmickou paměťovou složitost ze stejného důvodu, jako jsme se předtím omezili na polynomickou časovou složitost: problémy řešitelné v logaritmické paměti považujeme za řešitelné efektivně.

NL-úplnost

Definice

Přepisovač v logaritmické paměti je TS s

- vstupní čtecí páskou (read-only),
- pracovní čtecí/zapisovací páskou,
- výstupní zapisovací páskou (write-only),

pracovní páska může obsahovat $\mathcal{O}(\log n)$ symbolů.

Přepisovač v logaritmické paměti M vyčisluje funkci $f : \Sigma^* \to \Sigma^*$, kde $f(w)$ je řetězec, který je na výstupní pásce, jakmile M zastaví.

f nazýváme funkcí vyčislovanou v logaritmické paměti.

Jazyk A je redukovatelný v logaritmické paměti na jazyk B, pokud je redukovatelný na B s použitím funkce vyčislované v logaritmické paměti. Zapisujeme $A \leq_{\text{L}} B$.

Definice

Jazyk B je NL-úplný, pokud

- $B \in \mathrm{NL}$,
- každý $A \in \mathrm{NL}$ je redukovatelný v logaritmické paměti na B.

Věta

Pokud $A \leq_{\mathrm{L}} B$ a $B \in \mathrm{L}$, pak $A \in \mathrm{L}$.

Důkaz na tabuli.

Důsledek

Pokud je jakýkoli NL-úplný jazyk v L, pak L=NL.

4 Čtvrtý odstavec

Výroková logika: jazyk, formule, pravdivostní ohodnocení, tautologie, tabulková metoda, sémantické vyplývání, normální formy formulí, úplné systémy spojek. Axiomatický systém výrokové logiky, syntaktické vyplývání. Věta o dedukci. Věty o korektnosti a úplnosti výrokové logiky. Predikátová logika: jazyk, termy a formule, struktury pro jazyk, ohodnocení termů a formulí. Axiomatický systém predikátové logiky, syntaktické vyplývání. Věty o korektnosti a úplnosti predikátové logiky. Neklasické logiky, fuzzy logika. Základy logického programování, úvod do Prologu.

4.1 Výroková Logika

4.1.1 jazyk

Definice

Jazyk výrokové logiky se skládá z

- **výrokových symbolů** $p, q, r, \ldots$, popř. s indexy, $p_1, p_2, \ldots$; předpokládáme, že máme spočetně mnoho výrokových symbolů;
- **symbolů výrokových spojek** $\neg$ (negace), $\Rightarrow$ (implikace) [případně dále $\wedge$ (konjunkce), $\vee$ (disjunkce), $\Leftrightarrow$ (ekvivalence)];
- **pomocných symbolů** (,), [,] (různé druhy závorek).

Formální jazyk odstraňuje nevýhody přirozeného jazyka.
V jazyku VL například nejsou formulovatelná tvrzení obsahující autoreference (viz paradoxy).

Ze symbolů jazyka sestávají formule VL. (Poznamenejme, že formule jsou přesným zavedením intuitivního pojmu výrok.)

Definice

Nechť je dán jazyk výrokové logiky. **Formule** daného jazyka výrokové logiky je definována následovně

- každý výrokový symbol je formule (tzv. **atomická formule**);
- jsou-li φ a ψ formule, jsou i výrazy $\neg\varphi$, $(\varphi \Rightarrow \psi)$ formule.

Formule jsou tedy jisté konečné posloupnosti symbolů jazyka VL. Například posloupnosti q_3, $\neg\neg\neg p$, $((\neg r \Rightarrow \neg q) \Rightarrow \neg\neg r)$ jsou formule VL, naproti tomu posloupnosti $\neg(p)$, $q \Rightarrow$, $p\neg p$, ((nejsou formule VL. (Stejně tak "Prší $\Rightarrow$ sněží.", "$\neg$ prší" nejsou formule VL.)

4.1.2 pravdivostní ohodnocení

Definice

(Pravdivostní) ohodnocení je libovolné zobrazení e výrokových symbolů daného jazyka výrokové logiky do množiny $\{0,1\}$, tj. ohodnocení e přiřazuje každému výrokovému symbolu p hodnotu 0 nebo 1.

0 a 1 reprezentují pravdivostní hodnoty nepravda a pravda. Hodnotu přiřazenou ohodnocením e symbolu p označujeme $e(p)$. Je tedy $e(p) = 0$ nebo $e(p) = 1$. Je-li dáno ohodnocení e, můžeme říci, co je to pravdivostní hodnota formule.

