1. Regulární jazyky

Formální jazyky a automaty

Jiří Balun

Obsah

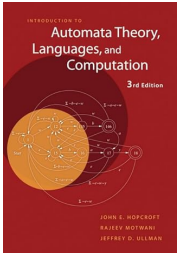
Úvod

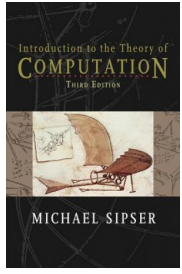
Organizační záležitosti

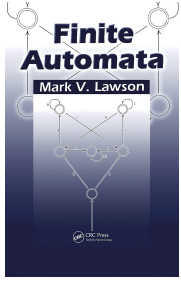
- **konzultační hodiny (kancelář 5.044):**
  - Úterý 15:00-16:00
  - Pátek 13:00-14:00
  - případně v jiný čas po domluvě emailem
- **cvičení:**
  - budeme řešit úkoly z materiálů dostupných na mém webu
- **zápočet:**
  - zápočet se skládá ze dvou částí, tj. dvou zápočtových testů
  - z každé části/testu potřebujete získat alespoň 50% bodů
  - každý test si budete moci jednou opravit
  - testy se budou psát v rámci přednášek
- **zkouška:**
  - ústní, příprava + cca 20 minut zkoušení
  - losujete 2 otázky – jedna na teorii, druhá na důkaz
  - více se dozvíte na konci semestru

Literatura

- Introduction to Automata Theory, Languages, and Computation (Hopcroft, Motwani, Ullman, 2006)
- Formální jazyky a automaty I (Černá, Křetínský, Kučera, 2006)
  - odkaz: https://is.muni.cz/elportal/?id=703389
- Introduction to the Theory of Computation (Sipser, 2012)
- Finite Automata (Lawson, 2003)







O kurzu 'Formální jazyky a automaty'

- jeden ze základních oborů teoretické informatiky
- je součástí většiny informaticky zaměřených oborů (MFF UK, ČVUT, VUT,...)
- slouží jako úvod pro další předměty jako jsou Vyčíslitelnost a Složitost
- za tento předmět je 6 kreditů, čemuž odpovídá jeho náročnost
- je to teoretický předmět, většinou si vystačíme s tužkou a papírem
- možné jako téma bakalářské/diplomové/disertační práce

Motivace

Na data a programy můžeme pohlížet jako množiny řetězců

- data a programy v počítači jsou reprezentovány pomocí posloupností symbolů 0 a 1
- přirozená motivace studovat jejich vlastnosti a metody jak s nimi pracovat

Studium výpočetních modelů

- jak silný model potřebujeme k řešení různých problémů
- začneme u těch jednoduchých a postupně budeme přecházet ke komplexnějším

Modely, které budeme studovat mají spoustu reálných aplikací

- regulární výrazy, překladače, modelování diskrétních systémů,...

Naučíme se, jak formálně zacházet s diskrétními modely

- běžně nedokazujeme, že program je korektní – je to příliš složité
- potřebujeme ale uvažovat o tom, proč funguje nějaká chytrá technika nebo konstrukce
- důkazy v tomto kurzu jsou často konstruktivní, tzn. samotný důkaz je návod/algoritmus, jak daný problém vyřešit

Obsah kurzu

Regulární jazyky

- regulární výrazy a příklady jejich použití
- konečné automaty, jejich varianty a ekvivalence jednotlivých modelů
- minimalizace automatů
- vlastnosti regulárních jazyků

Bezkontextové jazyky

- bezkontextové gramatiky a jejich normální formy
- zásobníkové automaty
- vlastnosti bezkontextových jazyků

Ostatní jazyky z Chomského hierarchie

- přehledově kontextové gramatiky a gramatiky bez omezení

Základní pojmy

Řešitelnost problémů

Motivace: při řešení reálných problému si dříve či později položíte otázku: *„Je daný problém vůbec řešitelný?“* a poté *„Co to znamená, že je problém řešitelný?“*.

