456-330/1: Teoretická informatika (TI)

doc. RNDr Petr Jančar, CSc.

Katedra informatiky FEI VŠB-TU
www.cs.vsb.cz/jancar

LS 2006/2007

Základní přehled o kursu

http://www.cs.vsb.cz/jancar/TEORET-INF/teoret-inf.htm

Návaznost na kurs Úvod do teoretické informatiky
(P. Hliněný: Úvod do Teoretické Informatiky, FEI VŠB - TUO, 2004-5)

Zde (pro TI) základním textem je
Jančar, Kot, Sawa: Teoretická informatika (VŠB-TU, Ostrava, 2007),
(pdf-soubor na webu; v něm další literatura, zahrnuje i materiál k ÚTI [se svolením P. Hliněného])
Jedná se o verzi z 20.2.2007. Není vhodné (natož nutné) ihned tisknout!
Text bude dále upravován a revidován.

Na webu také např. cvic-tjaa-2p.pdf, cvic-vycsloz-2p.pdf.

a hlavně informace o průběhu kursu po jednotlivých týdnech.

Užitečné jsou také současné slidy u předmětu ÚTI (Kot, Sawa), stažitelné na http://www.cs.vsb.cz/kot/.

Výuka, ESF projekt

V Katisu u TI je 4/4/0/0/0; de facto se má jednat o 4/2/0/0/2. (přednáška zahrnuje i prvky cvičení, samostatná práce studentů znamená referát, přípravu na cvičení, samostatné řešení dalších příkladů)

Vývoj učebních materiálů je také podporován ESF-projektem. V té souvislosti podpisy prezenčních listin (+ bydliště); bude realizováno na cvičeních.

Následující přehled témat kursu je jen orientační (čísla kapitol se odkazují k dřívějšímu materiálu a nejsou již relevantní).

Konečné automaty a regulární jazyky

Kapitolka 1 (úvod k části teorie jazyků a automatů).
Kapitola 2 (Konečné automaty, regulární jazyky)
Kapitola 3 (Návrh konečných automatů, regulární operace).
Kapitola 4 (Nedeterministické konečné automaty a jejich vztah k deterministickým.)
Kapitola 5 (Uzávěrové vlastnosti třídy regulárních jazyků.)
Kapitola 6 (Regulární výrazy a jejich ekvivalence s konečnými automaty.)
Kapitola 7 (Minimální konečné automaty a algoritmus minimalizace.)
Kapitola 8 (Neregulární jazyky. Pumping lemma pro regulární jazyky.)
Kapitola 9 (Další poznámky ke konečným automatům.)

Bezkontextové jazyky

Kapitola 10 (Bezkontextové gramatiky a jazyky.)
Kapitola 11 (Úloha syntaktické analýzy v překladačích)
Kapitola 12 (Speciální formy bezkontextových gramatik.)
Kapitola 13 (Zásobníkové automaty; ekvivalence s bezkontextovými gramatikami.)
Kapitola 14 (Pumping lemma pro bezkontextové jazyky.)
Kapitola 15 (Uzávěrové vlastnosti třídy bezkontextových jazyků.)
Kapitola 16 (Deterministické zásobníkové automaty.)
kapitoly 17 - 20 (Chomského hierarchie, Konečné automaty a regulární gramatiky, Turingovy stroje, Další poznámky ke vztahu automatů a gramatik)

Teorie algoritmů

Kapitola 21 (Složitost algoritmu. Model RAM.)
Kapitola 22 (Odhady složitosti; značení O(n) aj.)
Kapitola 23 (Návrh a analýza konkrétních rychlých algoritmů.)
Kapitola 24 (Problémy, třídy složitosti problémů, horní a dolní odhady.)
Kapitola 25 (Třída PTIME a její robustnost. Turingovy stroje.)
Kapitola 26 (začátek: Třída NPTIME.)
Kapitola 26 (pokračování: Polynomiální převeditelnost. NP-úplné problémy.)
Kapitola 27 (Další třídy časové i prostorové složitosti)
Kapitola 28 (Rozhodnutelnost a nerozhodnutelnost problémů.)
Kapitola 29 (Nerozhodnutelné problémy. Riceova věta.)
Kapitola 30, 1. - 2. (Aproximační algoritmy, pravděpodobnostní algoritmy.)
Kapitola 30, 3 - 5 (Paralelní algoritmy, distribuované algoritmy, nové výpočetní modely.)

