DSA

Různé algoritmy mají různou složitost

DSA

The complexity of different algorithms varies

Jazyk

Abeceda

Abeceda … konečná (neprázdná) množina symbolů
|A| … **mohutnost** abecedy A

Příklady: A = { ‘A’, ‘D’, ‘G’, ‘O’, ‘U’}, |A| = 5
A = {0,1}, |A| = 2
A = {○, □, △}, |A| = 3

Slovo

Slovo (nad abecedou A) … konečná (příp. prázdná) posloupnost symbolů abecedy (A)
také **řetězec**
|w| … **délka** slova w

Příklady: w = OUAGADOUGOU, |w| = 11
w = 1001, |w| = 4
w = □△○△□, |w| = 5

Jazyk

Jazyk

Jazyk … množina slov (=řetězců)
(ne nutně konečná, může být prázdná)
|L| … mohutnost jazyka L

---

1 **Specifikace jazyka** -- **Výčtem všech slov jazyka**
(jen pro konečný jazyk)

Příklady: $A_1 = \{‘A’, ‘D’, ‘G’, ‘O’, ‘U’\}$

$L_1 = \{ADA, DOG, GOUDA, D, GAG\}, \quad |L_1| = 5$

$A_2 = \{0,1\}$

$L_2 = \{0, 1, 00, 01, 10, 11\}, \quad |L_2| = 6$

$A_3 = \{ ○, □, △ \}$

$L_3 = \{ △△, ○□○, □□△○ \}, \quad |L_2| = 3$

Jazyk

② **Specifikace jazyka -- Volný (ale jednoznačný) popis v přirozeném jazyce (obvykle pro nekonečný jazyk)**

Příklady: $A_1 = \{$'A', 'D', 'G', 'O', 'U'$\}$

L_1: Množina všech slov nad A_1, která začínají na DA, končí na G a neobsahují podposloupnost AA.

$L_1 = \{DAG, DADG, DAGG, DAOG, DAUG, DADAG, DADDG\ldots\}$

$|L_1| = \infty$

$A_2 = \{0,1\}$

L_2: Množina všech slov nad A_2, která obsahují více 1 než 0 a za každou 0 jsou alespoň dvě 1.

$L_2 = \{ 1, 11, 011, 0111, 1011, 1111, \ldots, 011011, 011111, \ldots \}$

$|L_2| = \infty$

Konečný automat

3 Specifikace jazyka -- konečným automatem

Konečný automat (finite automaton)
je pětice (A, Q, σ, S_0, Q_F), kde:

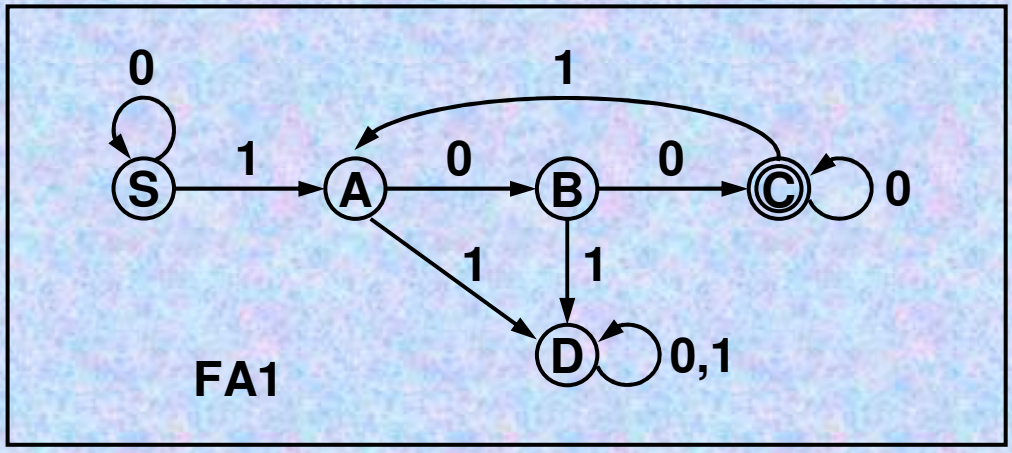
A … abeceda … konečná množina symbolů
|A| ... mohutnost abecedy
Q ... množina stavů (mnohdy očíslovaných) (co je to „stav“ ?)
σ ... přechodová funkce ... $\sigma: Q \times A \rightarrow Q$
S_0 ... počáteční stav $S_0 \in Q$
Q_F … neprázdná množina koncových stavů $\emptyset \neq Q_F \subseteq Q$

Konečný automat

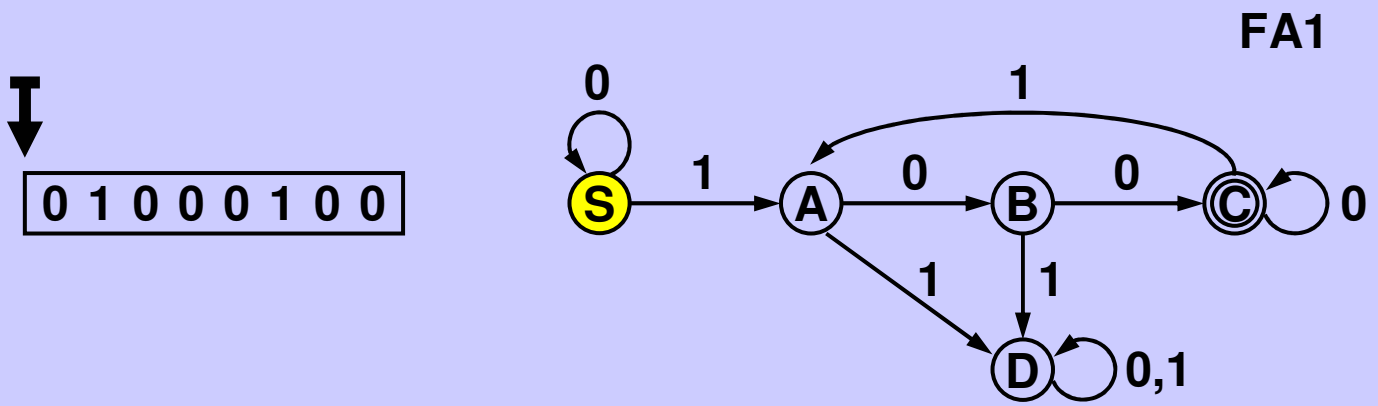
Automat FA1:

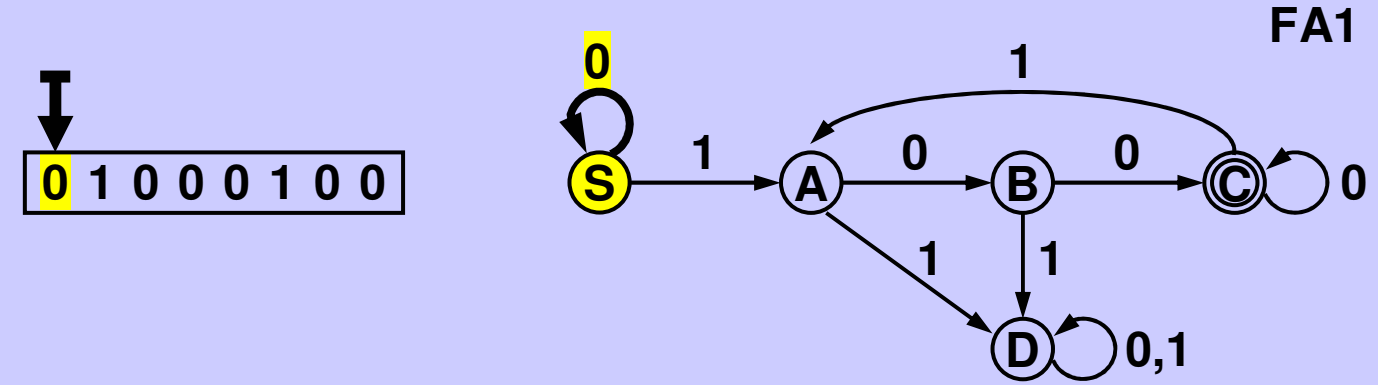
A … abeceda … {0,1}, |A| = 2
Q ... množina stavů {S, A, B, C, D }
σ ... přechodová funkce ... σ: Q×A → Q : {
σ(S,0) = S, σ(A,0) = B, σ(B,0) = C, σ(C,0) = C, σ(D,0) = D,
σ(S,1) = A, σ(A,1) = D, σ(B,1) = D, σ(C,1) = A, σ(D,1) = D }
S_0 ... počáteční stav S ∈ Q
Q_F … neprázdná množina koncových stavů ∅ ≠ { C } ⊆ Q