Definice

Problém je **řešitelný**, pokud existuje algoritmus, který pro každou instanci daného problému vrátí v konečném čase správný výsledek.

- problémy s odpovědí *ano*/*ne* označujeme jako **rozhodovací problémy**
- zavedeme si základní pojmy, proto abychom mohli problémy formálně definovat a studovat

Příklad

Během studia jste se setkali s mnoha řešitelnými problémy:

- rozhodovací problémy *„je daná sekvence čísel setřízená?“* nebo *„je strom vyvážený?“*,
- obecné problémy jako výpočet n-té mocniny nějakého čísla, a tak dále...

V tomto kurzu se setkáme i s problémy, které nelze algoritmicky vyřešit.

Abeceda

Intuice: abecedu budeme chápat v jejím přirozeném významu, tedy jako soubor znaků, které mohou být použity k vytvoření slov/řetězců.

Definice

Abeceda Σ je libovolná konečná neprázdná množina.

- prvky abecedy se nazývají **znaky**, případně **symboly**
- spousta příkladů ze života:
  - latinské písmo $\{a, b, c, \ldots, z\}$, japonské písmo $\{ア, キ, ラ, \ldots\}$, atd.
  - binární abecedy $\{0, 1\}$ nebo $\{\texttt{true}, \texttt{false}\}$
  - znaková sada ASCII $\{0, \ldots, 9, a, \ldots, z, A, \ldots, Z, \texttt{NULL}, \texttt{LF}, \ldots\}$
  - množina klíčových slov jazyka C: $\{\texttt{if}, \texttt{for}, \texttt{while}, \ldots\}$
  - symboly výrokové logiky $\{\neg, \vee, \wedge, (,), \varphi, \psi, \ldots\}$
  - příznaky zpráv při navazování TCP komunikace $\{\texttt{ACK}, \texttt{SYN}, \texttt{FIN}, \texttt{ACK-SYN}\}$
  - a mnoho dalších příkladů...
- každou abecedu lze zakódovat pomocí binarní abecedy

Řetězec

Definice

Řetězec $w = w_1 \dots w_n$ nad abecedou Σ je libovolná konečná posloupnost znaků této abecedy, tj. $w \in \Sigma^*$ (platí $w_i \in \Sigma$ pro všechna $i = 1, \dots, n$).

- délka řetězce $w = w_1 \dots w_n$ se značí $|w| = n$
- **prázdný řetězec** se značí ε a platí pro něj $|\varepsilon| = 0$
- řetězec x je podřetězcem y, pokud existují $a, b \in \Sigma^*$ takové, že $y = axb$
- Σ^* označuje množinu všech řetězců nad abecedou Σ
- $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ označuje množinu všech neprázdných řetězců nad Σ

Příklad

Mějme binární abecedu $\Sigma = \{0, 1\}$, pak:

- pro řetězec $w = 101010 \in \Sigma^*$ je jeho délka $|w| = 6$,
- $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$,
- $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, \dots\}$.

Uspořádání řetězců

Intuice: řetězce můžeme seřadit pomocí tzv. **shortlex** uspořádání, kde kratší řetězce jsou před delšími, a stejně dlouhé řetězce jsou uspořádány relací na Σ (například lexikograficky).

Definice

Mějme abecedu Σ a lineární uspořádání $<_\Sigma$ prvků ze Σ (například pro $\Sigma = \{a, \ldots, z\}$ použijeme klasické abecední pořadí znaků). Pro dva různé řetězce $x = x_1 x_2 \ldots x_n$ a $y = y_1 y_2 \ldots y_m$ nad Σ platí $x \prec y$ (tedy x je menší v **shortlex** uspořádání než y) pokud:

1. $|x| < |y|$, nebo
2. $|x| = |y|$ a existuje i takové, že $i = \min\{j \mid x_j \neq y_j\}$ a zároveň $x_i <_\Sigma y_i$.