Požadavky k zápočtu

- Na cvičeních ve 4., 7., 10. a 13. týdnu výuky (tedy 4 krát; viz "Průběh výuky") se vždy bude psát 25-minutová písemka; z každé z nich je možné získat až 6 bodů (celkově tedy 24 bodů). Nutnou podmínkou k získání zápočtu je zisk alespoň 4 bodů alespoň dvakrát. (Náhradní termíny pro nezúčastněné nebudou.)
- Další nezbytnou podmínkou k získání zápočtu je úspěšně zvládnutí referátu. 11 bodů obdrží za kvalitně vypracovaný referát, který srozumitelně odprezentuje (čímž mj. prokáže, že tématu skutečně rozumí); v opačném případě úkol nesplní a zápočet nedostane.
- Při nesplnění některé nutné podmínky student zápočet nedostane. (Za udělený zápočet tedy student získá minimálně $8 + 11 = 19$ bodů a maximálně $24 + 11 = 35$ bodů)

Zkouška

písemná (90-minutová), podle potřeby doplněná ústní částí:

(max. zisk 65 bodů; minimální zisk k uznanému absolvování: 25 bodů)

První 3 - 4 týdny: konečné automaty, regulární výrazy, regulární jazyky (z toho také první zápočtová písemka ve 4. týdnu).

Formou "dílny": při řešení konkrétních problémů "sami odvodíme" příslušné pojmy a metody (a prokážeme jejich správnost).

Mj. použijeme prostředek fiktivní komunikace mezi
"běžným programátorem" B (ten je tedy B-čko)
a absolventem informatiky na FEI VŠB-TUO, označeným A (tedy Á-čko)

Jednoho dne si B postěžoval na svou práci na pracovišti, kde musí psát programy pro vyhledávání ve velkých souborech dat (řetězcích znaků, a to nejen normálních textů ....) ...

Ilustroval to následujícím příkladem svého programu:

Vyhledávání řetězce *abaaba*; návrh B-čka

```
procedure SEARCH (var F: file)
const length = 6                (* delka hledaneho retezce *)
const P = [ a,b,a,a,b,a ]       (* hledany retezec *)
var A: array [ 1..length ] of char
begin
  for i:=1 to length do
    read( A[i], F ); if EOF (* end of file *) then return
  endfor
  while true do
    if EQUAL(P,A) then ‘‘vypis misto vyskytu’’
    for i:=1 to length-1 do
      A[i]:=A[i+1]
    endfor
    read( A[length], F ); if EOF then return
  endwhile
end
```

Procedura EQUAL v návrhu B-čka

```
function procedure EQUAL
(var S1,S2: array [ 1..length ] of char): boolean
begin
  for i:=1 to length do
    if not( S1[i] = S2[i] ) then return FALSE
  endfor
  return TRUE
end
```

B si stěžoval, že u hodně velkých souborů trvá prohledávání “dost dlouho”. Á-čko mu navrhl následující program, který pak potřebný čas (několika)násobně zkrátil.

Vyhledávání řetězce *abaaba*; návrh Á-čka

```
procedure SEARCH (var F: file)
const length = 6
type state = [0 .. length]
const A: array [ state , [‘a’,‘b’] ] of state
   = [ [1,0], [1,2], [3,0], [4,2], [1,5], [6,0], [4,2] ]
var q: state
begin
  q:=0
  while true do
    if q=6 then ‘‘vypis misto vyskytu’’
    read( ch, F ); if EOF then return
    q := A[ q, ch ]
  endwhile
end
```

B vyzvídal, jak na to A přišel, a jestli je jeho program zaručeně správně. (B-čko ze skušenosti věděl, že pár úspěšných běhů celkovou správnost nedokazuje.)