Pravdivostní hodnota libovolné formule je pravdivostním ohodnocením jednoznačně určena a je definována takto:

Definice

Nechť je dáno ohodnocení e. **Pravdivostní hodnota formule φ při ohodnocení e**, označujeme ji $\| \varphi \|_e$, je definována následovně:

- Je-li φ výrokovým symbolem p, pak $\| p \|_e = e(p)$.
- Je-li φ složená formule, tedy má tvar $\neg\psi$ nebo $\psi \Rightarrow \theta$, pak
  $\| \neg\psi \|_e = 1$, pokud $\| \psi \|_e = 0$;
  $\| \neg\psi \|_e = 0$, pokud $\| \psi \|_e = 1$ a

  $\| \psi \Rightarrow \theta \|_e = 1$, pokud $\| \psi \|_e = 0$ nebo $\| \theta \|_e = 1$;
  $\| \psi \Rightarrow \theta \|_e = 0$ jinak.

Je-li $\| \varphi \|_e = 1$ ($\| \varphi \|_e = 0$), říkáme, že **formule φ je při ohodnocení e pravdivá** (**nepravdivá**).

4.1.3 tautologie

Definice

Formule VL se nazývá

- **tautologie**, je-li při každém ohodnocení pravdivá,
- **kontradikce**, je-li při každém ohodnocení nepravdivá,
- **splnitelná**, je-li pravdivá při alespoň jednom ohodnocení.

Zřejmě splnitelné formule jsou právě ty, které nejsou kontradikcemi. Fakt, že formule φ je tautologie, zapisujeme $\models \varphi$, popřípadě $\| \varphi \| = 1$.

4.1.4 tabulková metoda

Pro n výrokových symbolů $p_1, \ldots, p_n$ existuje právě 2^n různých ohodnocení symbolů $p_1, \ldots, p_n$ (každému výrokovému symbolu se přiřazuje 0 nebo 1). Tyto úvahy jsou základem tzv. tabulkové metody pro zjištění pravdivostních hodnot formule.

Tabulková metoda slouží k vypsání (tabelaci) hodnot zadaných formulí $\varphi_1, \varphi_2, \ldots, \varphi_m$ v tabulce. Tabulka má 2^n řádků a $n+m$ sloupců, kde n je počet všech výrokových symbolů, které se vyskytují ve formulích $\varphi_1, \varphi_2, \ldots, \varphi_m$. Do řádků píšeme všechna možná ohodnocení těchto symbolů a hodnoty formulí $\varphi_1, \varphi_2, \ldots, \varphi_m$.
Formule φ_i je tautologií (kontradikcí, splnitelnou), právě když jí odpovídající sloupec pravdivostních hodnot obsahuje ve všech řádcích samé 1 (samé 0, aspoň jednu 1).

4.1.5 sémantické vyplývání

Definice

Formule ψ **sémanticky plyne z formule** φ, značíme $\varphi \models \psi$, jestliže ψ je pravdivá při každém ohodnocení, při kterém je pravdivá φ. Pokud ψ sémanticky plyne z φ a naopak, říkáme, že φ a ψ jsou **sémanticky ekvivalentní**.
Obecněji, formule ψ **sémanticky plyne z množiny formulí** T, značíme $T \models \psi$, je-li ψ pravdivá při každém ohodnocení, při kterém je pravdivá každá formule z T.

Pro ověření sémantického vyplývání je možné použít tabelaci (tabulkovou metodu).

Definice

Nechť V je množina výrokových symbolů. Pak

- **literál** nad V je libovolný výrokový symbol z V nebo jeho negace
- **úplná elementární konjunkce** nad V je libovolná konjunkce literálů, ve které se každý výrokový symbol z V vyskytuje právě v jednom literálu
- **úplná elementární disjunkce** nad V je libovolná disjunkce literálů, ve které se každý výrokový symbol z V vyskytuje právě v jednom literálu
- **úplná konjunktivní normální forma** nad V je konjukce úplných elementárních disjunkcí nad V
- **úplná disjunktivní normální forma** nad V je disjunkce úplných elementárních konjunkcí nad V.

Věta

Ke každé formuli VL, která není tautologií (kontradikcí) existuje s ní sémanticky ekvivalentní formule, která je ve tvaru úplné konjunktivní normální formy (úplné disjunktivní normální formy).

Příklad

Sestrojte ÚDNF a ÚKNF k formuli φ: $(p \Leftrightarrow q) \wedge (q \Rightarrow r)$

p	q	r	$p \Leftrightarrow q$	$q \Rightarrow r$	φ	ÚEK	ÚED
1	1	1	1	1	1	$p \wedge q \wedge r$	
1	1	0	1	0	0		$\neg p \vee \neg q \vee r$
1	0	1	0	1	0		$\neg p \vee q \vee \neg r$
1	0	0	0	1	0		$\neg p \vee q \vee r$
0	1	1	0	1	0		$p \vee \neg q \vee \neg r$
0	1	0	0	0	0		$p \vee \neg q \vee r$
0	0	1	1	1	1	$\neg p \wedge \neg q \wedge r$	
0	0	0	1	1	1	$\neg p \wedge \neg q \wedge \neg r$	

Tedy ÚDNF je $(p \wedge q \wedge r) \vee (\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge \neg q \wedge \neg r)$, ÚKNF je $(\neg p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (p \vee \neg q \vee r)$.