Přechodový diagram automatu FA1

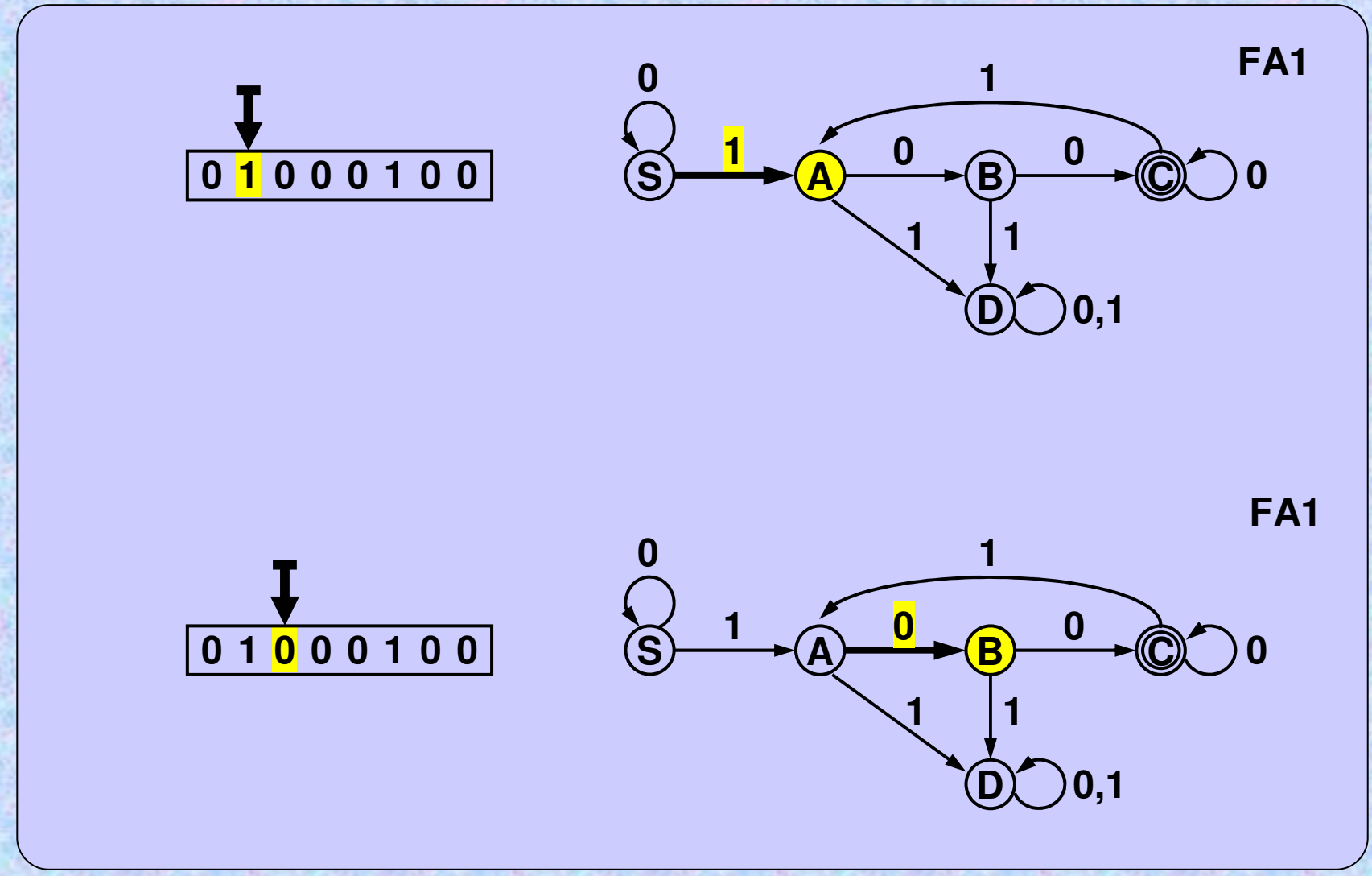


Konečný automat

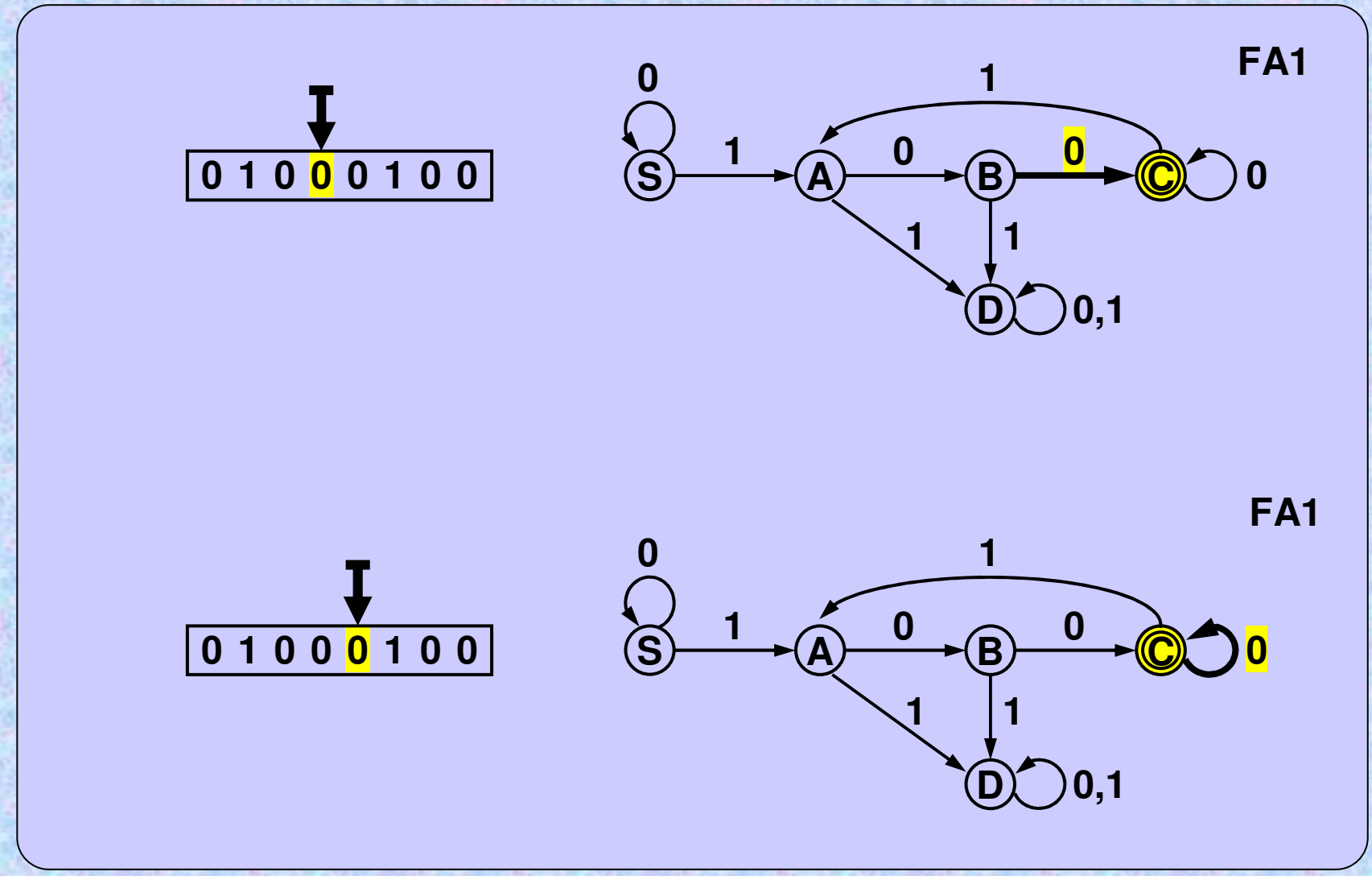




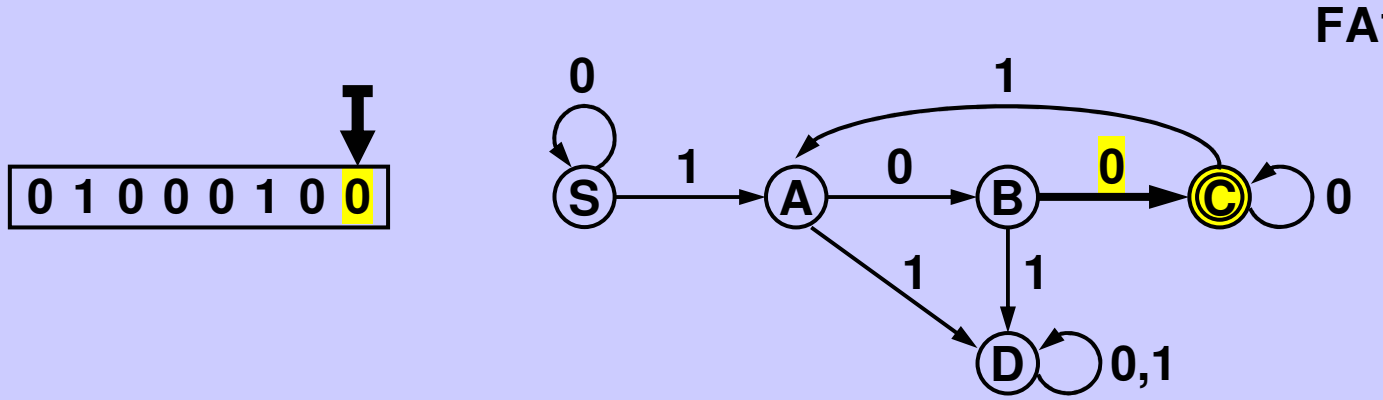
Konečný automat



Konečný automat



Konečný automat

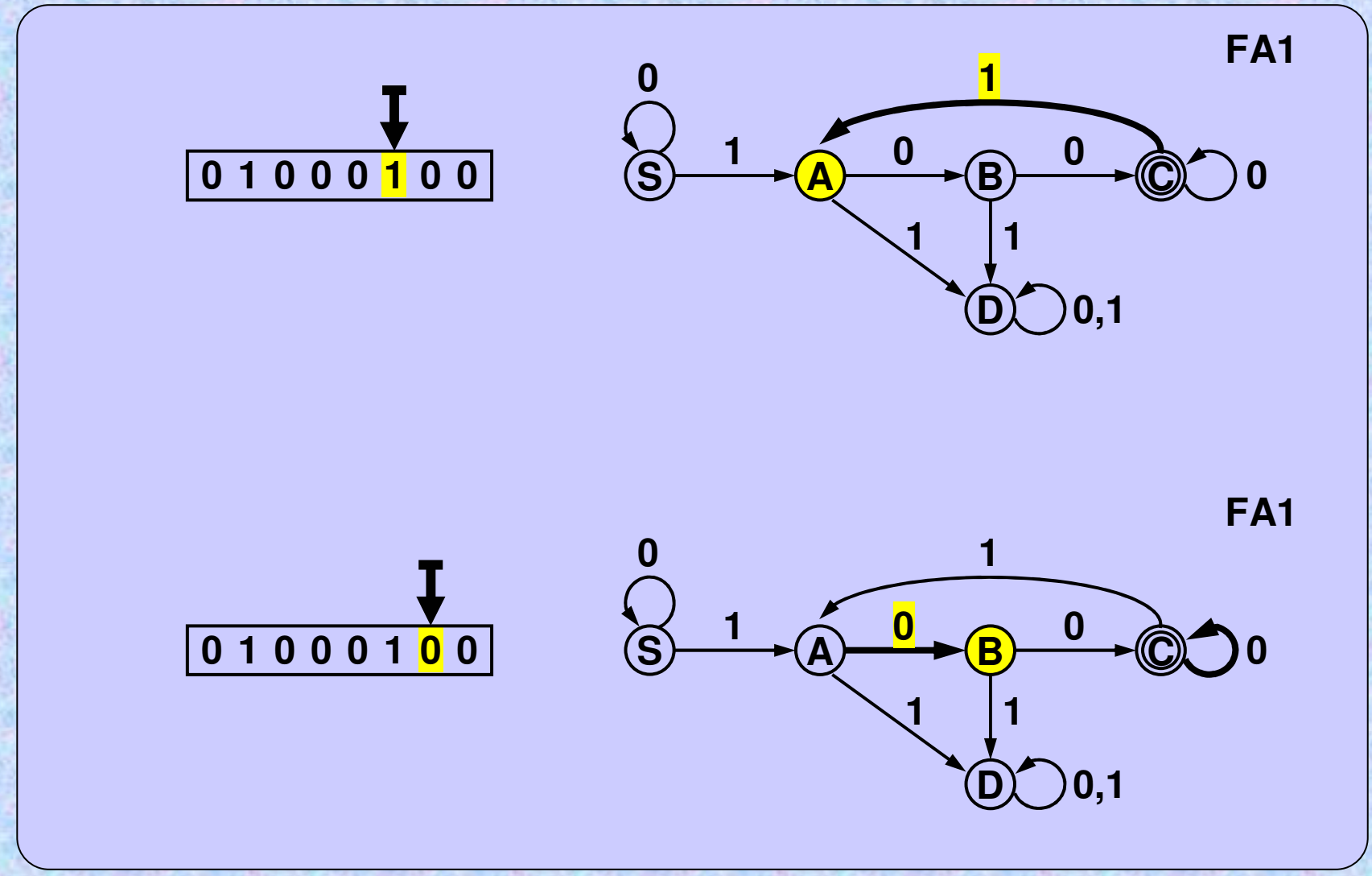


Po přečtení posledního symbolu je automat FA1 v koncovém stavu ◎

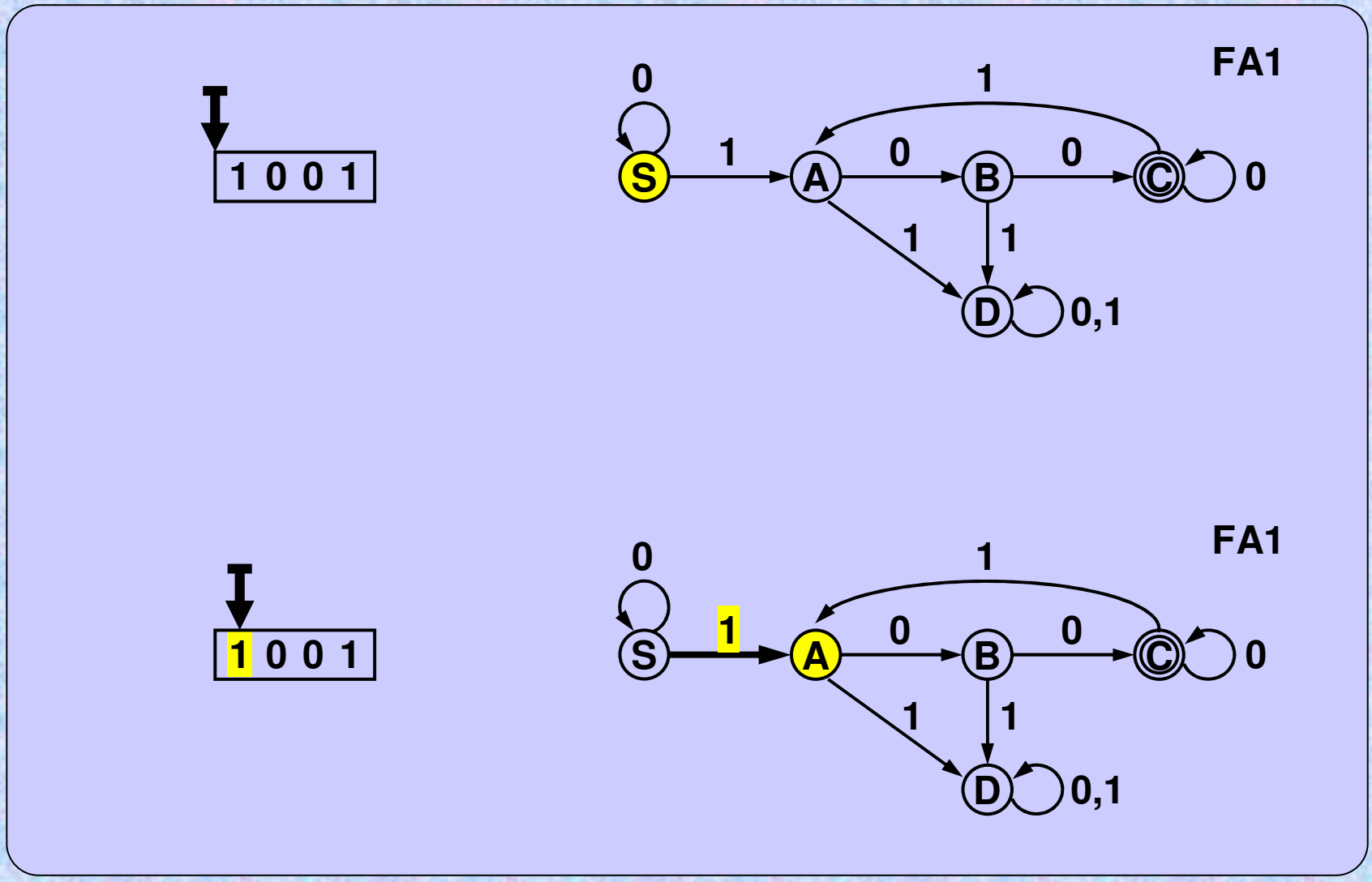
⇒