Á-čko mu vysvětlil, že na FEI jej učili, že
- zadaný úkol je nejdříve potřeba přesně specifikovat, promýšlet z různých úhlů atd., až
- důkladné promyšlení a porozumění úkolu "samo" navede k 'tomu pravému' řešení a zároveň k důkazu jeho správnosti

Máme řešit úkol U_0.
U_0 (specifikace: v daném znakovém souboru (velkém řetězci znaků a, b) "najdi a ohlaš" každý výskyt '*abaaba*')
Návrh realizace: Soubor bude čten sekvenčně, přičemž U_0 je realizováno takto:
U_0:
- přečti další znak;

- když je to a, tak
  U_1 (specifikace: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*' ale na začátku také případný výskyt prefixu '*baaba*')
- když je to b, tak
  U_2 (specifikace: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*'

Všimněme si, že U_2 je vlastně totéž, co U_0 (pokud za daný soubor teď považujeme nepřečtený zbytek souboru), takže máme:

U_0 (specifikace: přečti zbytek souboru a "ohlaš" každý výskyt '*abaaba*')
realizujeme takto:
- přečti další znak;

- když je to a, tak
  U_1 (specifikace: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*' ale na začátku také případný výskyt prefixu '*baaba*')
- když je to b, tak (udělej) U_0

realizace U_1:
- přečti další znak;

- když je to a, tak U_1
- když je to b, tak
  U_2 (specifikace: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*' ale na začátku také případný výskyt prefixu '*aaba*')

Zbývá tedy realizovat
U_2 (specifikace: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*' ale na začátku také případný výskyt prefixu '*aaba*')
B říká, že už rozumí a navrhne realizaci U_2 takto:
- přečti další znak;

- když je to *a*, tak
  U_3: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*' ale na začátku také případný výskyt prefixu '*aba*'
- když je to *b*, tak U_0

Je to tak v pořádku ?
Ne! *A* upozorní *B*, že nebyl dostatečně pečlivý a napíše:
U_3: přečti zbytek souboru, přičemž ohlaš každý výskyt '*abaaba*', ale na začátku také případný výskyt prefixu '*aba*' a případný výskyt prefixu '*baaba*'

Á-čko upozorní také na nutnost větší přehlednosti a navrhne:
Úkoly U_i mají stejné schéma, zapišme ke každému to podstatné stručněji (vedle přitom kreslí jakýsi graf; jaký asi?):

U_0 (všechny '*abaaba*'):
když *a*, tak U_1, když *b*, tak U_0

U_1 (prefix '*baaba*', všechny '*abaaba*'):
když *a*, tak U_1, když *b*, tak U_2

U_2 (prefix '*aaba*', všechny '*abaaba*'):
když *a*, tak U_3, když *b*, tak U_0

U_3 (prefixy '*aba*', '*baaba*' a všechny '*abaaba*'):
když *a*, tak U_4, když *b*, tak U_2

U_4 (prefixy '*ba*', '*baaba*' a všechny '*abaaba*'):
když *a*, tak U_1, když *b*, tak U_5

U_5 (prefixy '*a*', '*aaba*' a všechny '*abaaba*'):
když *a*, tak U_6, když *b*, tak U_0

U_6 (prefixy '', '*aba*', '*baaba*' a všechny '*abaaba*'):
OHLAŠ; pak když *a*, tak U_4, když *b*, tak U_2

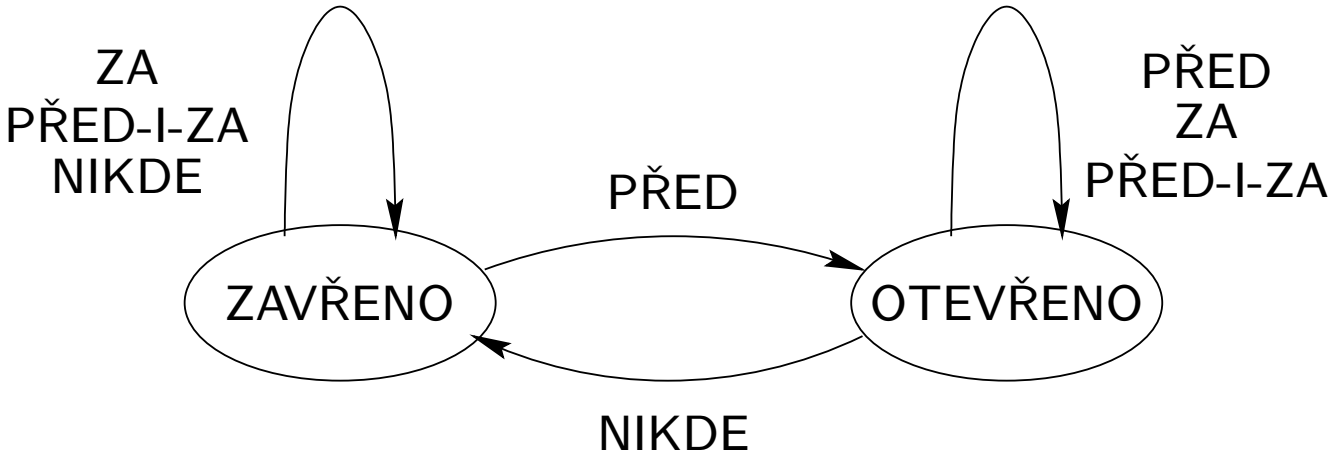
Přirozené zadání grafem ... (zvýší názornost pro člověka v případě dobrého nakreslení)
Do programu přirozeně zadáme tabulkou (dvourozměrným polem) ...
B-čko si všimne, že Á-čko má v přiloženém grafu v jednotlivých vrcholech napsáno
$(0 = \varepsilon), (1 = a), (2 = ab), (3 = aba), (4 = abaa), (5 = abaab), (6 = abaaba)$