4.1.6 normální formy formulí

4.1.7 úplné systémy spojek

Množina booleovských funkcí $\{f_1, \ldots, f_k\}$ je **funkčně úplná**, pokud každou booleovskou funkci $f : \{0,1\}^n \to \{0,1\}$ lze vyjádřit jako složení některých funkcí z $\{f_1, \ldots, f_k\}$.
Řekneme, že množina výrokových spojek je **úplná** (tvoří **úplný systém spojek**), jestliže je funkčně úplná množina jim odpovídajících booleovských funkcí.
Každý úplný minimální systém spojek VL nazveme **bází**.

Tvrzení

$\{\rightharpoonup, \curlyvee, \curlywedge\}$ tvoří úplný systém spojek VL.

Důkaz: Platnost plyne z tvrzení o ÚDNF (ÚKNF).

Z de Morganových zákonů je zřejmé, že systém $\{\rightharpoonup, \curlyvee, \curlywedge\}$ není bází. Jednoduše se dá ukázat, že existují dvouprvkové báze $\{\rightharpoonup, \curlyvee\}$, $\{\rightharpoonup, \curlywedge\}$, $\{\rightharpoonup, \to\}$.

Otázka: Existují jednoprvkové báze VL?

Speciální význam mají **Piercova** (**Nicodova**) **spojka** (význam: "ani ..., ani ..."; označujeme ji symbolem $\Downarrow$) a **Shefferova spojka** (význam: "pokud ..., pak neplatí ..."; označujeme ji symbolem $\Uparrow$), které samy o sobě tvoří úplný systém spojek.
Obě spojky jsou interpretovány následujícími pravdivostními funkcemi:

$\downarrow$	0	1
0	1	0
1	0	0

$\uparrow$	0	1
0	1	1
1	1	0

Existují pouze 2 jednoprvkové báze (Sheffer a Nicod)

4.1.8 axiomatický systém výrokové logiky

Nejprve si zavedeme nový pojem vyplývání, který nebude založen na pojmu pravdivostní ohodnocení, ale pouze na manipulaci s formulemi na úrovni jejich tvaru. Základní pojem, na kterém je tento typ vyplývání založen je **odvozovací pravidlo** – předpis pomocí nějž ze vstupních formulí odvozujeme další formule. Odvozovací pravidla formalizují elementární úsudky. Nám bude ve VL postačovat pouze jediné odvozovací pravidlo, tzv. **pravidlo odloučení** neboli **modus ponens** (MP), které lze schématicky vyjádřit

$$\text{MP:} \qquad \frac{\varphi, \varphi \Rightarrow \psi}{\psi}$$

a jehož význam je: "z formulí φ a $\varphi \Rightarrow \psi$ odvodíme formuli ψ". Formulím $\varphi, \varphi \Rightarrow \psi$ někdy říkáme **předpoklady**.

Například formule $\neg q$ vzniká použitím modus ponens z formulí $p \Rightarrow r$ a $(p \Rightarrow r) \Rightarrow \neg q$.

Při odvozování formulí budeme dále používat **axiomy**, což jsou formule, které automaticky přijímáme jako "platné". Axiomy popisují vlastnosti logických spojek a jejich vzájemný vztah. Axiomy VL si definujeme pomocí tří **axiomových schémat**:

(A1) $\varphi \Rightarrow (\psi \Rightarrow \varphi)$,

(A2) $(\varphi \Rightarrow (\psi \Rightarrow \chi)) \Rightarrow ((\varphi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow \chi))$,

(A3) $(\neg\psi \Rightarrow \neg\varphi) \Rightarrow (\varphi \Rightarrow \psi)$.

Jakákoli formule, která je ve tvaru jednoho ze schémat (A1) – (A3) se nazývá **axiom** VL.

Axiomová schémata jsou "předpisy", kterými definujeme všechny axiomy. Ačkoli budeme používat pouze tři axiomová schémata, axiomů jako takových je nekonečně mnoho. Např. formule $(\neg(p \Rightarrow q) \Rightarrow \neg\neg p) \Rightarrow (\neg p \Rightarrow (p \Rightarrow q))$ je axiom, který je instancí schéma (A3). Dále např. $p \Rightarrow (q \Rightarrow r)$ není axiom.

Množinu axiomů a odvozovacích pravidel, která používáme, souhrnně nazýváme **axiomatický systém**.

4.1.9 syntaktické výplývání

Definice

Důkaz formule φ z množiny formulí T je lib. posloupnost formulí $\varphi_1, \ldots, \varphi_n$ taková, že $\varphi_n = \varphi$ a každá φ_i $(i = 1, \ldots, n)$

- je axiomem,
- nebo náleží do T,
- nebo vzniká z předchozích formulí důkazu pomocí odvozovacího pravidla MP, tedy existují indexy $j, k < i$ tak, že φ_k je formule ve tvaru $\varphi_j \Rightarrow \varphi_i$.