Slovo 0 1 0 0 0 1 0 0 je přijímáno automatem FA1

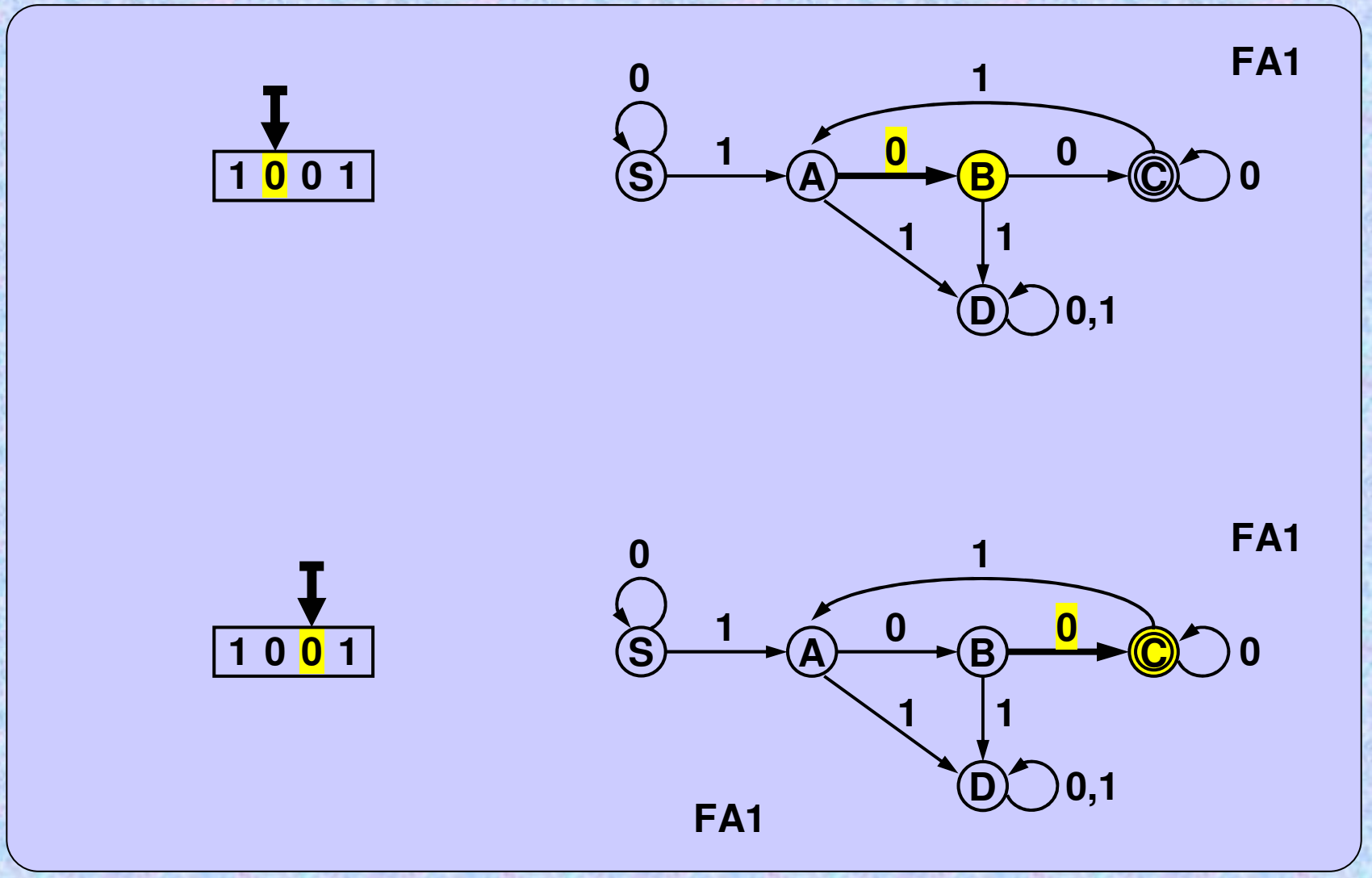
Konečný automat



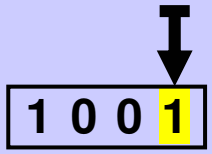
Konečný automat

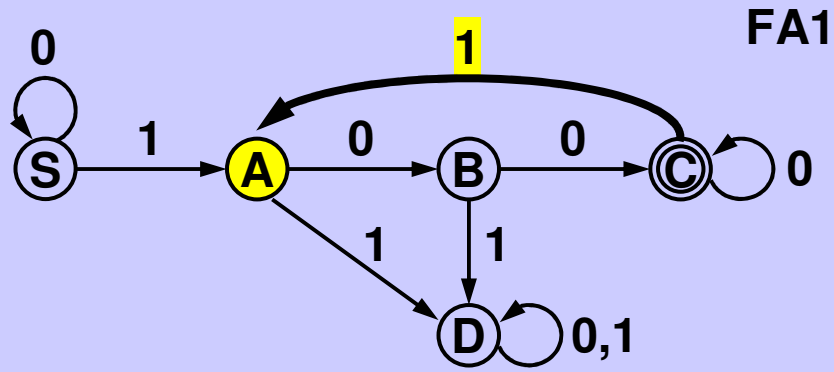


Konečný automat



Konečný automat



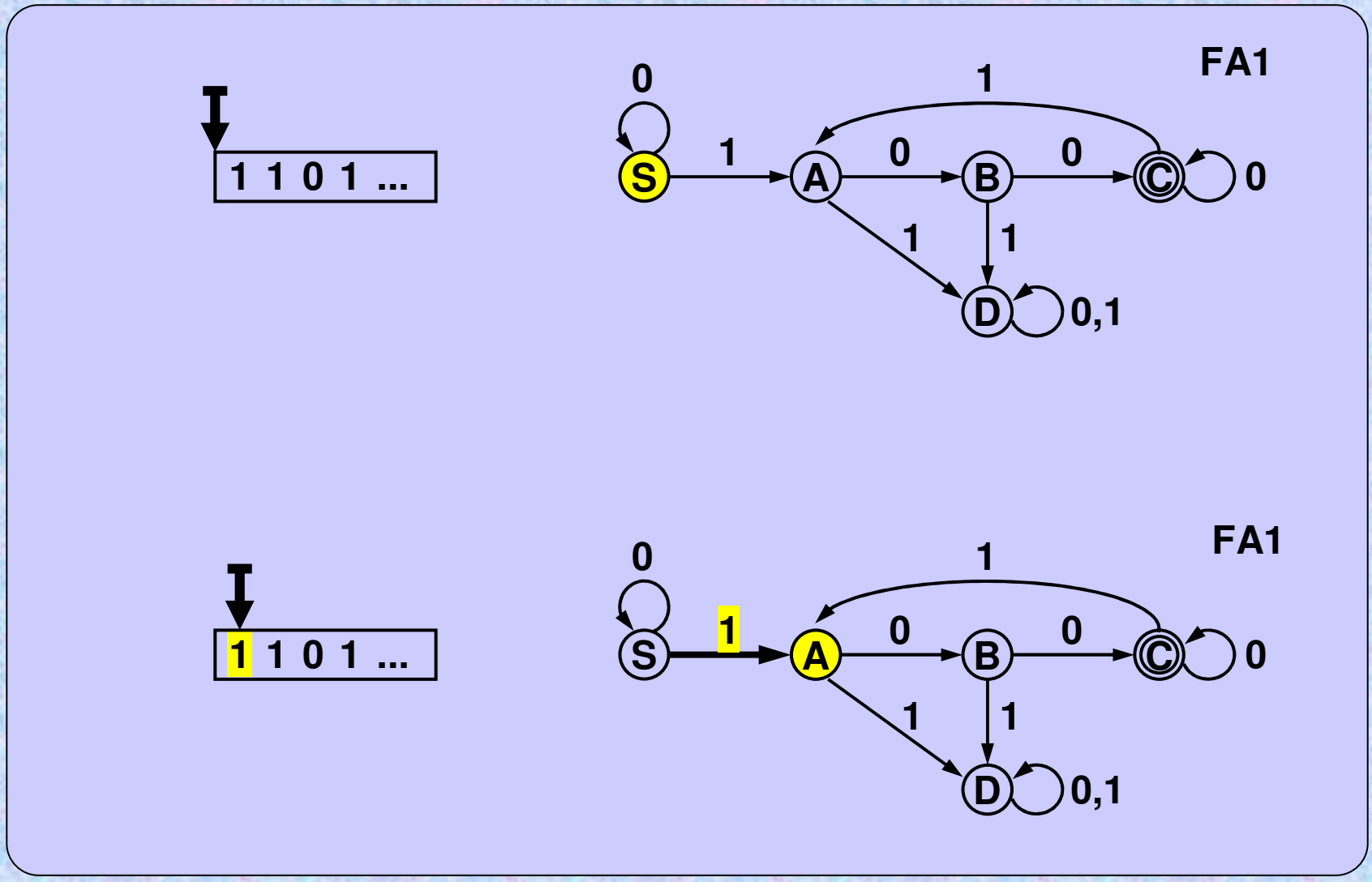


Po přečtení posledního symbolu je automat FA1 ve stavu, který není koncový ◯

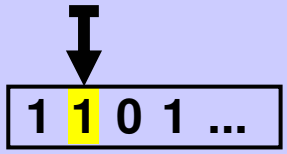