Příklad

Mějme abecedu $\Sigma = \{a, b, c\}$ s klasickým uspořádáním $a <_\Sigma b <_\Sigma c$, pak například $aa \prec ab$ nebo $c \prec aaa$. Můžeme také vypsat několik prvních řetězců z nekonečné množiny Σ^*:

$$\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \ldots\}$$

Důležitý důsledek existence uspořádání na Σ^* je, že Σ^* je **spočetně nekonečná množina**.

Jazyk

Definice

Jazyk nad abecedou Σ je libovolná podmnožina Σ^*. Pojmem **formální jazyk** myslíme nějakou množinu řetězců, které navíc mají určitou společnou vlastnost.

- **problém náležení do jazyka:** chceme rozhodnout zda řetězec patří do daného jazyka
- jazyky budeme dělit do tříd podle toho, jak silný model potřebujeme pro jejich rozhodování/generování

Základní dělení

1. konečné jazyky:
   - například množina slov českého jazyka (přibližně 250 000 slov)
   - je jednoduché ověřit zda nějaký řetězec do takového jazyka patří
   - reprezentace pomocí výčtu prvků
   - z našeho pohledu triviální a nezajímavé
2. nekonečné jazyky:
   - například množina všech syntakticky korektních programů jazyka C
   - ověřit náležení do takového jazyka může být velice těžké nebo dokonce nemožné
   - reprezentace pomocí nějakého verifikátoru/generátoru (pokud je to možné)

Příklady: formální jazyky

Příklad

Trivialní jazyky nad libovolnou abecedou Σ jsou $\emptyset$ (nejmenší) a Σ^* (největší).

Příklad

Konečný jazyk $L_1 = \{\varepsilon\}$ obsahující pouze prázdný řetězec (nad libovolnou abecedou).

Příklad

Jazyk $L_2 = \{w \mid w \text{ obsahuje podřetězec } \texttt{print("Hello world!")}\}$ nad abecedou ASCII.

Příklad

Jazyk $L_3 = \{a^n b^n \mid n \geq 0\}$ nad dvouprvkovou abecedou $\Sigma = \{a, b\}$.

- jazyky $\emptyset$, Σ^*, L_1 a L_2 lze popsat pomocí *regulárních výrazů* a rozhodovat problém náležení poměrně jednoduchým modelem tzv. *konečných automatů* (více později)
- rozhodování problému náležení do L_3 je komplikovanější, potřebujeme počítadlo pro n

Konečná reprezentace jazyka

Motivace: z praktického hlediska jsou nekonečné jazyky smysluplné, jen pokud je dovedeme v počítači uložit pomocí nějaké konečné reprezentace.

Definice

Mějme konečnou reprezentaci R nějakého jazyka K, pak $L(R)$ označíme jazyk, který R reprezentuje, tj. $L(R) = K$.

- konečná reprezentace jazyka (v počítači zakódovaná binárně) může být například:
  - výpočetní model/algoritmus, který rozhoduje daný jazyk (například konečný automat)
  - nějaký zkrácený formální popis (například regulární výraz nebo gramatika)
  - algoritmus, který na základě vstupů vygeneruje řetězec daného jazyka, atd. . .
  - v průběhu semestru si ukážeme několik výpočetních modelů/reprezentací jazyka
- řešitelné problémy můžeme zakódovat do jazyků (většinou nad rámec tohoto kurzu)
  - pro rozhodovací problémy stačí vytvořit jazyk těch instancí, pro které algoritmus rozhodující daný problém vrátí odpověď *ano*
  - v případě obecných problémů vytvoříme jazyk dvojic $\langle$*instance*, *korektní výstup*$\rangle$
  - umět rozhodnout o náležení do takového jazyka potom řeší původní problém

Příklady: reprezentace problému formálním jazykem

Příklad

Problém ověření emailové adresy lze definovat jako jazyk L_1 korektně zapsaných adres nad abecedou ASCII, kde například $john.doe@upol.cz \in L_1$, ale $john@doe@upol@cz \notin L_1$.