Formule φ je dokazatelná z T (zapisujeme $T \vdash \varphi$), pokud existuje důkaz formule φ z T. Pokud $\vdash \varphi$, pak říkáme, že φ je dokazatelná (z prázdného systému předpokladů).

Dokazatelnosti budeme také říkat **syntaktické vyplývání**, abychom tím zdůraznili, že jde o protějšek sémantického vyplývání. Fakt $T \vdash \varphi$ lze tedy číst "φ syntakticky plyne z T", případně "φ je syntaktickým důsledkem T".

4.1.10 Věta o dedukci

Věta o dedukci (VoD)

Pro každou množinu formulí T a formule φ, ψ platí: $T \vdash \varphi \Rightarrow \psi$, právě když $T, \varphi \vdash \psi$.

Věta o dedukci umožňuje mimo jiné zkracovat důkazy.

Příklad

Ukažme, že jestliže $T \vdash \varphi \Rightarrow \psi$ a $T \vdash \psi \Rightarrow \chi$, pak $T \vdash \varphi \Rightarrow \chi$ (tzv. **princip tranzitivity implikace**). Skutečně, máme $T, \varphi \vdash \psi$ (dle VoD aplikované na $T \vdash \varphi \Rightarrow \psi$), dále $T, \varphi \vdash \chi$ (použitím MP a monotonie dokazatelnosti) a konečně $T \vdash \varphi \Rightarrow \chi$ (VoD použitá na $T, \varphi \vdash \chi$).

4.1.11 Věty o korektnosti a úplnosti výrokové logiky

Věta o korektnosti

Pro libovolnou množinu formulí T a formuli φ platí, že je-li $T \vdash \varphi$, pak $T \models \varphi$. Speciálně tedy, každá dokazatelná formule je tautologií.

Poznámka: Korektnost lze využít k prokázání faktu, že některá formule není dokazatelná z jistého systému předpokladů. Reformulací korektnosti totiž dostáváme, že pokud φ sémanticky neplyne z T, pak φ není ze systému T ani dokazatelná. K tomu, abychom prokázali, že $T \nvdash \varphi$ tedy stačí ukázat $T \not\models \varphi$, což je výrazně jednodušší než prokázat "neexistenci důkazu", protože důkazů, jakožto konečných posloupností formulí, je obecně nekonečně mnoho.

Věta o úplnosti, silná verze

Pro libovolnou množinu T formulí a formuli φ platí, že z $T \models \varphi$ plyne $T \vdash \varphi$.

4.2 Predikátová logika

4.2.1 jazyk

Definice

Jazyk $\mathscr{J}$ **PL** obsahuje (a je tím určen)

- (**předmětové**) **proměnné** $x, y, z, \ldots, x_1, x_2, \ldots$
- **relační symboly** $p, q, r, \ldots, p_1, p_2 \ldots$, ke každému relačnímu symbolu r je dáno nezáporné celé číslo $\sigma(r)$ nazývané arita symbolu r; musí existovat alespoň jeden relační symbol
- **funkční symboly** $f, g, h, \ldots, f_1, f_2, \ldots$ ke každému funkčnímu symbolu f je dáno nezáporné celé číslo $\sigma(f)$ nazývané arita symbolu f
- **symboly pro logické spojky** $\neg$ (negace) a $\Rightarrow$ (implikace)
- **symbol pro univerzální kvantifikátor** $\forall$
- **pomocné symboly** – různé typy závorek a čárka.

Množina všech relačních (někdy se říká predikátových) symbolů jazyka $\mathscr{J}$ se značí R; množina všech funkčních (někdy se říká operačních) symbolů jazyka $\mathscr{J}$ se značí F. Je-li $r \in R$ a $\sigma(r) = n$, pak říkáme, že r je **n-ární**. Podobně pro $f \in F$. Je-li $f \in F$ 0-ární, nazývá se f **symbol konstanty** (neboť funkce, která má 0 argumentů, musí přiřazovat vždy stejnou hodnotu, tj. je konstantní).

Je zřejmé, že jazyk je jednoznačně určen svými relačními symboly, funkčními symboly a jejich aritami (vše ostatní mají všechny jazyky stejné). Trojici $\langle R, F, \sigma \rangle$ proto nazýváme **typ jazyka**. (Pochopitelně předpokládáme, že $R \cap F = \emptyset$.)

4.2.2 termy a formule

Základní syntaktické jednotky vybudované ze symbolů jazyka PL jsou termy a formule. Termy jsou výrazy reprezentující funkci aplikovanou na své operandy; formule reprezentují tvrzení o prvcích univerza.

Definice

Term jazyka typu $\langle R, F, \sigma \rangle$ je induktivně definován takto:

(i) každá proměnná x je term

(ii) je-li $f \in F$ n-ární a jsou-li $t_1, \ldots, t_n$ termy, pak $f(t_1, \ldots, t_n)$ je term.