⇒

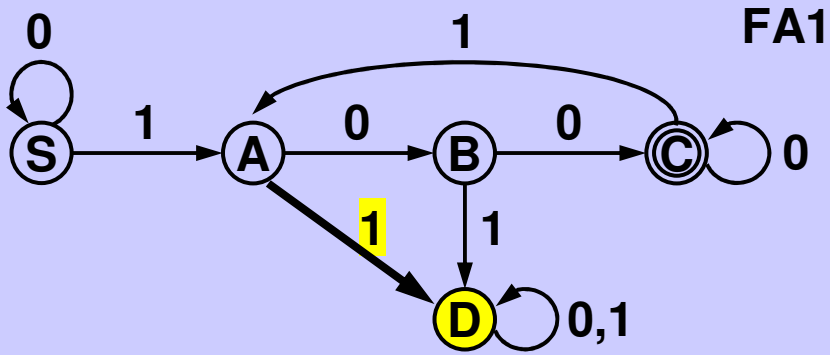
Slovo 1 0 0 1 <u>není přijímáno</u> automatem FA1

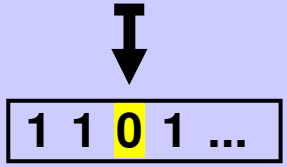
Konečný automat

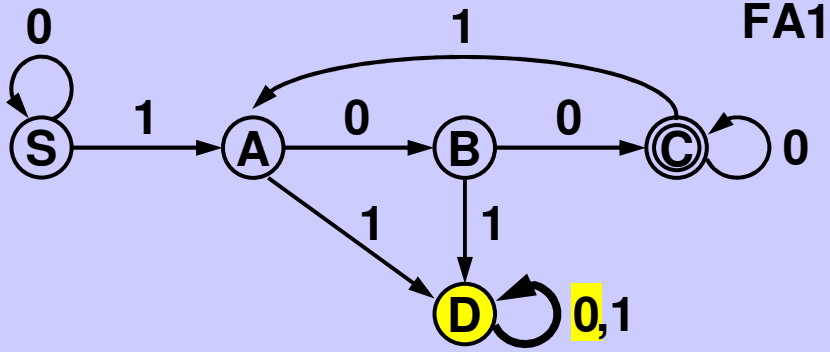


Konečný automat

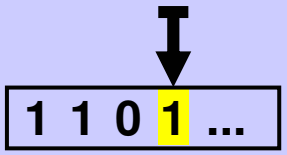


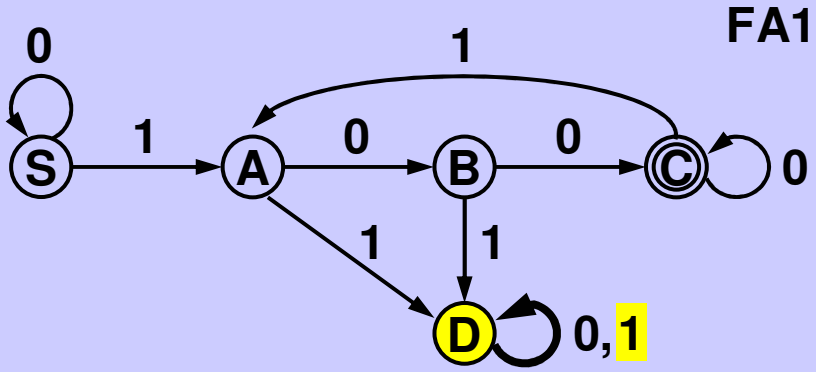






Konečný automat





Žádné slovo začínající	1 1 ...	není přijímáno automatem FA1
Žádné slovo obsahující	... 1 1 ...	není přijímáno automatem FA1
Žádné slovo obsahující	... 1 0 1 ...	není přijímáno automatem FA1

Automat FA1 přijímá pouze slova -- obsahující alespoň jednu jedničku
-- obsahující za každou jedničkou alespoň dvě nuly

Jazyk přijímaný automatem = množina všech slov přijímaných automatem

Konečný automat

Činnost automatu:

Na začátku je automat ve startovním stavu.