Á-čko říká, že tohle značení slouží pro umožnění snadného ověření správnosti programu.

B-čku sice ještě plně 'nedocvakla' souvislost prefixů hledaného vzorku s prefixy uvedenými u $U_0, U_1, \ldots, U_6$, ale je spokojen, že alespoň pochopil způsob práce Á-čka zlepšující jeho dříve používanou metodu.

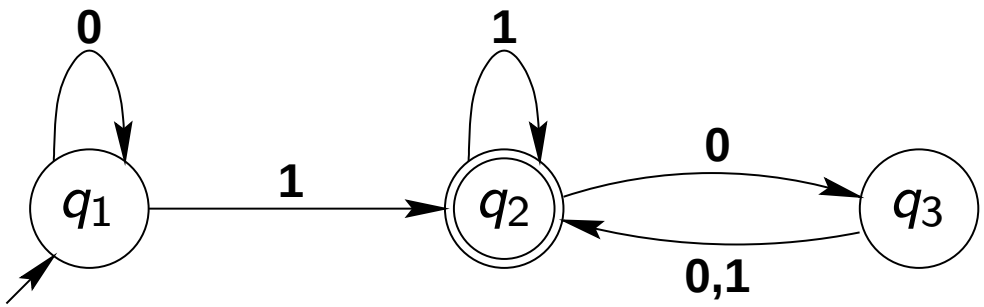
B teď sleduje krátký výklad Á-čka o tzv. konečných automatech a jazycích jimi přijímaných.
Pravda je, že B je trochu zaražen, když Á-čko najednou používá řecká písmena, píše jakési ‘hieroglyfy’ typu
$A = (Q, \Sigma, \delta, q_0, F)$,
$\delta : Q \times \Sigma \to Q$,
zápis ‘$L \subseteq \Sigma^*$’ kupodivu přečte jako ‘jazyk nad abecedou sigma’,
zápis ‘$L_2 \backslash L_1$’ čte jako ‘levý kvocient jazyka el-jedna podle jazyka el-dva’ apod.
B-čko moc nechápe, k čemu jsou vlastně takové kejkle dobré, má trochu podezření, že Á-čko chce jen ‘frajeřit’ s matematickou notací,
ale alespoň je mu nakonec jasné, že to vše nějak ‘matematicky’ zachycuje to, co předtím už intuitivně pochopil.