Termy jsou tedy jisté konečné posloupnosti prvků daného jazyka. Je-li $f \in F$ binární, používáme také tzv. infixovou notaci a píšeme $x\, f\, y$ nebo $(x\, f\, y)$ místo $f(x, y)$, např. $2+3$ místo $+(2,3)$; ve složených termech používáme závorky, např. $(2+3)\cdot 5$.

Definice

Formule jazyka typu $\langle R, F, \sigma \rangle$ je induktivně definována takto:

(i) je-li $r \in R$ n-ární a jsou-li $t_1, \ldots, t_n$ termy, pak $r(t_1, \ldots, t_n)$ je formule

(ii) jsou-li φ a ψ formule, pak $\neg\varphi$, $(\varphi \Rightarrow \psi)$ jsou také formule

(iii) je-li φ formule a x proměnná, pak $(\forall x)\varphi$ je formule.

Formule vytvořené dle (i) se nazývají **atomické**. Je-li $r \in R$ binární, píšeme také $t_1\, r\, t_2$ nebo $(t_1\, r\, t_2)$ místo $r(t_1, t_2)$, tedy např. $x \leq y$ místo $\leq (x, y)$. Obzvláště píšeme $t_1 \approx t_2$ místo $\approx (t_1, t_2)$.

4.2.3 struktury pro jazyk

Definice

Struktura pro jazyk typu $\langle R, F, \sigma \rangle$ je trojice $\mathbf{M} = \langle M, R^{\mathbf{M}}, F^{\mathbf{M}} \rangle$, která sestává z neprázdné množiny M a dále z množin

$$R^{\mathbf{M}} = \{ r^{\mathbf{M}} \subseteq M^n \mid r \in R, \sigma(r) = n \},$$

$$F^{\mathbf{M}} = \{ f^{\mathbf{M}} : M^n \to M \mid f \in F, \sigma(f) = n \}.$$

Pokud $\approx \in R$, pak $\approx$ interpretujeme vždy relací identity, tj. $\approx^{\mathbf{M}} = \omega_M = \{ \langle u, u \rangle \mid u \in M \}$.

Jinými slovy, struktura $\mathbf{M}$ pro jazyk typu $\langle R, F, \sigma \rangle$ je systém relací a funkcí na jisté množině M, přitom ke každému n-árnímu relačnímu symbolu $r \in R$ je ve struktuře $\mathbf{M}$ odpovídající n-ární relace $r^{\mathbf{M}} \in R^{\mathbf{M}}$ na M a ke každému n-árnímu funkčnímu symbolu $f \in F$ je ve struktuře $\mathbf{M}$ odpovídající n-ární funkce $f^{\mathbf{M}} \in F^{\mathbf{M}}$ v M. Nehrozí-li nebezpečí nedorozumění, budeme někdy vynechávat horní indexy a místo $r^{\mathbf{M}}$ a $f^{\mathbf{M}}$ budeme psát jen r a f.

4.2.4 ohodnocení termu a formulí

Nechť **M** je struktura pro jazyk typu $\langle R, F, \sigma \rangle$. **M-ohodnocení proměnných** (krátce jen **M-ohodnocení**, popř. jen **ohodnocení**) je zobrazení v přiřazující každé proměnné x prvek $v(x) \in M$. Jsou-li v a v' ohodnocení a x je proměnná, píšeme $v =_x v'$ pokud pro každou proměnnou $y \neq x$ je $v(y) = v'(y)$, tj. v a v' se liší nejvýše v tom, jakou hodnotu přiřazují proměnné x.

Definice

Nechť v je **M**-ohodnocení. **Hodnota $\| \mathbf{t} \|_{\mathbf{M},\mathbf{v}}$ termu t v M při v** je definována

$$\| t \|_{\mathbf{M},v} = \begin{cases} v(x), & \text{je-li } t \text{ proměnná } x \\ f^{\mathbf{M}}(\| t_1 \|_{\mathbf{M},v}, \ldots, \| t_k \|_{\mathbf{M},v}), & \text{je-li } t \text{ tvaru } f(t_1, \ldots, t_k). \end{cases}$$

Uvědomme si, že při dané struktuře **M** a při daném **M**-ohodnocení v je každému termu t přiřazena právě jedna hodnota $\| t \|_{\mathbf{M},v}$ z univerza M. Dále je patrné, že hodnota $\| t \|_{\mathbf{M},v}$ nezávisí na hodnotách přiřazených ohodnocením v těm proměnným, které se v t nevyskytují (lze dokázat jednoduše strukturální indukcí).