Pak čte zadané slovo znak po znaku a přechází

do dalších stavů podle přechodové funkce.

Když dočte slovo, nachází se opět v některém ze svých stavů.

Pokud je v koncovém stavu, řekneme, že dané slovo přijímá,

pokud není v koncovém stavu, řekneme, že dané slovo nepřijímá.

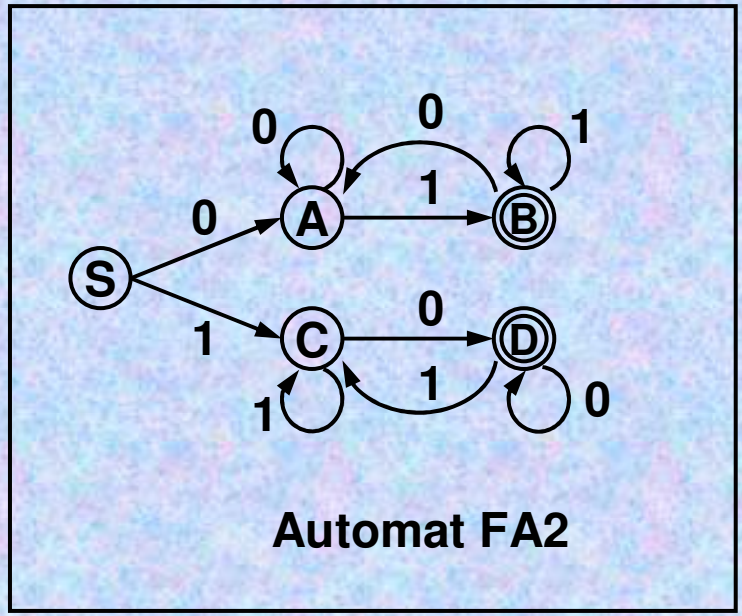
Všechna slova přijímaná automatem tvoří dohromady

jazyk přijímaný (rozpoznávaný) automatem

Konečný automat

Jazyk nad abecedou {0,1} :

Pokud slovo začíná 0, končí 1,
pokud slovo začíná 1, končí 0.



Automat FA2

Ukázka analýzy slov automatem FA2:

0 1 0 1 0 : (S),0 → (A),1 → (B),0 → (A),1 → (B),0 → (A)

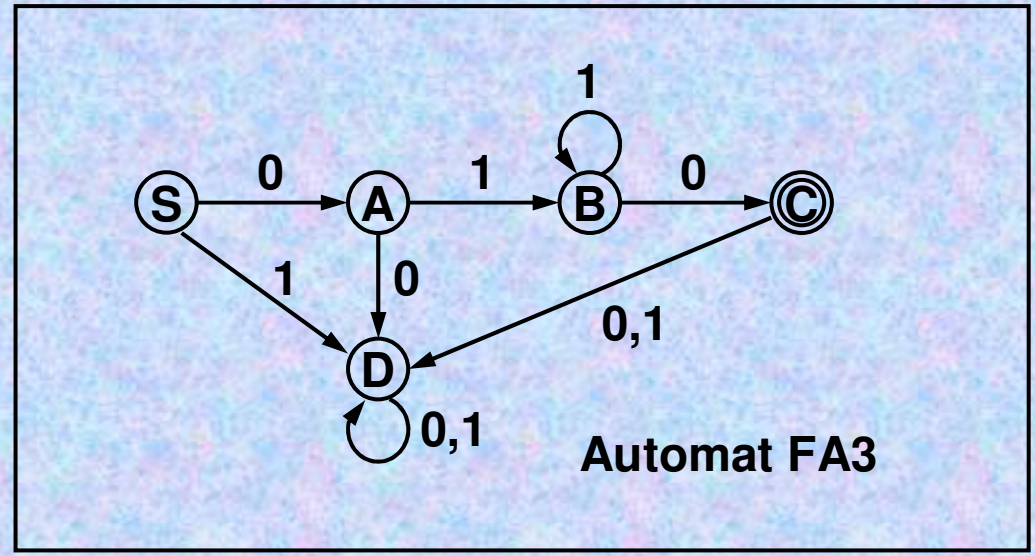
(A) není koncový stav, slovo 0 1 0 1 0 automat FA2 nepřijímá.

1 0 1 1 0 : (S),1 → (C),0 → (D),1 → (C),1 → (C),0 → (D)

(D) je koncový stav, slovo 1 0 1 1 0 automat FA2 přijímá.

Konečný automat

Jazyk:
{
0 1 0,
0 1 1 0,
0 1 1 1 0,
0 1 1 1 1 0,
0 1 1 1 1 1 0,
... **}**



Ukázka analýzy slov automatem FA3

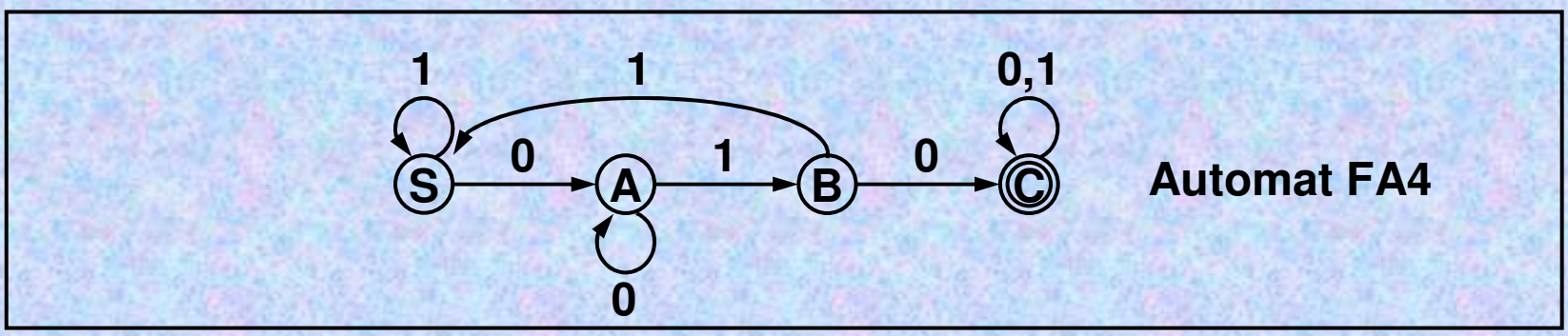
0 1 0 1 0 : (S),0 → (A),1 → (B),0 → (C),1 → (D),0 → (D)

(D) není koncový stav, slovo 0 1 0 1 0 automat FA3 nepřijímá.

0 1 1 1 0 : (S),0 → (A),1 → (B),1 → (B),1 → (B),0 → (C)

(C) je koncový stav, slovo 0 1 1 1 0 automat FA3 přijímá.

Konečný automat



Automat FA4 příjme každé slovo nad abecedou {0,1} obsahující podposloupnost ... 0 1 0 ...