Příklad

Výpočet druhé mocniny lze definovat jako jazyk L_2 nad abecedou $\Sigma = \{0, 1, \ldots, 9\}$

$$L_2 = \{\langle a, b\rangle \mid \text{číselná hodnota } a \in \Sigma^* \text{ je druhá mocnina číselné hodnoty } b \in \Sigma^*\},$$

kde například $\langle 25, 5\rangle \in L_2$, protože $25 = 5^2$, ale $\langle 42, 0\rangle \notin L_2$; nelze rozhodovat bez nějakého výpočtu druhé mocniny, proto na rozhodování potřebujeme už hodně silný model.

Příklad

Jazyk programů jazyka C nad abecedou `ASCII`, který nelze rozhodovat:

$$HALT_C = \{\langle c, w\rangle \mid c \text{ je zdrojový kód jazyka C, který pro vstup } w \text{ zastaví}\}.$$

Operace s řetězci

Definice

Pro libovolné řetězce $x = x_1 \dots x_n$ a $y = y_1 \dots y_m$ definujeme

- binární operaci **zřetězení** $x \cdot y$ (nebo zkráceně xy):

$$x \cdot y = xy = x_1 \dots x_n y_1 \dots y_m$$

- unární operaci n**-té mocniny** x^n:

$$x^n = \begin{cases} \varepsilon & \text{pro } n = 0 \\ x \cdot x^{n-1} & \text{pro } n > 0 \end{cases}$$

Příklad

Pro řetězce $x = pi$, $y = ka$ a $z = chu$

- $x \cdot y \cdot z = pikachu$
- $x^3 = pipipi$
- $(xy)^2 = pikapika$

Operace s jazyky

Definice

Pro libovolné jazyky L, L_1 a L_2 definujeme

- klasické množinové operace průnik, sjednocení, doplněk (vzhledem k Σ^*), ...
- binární operaci **zřetězení jazyků** $L_1 \cdot L_2$:

$$L_1 \cdot L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$

- unární operaci n**-té mocniny** L^n:

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

- unární operace **Kleeneho uzávěr** L^* a **pozitivní uzávěr** L^+:

$$L^* = \bigcup_{n=0}^{\infty} L^n \qquad L^+ = \bigcup_{n=1}^{\infty} L^n$$

Operace s jazyky – příklady

Příklad

Pro jazyky $L_1 = \{0, 00\}$, $L_2 = \{a, ab\}$ máme

$$\begin{aligned} L_1 \cdot L_2 &= \{0a, 00a, 0ab, 00ab\} \\ L_2 \cdot L_1 &= \{a0, a00, ab0, ab00\} \\ L_1^3 &= \{0^3, 0^4, 0^5, 0^6\} \\ L_1^* &= \{\varepsilon, 0, 00, 000, \ldots\} = \{0\}^* \\ L_2^+ &= \{aa, aaa, aab, aba, aaaa, aaab, aaba, abaa, abab, \ldots\} \end{aligned}$$

Příklad

Mějme jazyk L_3 nad abecedou $\Sigma = \{a\}$ definován jako $L_3 = \{a^p \mid p \text{ je prvočíslo}\}$, pak

$$\begin{aligned} L_3 \cdot \{aa\} &= \{a^{p+2} \mid p \text{ je prvočíslo}\} \\ L_3^2 &= \{a^\ell \mid \ell \text{ je násobkem dvou prvočísel}\} \end{aligned}$$

Regulární jazyky (RL) a regulární výrazy (RE)

Motivační příklad pro regulární výrazy

- regulární výraz je formalismus pro čitelný a úsporný zápis regulárních jazyků
- mnoho aplikací: `grep`, `sed`, ověřování formulářů, nahrazování řetězce v kódu atd.
- představil je v roce 1951 Stephen Cole Kleene