Definice

Pravdivostní hodnota $\| \varphi \|_{\mathbf{M},v}$ **formule** φ **při M-ohodnocení v** je definována následovně:

(i) pro atomické formule

$$\| r(t_1,\ldots,t_n) \|_{\mathbf{M},v} = \begin{cases} 1, & \text{je-li } \langle \| t_1 \|_{\mathbf{M},v},\ldots,\| t_n \|_{\mathbf{M},v} \rangle \in r^{\mathbf{M}} \\ 0, & \text{jinak} \end{cases}$$

(ii) pro formule φ ve tvaru $\neg\alpha$ a $\alpha \Rightarrow \beta$

$$\| \neg\alpha \|_{\mathbf{M},v} = \begin{cases} 1, & \text{pokud } \| \alpha \|_{\mathbf{M},v} = 0 \\ 0, & \text{pokud } \| \alpha \|_{\mathbf{M},v} = 1 \end{cases}$$

$$\| \alpha \Rightarrow \beta \|_{\mathbf{M},v} = \begin{cases} 1, & \text{pokud } \| \alpha \|_{\mathbf{M},v} = 0 \text{ nebo} \\ & \| \beta \|_{\mathbf{M},v} = 1 \\ 0, & \text{jinak} \end{cases}$$

(iii) pro kvantifikovanou formuli φ

$$\| (\forall x)\varphi \|_{\mathbf{M},v} = \begin{cases} 1, & \text{pokud pro každé } v' \text{ takové, že} \\ & v' =_x v \text{ je } \| \varphi \|_{\mathbf{M},v'} = 1 \\ 0, & \text{jinak.} \end{cases}$$

Je-li $\| \varphi \|_{\mathbf{M},v} = 1$ ($\| \varphi \|_{\mathbf{M},v} = 0$), říkáme, že formule φ je **pravdivá** (**nepravdivá**) **ve struktuře M při ohodnocení v**.

4.2.5 axiomatický systém predikátové logiky

Axiomy jsou formule tvaru (A1) – (A5)

(A1) $\varphi \Rightarrow (\psi \Rightarrow \varphi)$,

(A2) $(\varphi \Rightarrow (\psi \Rightarrow \chi)) \Rightarrow ((\varphi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow \chi))$,

(A3) $(\neg\psi \Rightarrow \neg\varphi) \Rightarrow (\varphi \Rightarrow \psi)$,

(A4) $(\forall x)\varphi \Rightarrow \varphi(x/t)$,
je-li t substituovatelný za x,

(A5) $(\forall x)(\varphi \Rightarrow \psi) \Rightarrow (\varphi \Rightarrow (\forall x)\psi)$,
nemá-li x ve φ volný výskyt,

kde φ, χ, ψ jsou formule PL, t je term a x je proměnná. (A4) se nazývá **axiom specifikace** (**substituce**), (A5) se nazývá **axiom distribuce**.

Odvozovací pravidla jsou **modus ponens** (MP), a **pravidlo generalizace** (G) (též **pravidlo zobecnění**), které říká z φ odvoď $(\forall x)\varphi$.

4.2.6 syntaktické vyplývání

Definice

Množina S formulí **sémanticky plyne** z množiny T formulí (píšeme $T \models S$; píšeme také $T \models \varphi$, jestliže $S = \{\varphi\}$, podobně když $T = \{\psi\}$), jestliže každý model T je modelem S.

Tedy $T \models S$, právě když v každé struktuře, ve které jsou pravdivé všechny formule z T, jsou také pravdivé všechny formule z S.

Definice

Důkaz formule φ z množiny formulí T je libovolná posloupnost formulí $\varphi_1, \ldots, \varphi_n$, pro kterou platí, že $\varphi_n = \varphi$ a každá φ_i (pro $i \leq n$)

- je axiomem PL
- nebo je formulí z T (je axiomem z T)
- nebo plyne z předchozích formulí důkazu pomocí MP nebo odvozovacího pravidla G (tj. existuje $j < i$ tak, že φ_i je formule $(\forall x)\varphi_j$).

Formule se nazývá **dokazatelná z T** (**věta teorie T**), existuje-li důkaz této formule z T (zapisujeme $T \vdash \varphi$, popř. jen $\vdash \varphi$, je-li $T = \emptyset$).

Formule je dokazatelná z T = formule syntakticky plyne z T.

4.2.7 věty o korektnosti a úplnosti predikátové logiky

Věta o korektnosti (VoK)

Pro libovolnou teorii T a libovolnou formuli φ jazyka teorie T platí, že z $T \vdash \varphi$ plyne $T \models \varphi$.

Důkaz: Analogicky jako ve VL.

Poznámka: Jednoduchým důsledkem je fakt: sporná teorie nemá model. Totiž byla-li by T sporná, pak pro každou formuli φ by platilo $T \vdash \varphi$ i $T \vdash \neg\varphi$. Dle VoK by muselo být v každém modelu teorie T pravdivé φ i $\neg\varphi$, což není možné.

Věta o úplnosti (VoÚ)

(1) Každá bezesporná teorie má model.

(2) Pro každou teorii T a každou formuli φ platí, že je-li $T \models \varphi$, pak $T \vdash \varphi$.