Konečný automat



	PŘED	ZA	PŘ-I-ZA	NIKDE
ZAV	OTEV	ZAV	ZAV	ZAV
OTEV	OTEV	OTEV	OTEV	ZAV

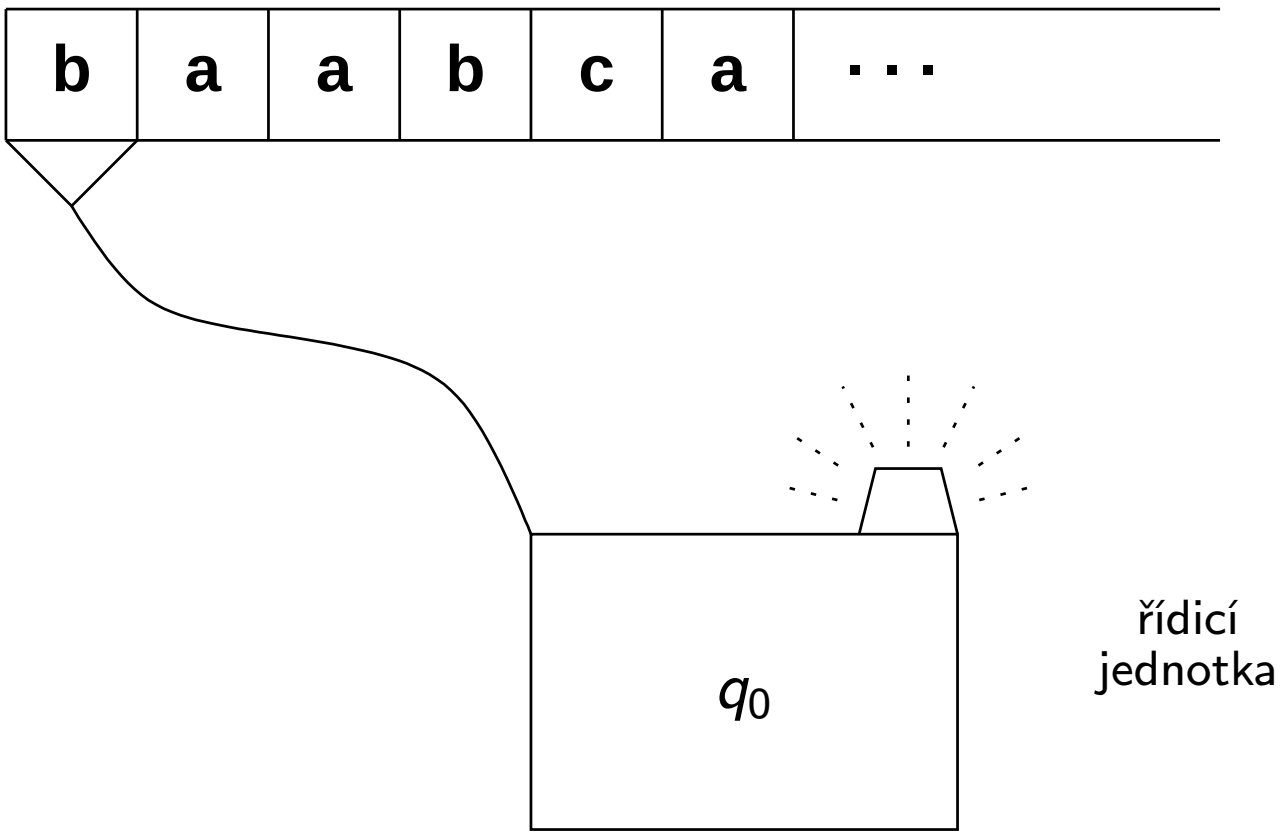
Konečný automat - rozpoznávač jazyka



přijímá:
1101, 010101, . . .

nepřijímá:
0110, 0010, . . .

Konečný automat - "vnější" pohled



Konečný automat - definice

Konečný automat (KA)
$A = (Q, \Sigma, \delta, q_0, F)$

Q ... konečná množina *stavů*

Σ ... konečná množina (vstupních) *symbolů* (*abeceda*)

$\delta : Q \times \Sigma \to Q$ ... *přechodová funkce*

$q_0 \in Q$ ... *počáteční stav*

$F \subseteq Q$ ... množina *přijímajících (koncových) stavů*

Abeceda, slovo, jazyk

abeceda .... označujeme Σ, Γ, Δ, ...

prvky abecedy (*písmena*) ... označujeme $a, b, c, \ldots$

slovo .... $u, v, w, \ldots$, $u = a_1 a_2 \ldots a_n$

délka slova ... $|u|$
prázdné slovo ... ε ($|\varepsilon| = 0$)

zřetězení slov ... $u \cdot v$, stručněji uv

u je *podslovem* slova $v \quad \Leftrightarrow_{df}$ ex. slova w_1, w_2 tž. $v = w_1 u w_2$.
u je *prefixem* (*sufixem*) slova $v \quad \Leftrightarrow_{df} \quad \ldots$

značení $u^0 = \varepsilon$, $u^1 = u$, $u^2 = uu$, ... $u^n = uu..u$ (n-krát)

Σ^* ... množina všech slov v abecedě Σ

Jazyk nad abecedou Σ ... $L \subseteq \Sigma^*$

Regulární jazyky

$A = (Q, \Sigma, \delta, q_0, F)$
$\delta^* : Q \times \Sigma^* \to Q$

1. $\delta^*(q, \varepsilon) = q$,
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