Příklad

Chceme úsporně zapsat jazyk $L_{sfx} = \{w \mid w$ má příponu `.pdf`, `.png`, nebo `.gz`$\}$ nad abecedou $\Sigma = \{., a, \ldots, z\}$. Nejprve jej vyjádříme pomocí zřetězení a Kleeneho uzávěru:

$$\{., a, \ldots, z\}^* \cdot \{.pdf, .png, .gz\}$$

což je ekvivalentní s výrazem:

$$(\{.\} \cup \{a\} \cup \ldots \cup \{z\})^* \cdot (\{.pdf\} \cup \{.png\} \cup \{.gz\})$$

na kterém lze vidět podobnost s ekvivalentním regulárním výrazem:

$$(. + \boldsymbol{a} + \ldots + \boldsymbol{z})^*(\boldsymbol{.pdf} + \boldsymbol{.png} + \boldsymbol{.gz})$$

Regulární výraz

Intuice: v induktivní definici nejdříve definujeme regulární výrazy pro elementární jazyky (bod 1 až 3), z nichž vytváříme složitější výrazy pomocí základních operací (bod 4 až 6).

Definice

Regulární výraz (RE z *regular expression*) nad abecedou Σ definujeme jako:

1. každý symbol $a \in \Sigma$ je $\boldsymbol{a}$ RE s jazykem $L(\boldsymbol{a}) = \{a\}$,
2. $\boldsymbol{\varepsilon}$ je RE s jazykem $L(\boldsymbol{\varepsilon}) = \{\varepsilon\}$,
3. $\boldsymbol{\emptyset}$ je RE s jazykem $L(\boldsymbol{\emptyset}) = \emptyset$,
4. nechť $\boldsymbol{E}_1$ a $\boldsymbol{E}_2$ jsou RE, pak $\boldsymbol{E}_1 + \boldsymbol{E}_2$ je RE s jazykem $L(\boldsymbol{E}_1 + \boldsymbol{E}_2) = L(\boldsymbol{E}_1) \cup L(\boldsymbol{E}_2)$,
5. nechť $\boldsymbol{E}_1$ a $\boldsymbol{E}_2$ jsou RE, pak $\boldsymbol{E}_1\boldsymbol{E}_2$ je RE s jazykem $L(\boldsymbol{E}_1\boldsymbol{E}_2) = L(\boldsymbol{E}_1) \cdot L(\boldsymbol{E}_2)$,
6. nechť $\boldsymbol{E}$ je RE, pak $\boldsymbol{E}^*$ je RE s jazykem $L(\boldsymbol{E}^*) = (L(\boldsymbol{E}))^*$.

- priorita operátorů: nejvyšší má operace *, pak zřetězení, a nejnižší má operace $+$
- kvůlu přehlednosti si zavedeme zkrácený zápis pro tyto regulární výrazy:
  - $\boldsymbol{\Sigma}$ označuje výraz s jazykem $L(\boldsymbol{\Sigma}) = \Sigma$
  - $\boldsymbol{E}^+ = \boldsymbol{E}\boldsymbol{E}^*$

Příklad: regulární výraz

Příklad

RE E_1 pro jazyk $L_{mod} = \{w \in \{a,b\}^* \mid \text{pro } w \text{ platí, že } |w| \equiv 0 \,(\text{mod } 4)\}$, jehož slova mají délku násobků čísla 4:

$$((\boldsymbol{a}+\boldsymbol{b})(\boldsymbol{a}+\boldsymbol{b})(\boldsymbol{a}+\boldsymbol{b})(\boldsymbol{a}+\boldsymbol{b}))^*$$

nebo také pomocí zjednodušeného zápisu $(\boldsymbol{\Sigma\Sigma\Sigma\Sigma})^*$, kde $\Sigma = \{a,b\}$. Pro $L(E_1)$ platí:

- $\varepsilon \in L(E_1)$, protože $|\varepsilon| = 0 \equiv 0 \,(\text{mod } 4)$,
- $abba \in L(E_1)$, protože $|abba| = 4 \equiv 0 \,(\text{mod } 4)$,
- $aaaabbbb \in L(E_1)$, protože $|aaaabbbb| = 8 \equiv 0 \,(\text{mod } 4)$,
- $abbab \notin L(E_1)$, protože $|abbab| = 5 \not\equiv 0 \,(\text{mod } 4)$,

Nutno dodat, že mocnina není součástí definice RE, proto $(\boldsymbol{\Sigma}^4)^*$ z našeho pohledu **není** validní regulární výraz.

Regulární jazyk

Definice

Třídu jazyků, které lze popsat pomocí regulárních výrazů, označujeme **regulární jazyky** (zkráceně RL z *regular language*). To znamená, že:

- každý regulární výraz popisuje nějaký regulární jazyk,
- každý regulární jazyk lze popsat pomocí nějakého regulárního výrazu.

- říkáme, že RE E **popisuje** jazyk $L(E)$ nebo také, že E je vzor pro jazyk $L(E)$
- RE lze tedy chápat jako formalismus pro definování regulárních jazyků
  - matematický popis RE nic neříká o konkrétní implementaci, jak ověřit náležení do $L(E)$
  - jsou implementovány ve všech současných programovacích jazycích
  - později si formálně dokážeme, že verifikaci regulárních jazyků lze provádět na počítači
- z uvedených příkladů:
  - každý konečný jazyk je regulární
  - jazyk emailových adres je regulární (na dalším slidu sestrojíme RE, který jej popisuje)
  - jazyk druhých mocnim nebo $\{a^p \mid p \text{ je prvočíslo}\}$ nejsou regulární (ukážeme později)

Příklady: regulární jazyky a jejich RE

Příklad

RE E_1 pro libovolný konečný jazyk $L = \{w_1, w_2, \ldots, w_n\}$ nad nějakou abecedou Σ:

$$\boldsymbol{w_1 + w_2 + \ldots + w_n}$$

Příklad

RE E_2 pro jazyk $L = \{w \in \{a, b\} \mid w \neq bbb\}$:

$$\boldsymbol{\Sigma^* a \Sigma^* + b + bb + bbbb^+}$$

Příklad

RE E_3 pro (zjednodušený) jazyk emailových adres nad abecedou $\Sigma = \{., @, a, \ldots, z\}$:

$$(\boldsymbol{a + \ldots + z})(\boldsymbol{. + a + \ldots + z})^*\boldsymbol{@}(\boldsymbol{a + \ldots + z})^+\boldsymbol{.}(\boldsymbol{a + \ldots + z})^+$$

Regulární výrazy v praxi

- každý programovací jazyk si RE implementuje po svém (existují různé standardy)
- používají se tzv. *rozšířené regulární výrazy* (nebo také regexy)
  - poskytují navíc například syntaxi pro doplněk, mocniny, speciální znaky, atd...
  - více naleznete v přiložené ukázce (demonstrované jazykem Python)
  - doporučuji regexy používat na jakékoliv zpracování textu (například výstupy skriptů, ověřování vstupů uživatele atd.) namísto manuálního parsovaní textu

Základní nástroje pro práci s regexy v Pythonu (v balíčku `re`):

- `re.search` – najde první výskyt řetězce v textu, který odpovídá vzoru
- `re.findall` – vrátí seznam všech řetězců v textu, který odpovídají vzoru
- `re.sub` – nahradí každý řetězec v textu, který odpovídá vzoru za jiný řetězec
- `re.split` – rozdělí text na podřetězce, podle každé shody se vzorem
- `capture groups` – každá část regexu, která je uzavřená v kulatých závorkách, se uloží rozpoznaný řetězec do proměnné (v případě nalezení shody se vzorem)