Ukázka analýzy slov automatem FA4

0 0 1 0 1 : (S),0 → (A),0 → (A),1 → (B),0 → (C),1 → (C)

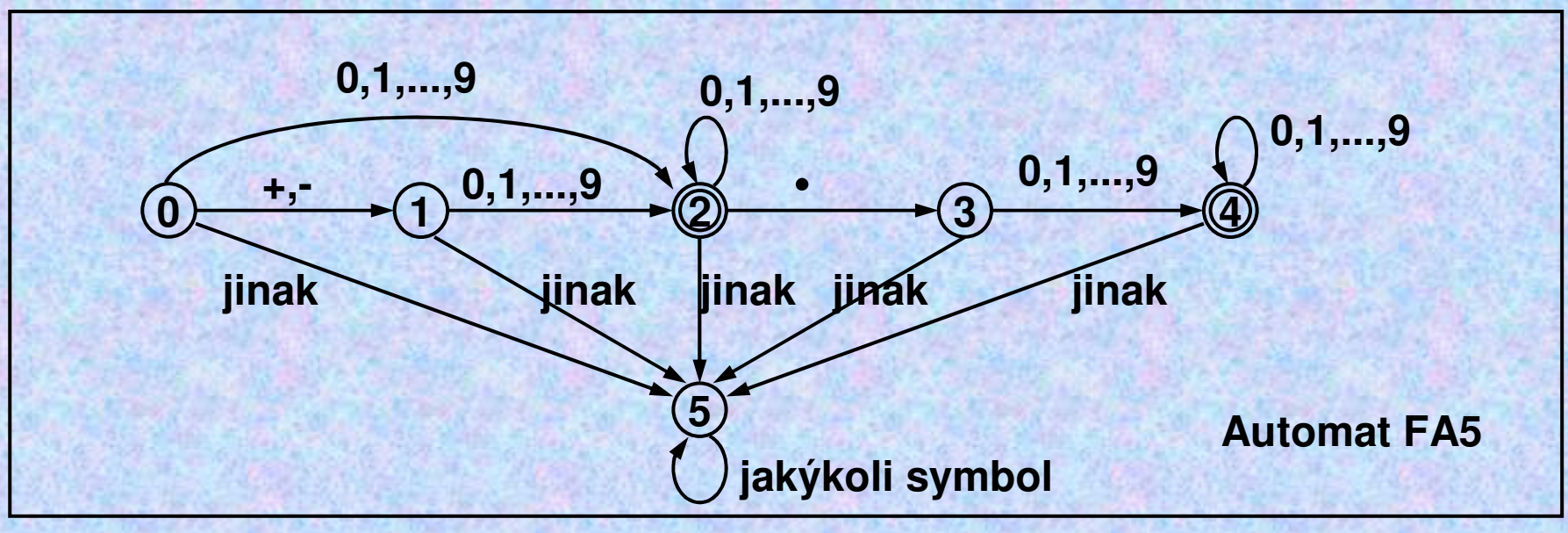
(C) je koncový stav, slovo 0 0 1 0 1 automat FA4 přijímá.

0 1 1 1 0 : (S),0 → (A),1 → (B),1 → (S),1 → (S),0 → (A)

(A) není koncový stav, slovo 0 1 1 1 0 automat FA4 nepřijímá.

Konečný automat

Jazyk nad abecedou { +, - , . , 0, 1, …, 8, 9, … } jehož slova představují zápis desetinného čísla v desítkové soustavě



Ukázka analýzy slov

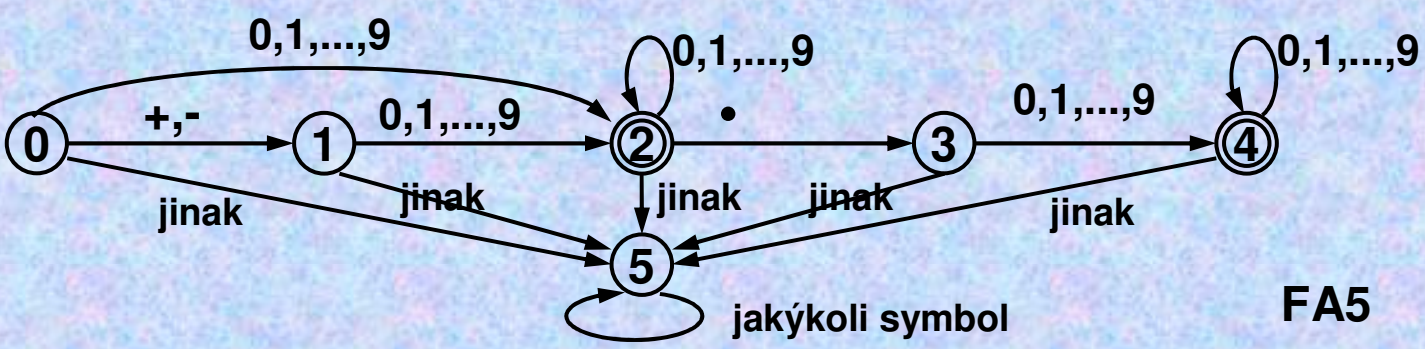
+87.09: (0),+ → (1),8 → (2),7 → (2), . → (3),0 → (4),9 → (4)

(4) je koncový stav, slovo +87.05 automat přijímá.

76+2: (0),7 → (2),6 → (2),+ → (5),2 → (5)

(5) není koncový stav, slovo 76+2 automat nepřijímá.

Implementace konečného automatu



FA5

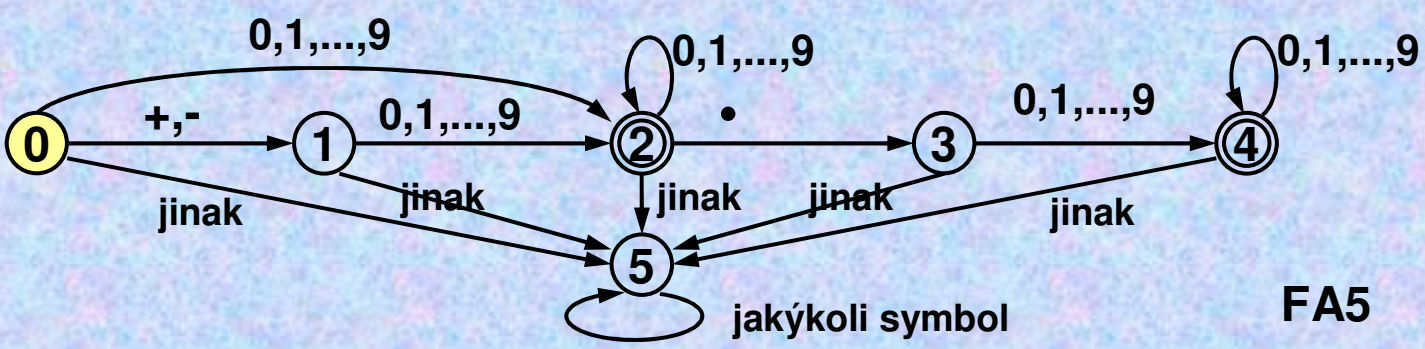
Kód konečného automatu

(Čtená posloupnost znaků je uložena v poli arr[ ]):

```
int isDecimal(int arr[], int length) {
int i;
int state = 0;

for(i = 0; i < length; i++) {  // check each symbol
  switch (state) {
  ...
```