Slovo $w \in \Sigma^*$ je *přijímáno* automatem $A \quad \Leftrightarrow_{df} \quad \delta^*(q_0, w) \in F$

Jazykem rozpoznávaným (přijímaným) automatem A rozumíme jazyk
$L(A) = \{w \mid$ slovo w je přijímáno $A\}$

Jazyk L je *regulární* (tj. rozpoznatelný konečným automatem) $\Leftrightarrow_{df}$
existuje KA A tž. $L(A) = L$

B-čko se neprozřetelně zeptal, jestli jsou taky nějaké neregulární jazyky. Á-čko mu nato začal vykládat něco o spočetných a nespočetných množinách, množinách všech podmnožin atd., až z toho B-čku šla hlava kolem.

Nakonec Á-čko B-čku slíbil, že se k té otázce někdy vrátí, až bude více času. (B-čko zase musel do práce, nemá holt moc času na studium nějakých 'ezoterických' matematických pojmů.)

Později se Á-čko zase setká s B-čkem.
B je celý rozzářený, říká, že dostal úkol, s kterým by dříve zřejmě nehnul, ale zachránila ho metoda, kterou ho A naučil.
A aby ukázal, že taky umí ‘matematicky frajeřit’, tak ten úkol zformuloval takto:

měl jsem navrhnout konečný automat pro rozpoznávání jazyka

$$L_0 = \{\, u \in \{0,1\}^* \mid bn(u) \mod 3 = 0 \text{ a } u \text{ neobsahuje podslovo } 101 \,\}$$

kde $bn(u)$ je číslo s binárním zápisem u. ($bn(\varepsilon)$ je chápáno jako 0.)

Á-čka samozřejmě zajímá, jak B postupoval. Ten mu postup s hrdostí ukazuje. Říká, že to byla fuška, ale za dvě šichty to měl.

$L_0 = \{ u \in \{0,1\}^* \mid bn(u) \mod 3 = 0 \text{ a } u \text{ neobsahuje podslovo } 101 \}$

$$\begin{aligned}
\{0\}\backslash L_0 &= \{ u \mid 0u \in L_0 \} \\
&= \{ u \mid bn(0u) \mod 3 = 0 \text{ a } 0u \text{ neobsahuje podslovo } 101 \} \\
&= \{ u \mid bn(u) \mod 3 = 0 \text{ a } u \text{ neobsahuje podslovo } 101 \} = L_0
\end{aligned}$$

$$\begin{aligned}
\{1\}\backslash L_0 &= \{ u \mid 1u \in L_0 \} \\
&= \{ u \mid bn(1u) \mod 3 = 0 \text{ a } 1u \text{ neobsahuje podslovo } 101 \} \\
&= \{ u \mid (2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo } 101 \text{ a } u \text{ nezačíná } 01 \} = L_1
\end{aligned}$$

$$\begin{aligned}
\{0\}\backslash L_1 &= \{ u \mid 0u \in L_1 \} \\
&= \{ u \mid (2^{|0u|} + bn(0u)) \mod 3 = 0 \\
&\qquad \text{a } 0u \text{ neobsahuje podslovo } 101 \text{ a } 0u \text{ nezačíná } 01 \} \\
&= \{ u \mid (2^{|u|+1} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo } 101 \text{ a } u \text{ nezačíná } 1 \} \\
&= \{ u \mid (2 \cdot 2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo } 101 \text{ a } u \text{ nezačíná } 1 \} = L_2
\end{aligned}$$

$$L_1 = \{\, u \mid (2^{|u|} + bn(u)) \mod 3 = 0$$
$$\text{a } u \text{ neobsahuje podslovo 101 a } u \text{ nezačíná 01} \,\}$$

$$\begin{aligned}
\{1\}\backslash L_1 &= \{\, u \mid 1u \in L_1 \,\} \\
&= \{\, u \mid (2^{|1u|} + bn(1u)) \mod 3 = 0 \\
&\qquad \text{a } 1u \text{ neobsahuje podslovo 101 a } 1u \text{ nezačíná 01} \,\} \\
&= \{\, u \mid (2 \cdot 2^{|u|} + 2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo 101 a } u \text{ nezačíná 01} \,\} \\
&= \{\, u \mid bn(u) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo 101 a } u \text{ nezačíná 01} \,\} = L_3
\end{aligned}$$