4.3 Neklasické logiky

- Modální logika (založena na pojmu „možný svět“, spojky: *je možné že, je nutné že*)
- Temporální logika (logika času - pravdivost tvrzení závisí na čase)
- Epistemická logika (logika znalostí, spojky: *ví se že, věří se že*)
- Fuzzy logika

4.3.1 fuzzy logika

Klasická logika (VL a PL) nestačí při modelování tzv. vágních tvrzení, např. „Petr je silný.“, „Teplota je vysoká.“, „Zákazník je spokojený.“. Uvedená tvrzení často (intuitivně) nepovažujeme za ani úplně nepravdivá, ani úplně pravdivá, tj. za tvrzení, jejichž pravdivostní hodnota leží mezi 0 a 1, např. je to $0,9$ (skoro pravda), $0,5$ (napůl pravda), $0,1$ (skoro nepravda).
S vágními tvrzeními se setkáváme téměř při každém popisu reálného světa. Jedná se tedy o netriviální a širokou oblast.

Množinu pravdivostních hodnot budeme značit L. Požadujeme, aby $0,1 \in L$ a aby L byla částečně uspořádána relací $\leq$.
Například tedy $L = [0,1]$; $L = \{0,1\}$ (klasická logika);
$L = \{0,1\} \times \{0,1\}$ (nelineární logika).

Musí existovat operace na L modelující logické spojky (zejména $\otimes$ pro konjunkci a $\rightarrow$ pro implikaci). Tyto operace by měly mít přirozené vlastnosti odpovídající vlastnostem požadovaným po logických spojkách (například komutativita $\otimes$, monotónnost $\otimes$ apod.).
Dále, aby ve FL „dobře fungovalo“ pravidlo MP, je potřeba, aby:
$a \otimes b \leq c \Leftrightarrow a \leq b \rightarrow c$.

Výše uvedené požadavky na strukturu pravdivostních hodnot (a některé neuvedené) vedou k jedné ze základních struktur pravdivostních hodnot ve FL, k tzv. reziduovaným svazům – viz následující definici.

Definice

Úplný reziduovaný svaz je struktura $\mathscr{L} = (L; \wedge, \vee, \otimes, \rightarrow, 0, 1)$, kde

(1) $(L; \wedge, \vee, 0, 1)$ je úplný svaz (s nejmenším prvkem 0 a největším prvkem 1)

(2) $(L; \otimes, 1)$ je komutativní monoid (tj. $\otimes$ je binární operace na L, která je komutativní, asociativní a platí $a \otimes 1 = a$)

(3) $\otimes, \rightarrow$ jsou binární operace na L (nazývané „**násobení**“ a „**reziduum**“) splňující tzv. **podmínku adjunkce**:

$$a \otimes b \leq c \quad \text{právě když} \quad a \leq b \rightarrow c.$$

Mezi nejčastěji používané struktury pravdivostních hodnot patří ty, které mají za nosič reálný interval $[0, 1]$ s přirozeným uspořádáním, tedy $a \wedge b = \min(a, b)$, $a \vee b = \max(a, b)$. Na nich se používají tři páry adjungovaných operací $\otimes$ a $\rightarrow$:

(I) **Łukasiewiczovy operace:**
$a \otimes b = \max(a + b - 1, 0), \qquad a \rightarrow b = \min(1 - a + b, 1)$

(II) **Gödelovy operace:**
$$a \otimes b = \min(a, b), \qquad a \rightarrow b = \begin{cases} 1, & \text{pro } a \leq b, \\ b, & \text{jinak} \end{cases}$$

(III) **součinové operace:**
$$a \otimes b = a \cdot b, \qquad a \rightarrow b = \begin{cases} 1, & \text{pro } a \leq b, \\ b/a, & \text{jinak.} \end{cases}$$

4.4 Základy logického programování, úvod do Prologu

Logické programování je jedním z paradigmat programování. Uveďme nyní základní rysy, kterými se logické programování zásadně odlišuje od ostatních programovacích paradigmat:

(1) programátor specifikuje, **co** se má vypočítat, a ne **jak** se to má vypočítat a **kam** uložit mezivýsledky

(2) Prolog nemá příkazy pro řízení běhu výpočtu ani pro řízení toku dat, nemá příkazy cyklů, větvení, přiřazovací příkaz

(3) neexistuje rozdělení proměnných na vstupní a výstupní

(4) nerozlišuje se mezi daty a programem.

Prolog je interpretační (neprocedurální) jazyk. Patří mezi deklarativní programovací jazyky – potlačuje imperativní složku. (Připomeňme, že imperativní paradigma popisuje **jak** vyřešit problém, deklarativní paradigma popisuje **co** je problém.)

Prolog je využíván především v oboru umělé inteligence a v počítačové lingvistice (obzvláště zpracování přirozeného jazyka, pro nějž byl původně navržen).

Prolog je založen na PL (prvního řádu); konkrétně se omezuje na Hornovy klauzule. Základními využívanými přístupy jsou unifikace, rekurze a backtracking.

Základní koncepci logického programování vyjadřuje následující dvojice „rovností“:

$$\begin{aligned} \text{program} &= \text{množina axiomů} \\ \text{výpočet} &= \text{konstruktivní důkaz uživatelem zadaného cíle,} \end{aligned}$$

nebo volněji: program je souborem tvrzení, kterými programátor (uživatel, expert) popisuje určitou část okolního světa; výpočet nad daným programem, který je iniciován zadáním dotazu, je hledání důkazu dotazu z daného souboru tvrzení.

Shrňme a doplňme základní rysy logického programování:

- logický program je konečná množina formulí speciálního tvaru
- výpočet je zahájen zadáním formule – dotazu (tu zadá uživatel)
- cílem výpočtu je najít důkaz potvrzující, že dotaz logicky vyplývá (je dokazatelný) z logického programu (konstruktivnost)
- pokud je takto zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno „Yes“ s hodnotami případných proměnných, které se v dotazu vyskytují
- pokud není zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno „No“
- může se stát, že výpočet neskončí.

Poznámka: Základem Prologu je databáze klauzulí, které lze dále rozdělit na fakta a pravidla, nad kterými je možno klást dotazy formou tvrzení, u kterých Prolog zhodnocuje jejich pravdivost (dokazatelnost z údajů obsažených v databázi). Nejjednoduššími klauzulemi jsou fakta, která pouze vypovídají o vlastnostech objektu nebo vztazích mezi objekty. Složitějšími klauzulemi jsou pravidla, která umožňují (pomocí implikace) odvozovat nová fakta. Zapisují se ve tvaru hlavička :- tělo, kde hlavička definuje odvozovaný fakt, tělo podmínky, za nichž je pravdivý, obsahuje jeden či více cílů. Pokud se interpretu podaří odvodit, že tělo je pravdivé, ověřil tím pravdivost hlavičky.

Definice

Literál je libovolná atomická formule (tzv. pozitivní literál) nebo negace atomické formule (tzv. negativní literál). **Klauzule** je libovolná disjunkce literálů. **Hornovská klauzule** je klauzule, ve které se vyskytuje maximálně jeden pozitivní literál. Symbolem $\square$ se označuje prázdná klauzule (tj. klauzule obsahující 0 literálů).

Poznámka: $\square$ je v logickém programování symbolem sporu.

Poznámka: Klauzule $L_1 \vee \cdots \vee L_n$ (L_i jsou literály) se v logickém programování často zapisují jako $\{L_1, \ldots, L_n\}$; zde čárky znamenají disjunkci. Pak $\square$ se značí $\{\}$.

Jak brzy uvidíme, pracuje Prolog následovně: Je-li dán logický program P a zadá-li uživatel dotaz G (popř. obecněji $G_1, \ldots, G_n$), překladač Prologu přidá k P negaci dotazu, tj. přidá $\neg G$ a snaží se z $P, \neg G$ (to je vlastně množina formulí) odvodit (tzv. rezoluční metodou) spor, tj. odvodit $\square$. Dá se dokázat, že G sémanticky vyplývá z P ($P \models G$), právě když je z $P, \neg G$ odvoditelná $\square$. Oznámí-li prologovský překladač po zadání dotazu G na program P odpověď „Yes“, znamená to právě, že překladač odvodil z $P, \neg G$ klauzuli $\square$.

Hornovská klauzule má jeden z následujících tvarů:

(a) (nenulový počet negativních literálů, právě jeden pozitivní literál) $A \vee \neg B_1 \vee \cdots \vee \neg B_n$ (A, B_i jsou atomické formule); tato klauzule je ekvivalentní formuli $B_1 \wedge \cdots \wedge B_n \Rightarrow A$ (Proč? Uvědomme si, že $B \Rightarrow A$ je ekvivalentní $A \vee \neg B$ a že $\neg(B_1 \wedge \cdots \wedge B_n)$ je ekvivalentní $\neg B_1 \vee \cdots \vee \neg B_n$.) Tyto klauzule odpovídají prologovským pravidlům.

(b) (nulový počet negativních literálů, právě jeden pozitivní literál) A. Tyto klauzule odpovídají prologovským faktům.

(c) (nenulový počet negativních literálů, žádný pozitivní literál) $\neg C_1 \vee \cdots \vee \neg C_n$ (C_i jsou atomické formule); tato klauzule, respektive její obecný uzávěr $\forall(\neg C_1 \vee \cdots \vee \neg C_n)$ je ekvivalentní formuli $\neg\exists(C_1 \wedge \cdots \wedge C_n)$ (Proč? Uvědomme si, že $\neg(\exists x)\varphi$ je ekvivalentní $(\forall x)\neg\varphi$.) Tyto klauzule odpovídají prologovským dotazům.

Definice

Logický program (někdy **definitní logický program**) je konečná množina hornovských klauzulí s jedním pozitivním literálem (tj. klauzulí odpovídajících pravidlům a faktům).

Zbytek prologu už je nechutně hnusnej a radši chcípnu než abych se to zas učil.