$$L_2 = \{\, u \mid (2 \cdot 2^{|u|} + bn(u)) \mod 3 = 0 \text{ a } u \text{ neobsahuje podslovo 101 a } u \text{ nezačíná 1} \,\}$$

$$\begin{aligned}
\{0\}\backslash L_2 &= \{\, u \mid 0u \in L_2 \,\} \\
&= \{\, u \mid (2 \cdot 2^{|0u|} + bn(0u)) \mod 3 = 0 \\
&\qquad \text{a } 0u \text{ neobsahuje podslovo 101 a } 0u \text{ nezačíná 1} \,\} \\
&= \{\, u \mid (2 \cdot 2 \cdot 2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo 101} \,\} \\
&= \{\, u \mid (3 \cdot 2^{|u|} + 1 \cdot 2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo 101} \,\} \\
&= \{\, u \mid (2^{|u|} + bn(u)) \mod 3 = 0 \\
&\qquad \text{a } u \text{ neobsahuje podslovo 101} \,\} = L_4
\end{aligned}$$

B-čko byl pečlivý a nezaváděl zbytečné nové stavy, tedy všechny $L_0, L_1, L_2, \ldots$, které takto zavedl (a popsal) byly navzájem různé. Automat s kolika stavy tak zkonstruoval?
Á-čko pochválil jeho pečlivost, ale navrhne mu pro příště údajně lepší metodu. Napíše

Ke konečným automatům $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$,
$A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
sestroj
$A = (Q, \Sigma, \delta, q_0, F)$ takto:

- $Q = Q_1 \times Q_2$
- pro každé $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$:
  $\delta(\,(q_1, q_2), a\,) = (\,\delta_1(q_1, a), \delta_2(q_2, a)\,)$
- $q_0 = (q_{01}, q_{02})$
- $F = F_1 \times F_2$

B-čko nerozumí, ani když Á-čko připíše $L(A) = L(A_1) \cap L(A_2)$.

Když mu ale Á-čko ukáže následující fragment programu, tak B-čku začne pomalu svítat.

```
type state1 = 1 .. length1
type state2 = 1 .. length2
const A1: array [ state1 , [‘0’,‘1’] ] of state1 = ...
const A2: array [ state2 , [‘0’,‘1’] ] of state2 = ...
var q: state1
var r: state2
begin
  q:=1; r:=1
  while true do
    if ( (q má hlásit) a (r má hlásit) ) then OHLAŠ
    read( ch, F ); if EOF then return
    q := A1[ q, ch ] ; r := A2[ r, ch ]
  endwhile
end
```

Spolu pak navrhli 3-stavový automat A_1 tak, že

$$L(A_1) = \{ u \in \{0,1\}^* \mid bn(u) \mod 3 = 0 \}$$

(Jaký?)

Potom navrhli 4-stavový automat A_2 tak, že

$$L(A_2) = \{ u \in \{0,1\}^* \mid u \text{ neobsahuje podslovo } 101 \}$$

(Jaký?)

Pak (naprosto rutinně) vyrobili 12-stavový automat A tak, že

$L(A) = L(A_1) \cap L(A_2) =$
$= \{ u \in \{0,1\}^* \mid bn(u) \mod 3 = 0 \text{ a } u \text{ neobsahuje podslovo } 101 \}$

(Udělejme to také!)

B-čko přiznal, že uvedený způsob (modulárního návrhu) je jistě intelektuálně méně náročný a vede rychleji k cíli (Á-čko mu připomínal, že rozdělení velkého úkolu na menší a složení řešení většího z řešení menších je samozřejmě obecně prospěšný postup),
chlubil se ale, že jeho úsilí má výhodu alespoň v tom, že vedlo k menšímu automatu pro daný jazyk. (Jeho automat měl totiž méně stavů než oněch 12.)

Á-čko to sice ocenil, ale řekl, že i při takové chvályhodné snaze o minimalizaci konstruovaného automatu (což může být v jistých praktických kontextech skutečně velmi důležité) existují vcelku přímočaré rutinní (tj. algoritmické) postupy, které k cíli vedou zaručeně a bez mentální námahy.

Pak předvedl B-čku takový postup na zkonstruovaném automatu A s 12 stavy.