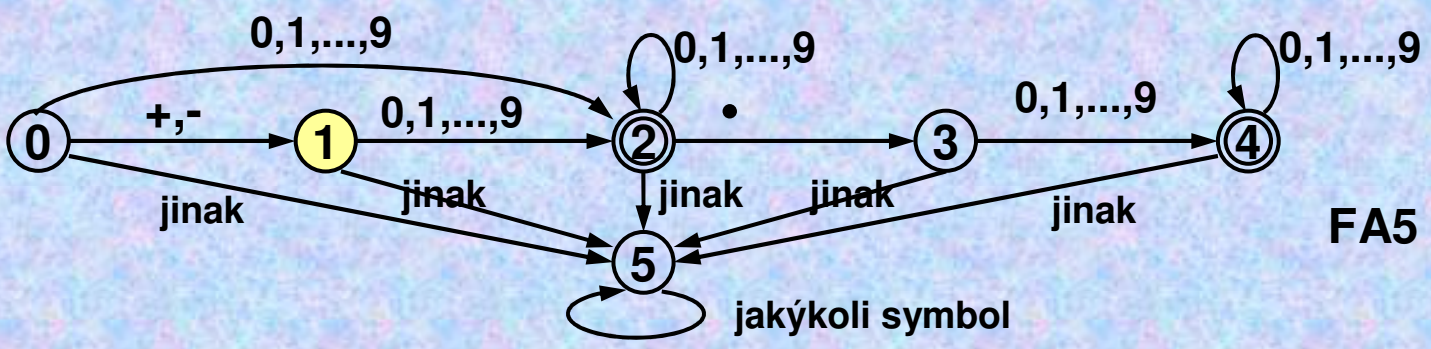
Implementace konečného automatu



FA5

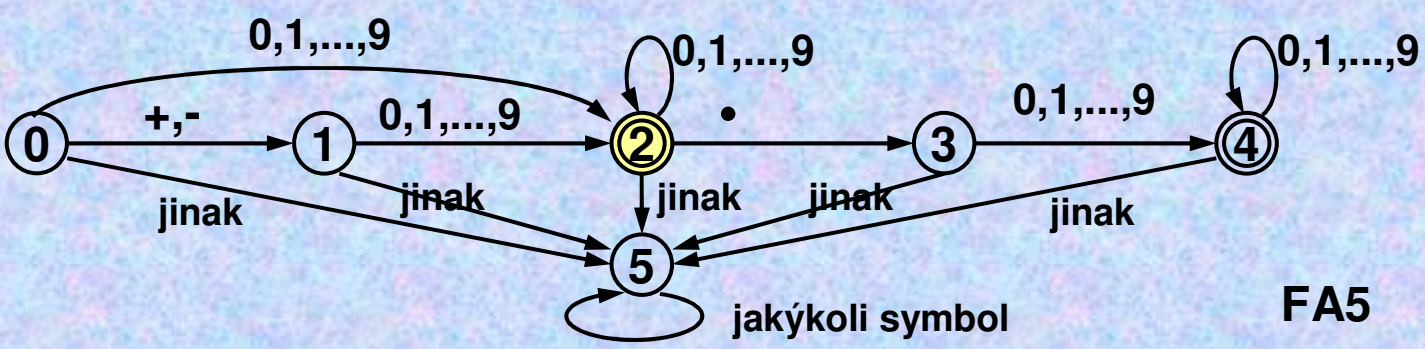
```
case 0:
  if ((arr[i] == '+') || (arr[i] == '-')) state = 1;
  else
  if ((arr[i] >= '0') && (arr[i] <= '9')) state = 2;
  else state = 5;
  break;
```

Implementace konečného automatu



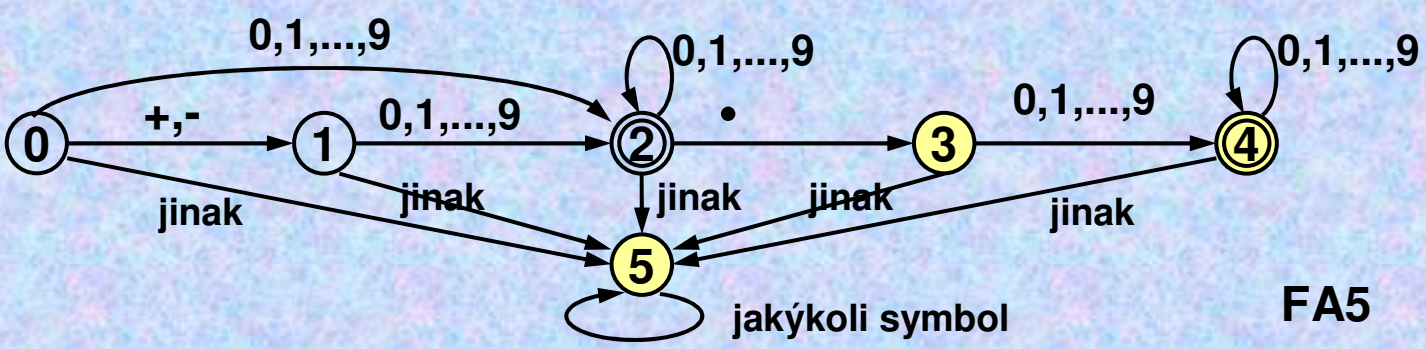

```
case 1:
  if ((arr[i] >= '0') && (arr[i] <= '9')) state = 2;
  else state = 5;
  break;
```

Implementace konečného automatu



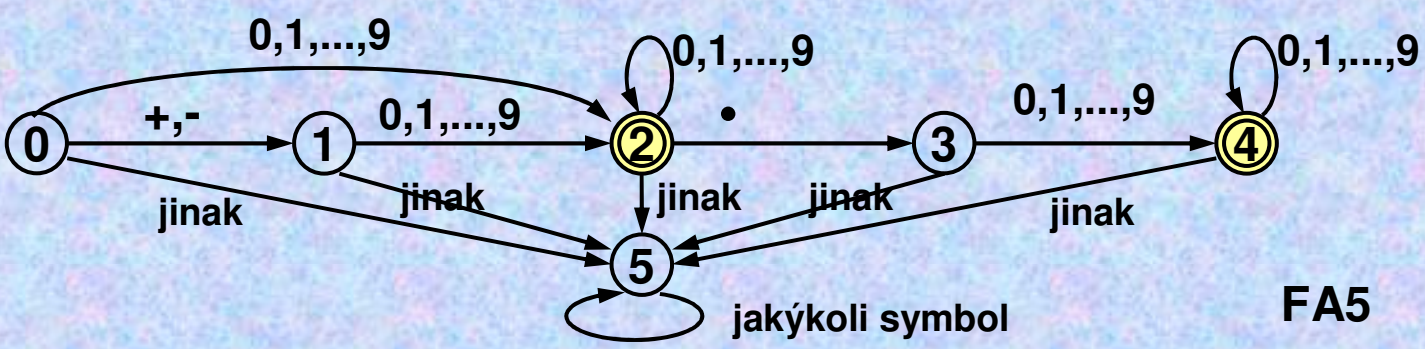

```
case 2:
  if ((arr[i] >= '0') && (arr[i] <= '9')) state = 2;
  else
  if (arr[i] == '.') state = 3;
  else state = 5;
  break;
```

Implementace konečného automatu




```
③ case 3:
     if ((arr[i] >= '0') && (arr[i] <= '9')) state = 4;
     else state = 5;
     break;
④ case 4:
     if ((arr[i] >= '0') && (arr[i] <= '9')) state = 4;
     else state = 5;
     break;
⑤ case 5: break;  // no need to react anyhow
   default : break;
 } // end of switch
```

Implementace konečného automatu




```
  } // end of for loop -- word has been read

if ((state == 2) || (state == 4)) // final states!!
     return 1;                    // success - decimal OK
else
     return 0;                    // not a decimal

} // end of function isDecimal()
```

Složitost rekurzivních algoritmů

Metoda

- stromu rekurze
- substituční
- mistrovská

Složitost rekurzivních algoritmů

T(n) -- asymptotická složitost algoritmu při vstupu o velikosti n

Př.: Merge sort

Asymptotická složitost vyřešení triviální úlohy

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

Jak asymptotická složitost při vstupu o velikosti n závisí na asymptotické složitosti při vstupu o velikosti n/2

Složitost rozdělení problému a spojení dílčích řešení (polovin pole v Merge sortu)

Složitost rekurzivních algoritmů

Co lze zanedbat

! typickou hodnotu n si vhodně zvolíme
(v Merge sortu mocninu 2)

! konkrétní konstanta neovlivní výslednou asymptotickou složitost

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

! n/2 a obecně n/konst není celé číslo, mysleme si však, že (víceméně) je,
a použijme jej místo správného $\lceil n/2 \rceil$ či $\lfloor n/2 \rfloor$ apod.

Složitost rekurzivních algoritmů

Příklad

Pro algoritmus A platí

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

Rozdělí data na čtvrtiny. Vyřešení „čtvrtinové“ úlohy trvá $T(\lfloor n/4 \rfloor)$.

Jedna čtvrtina se nezpracovává $\Rightarrow$ 3 se zpracují v čase $3T(\lfloor n/4 \rfloor)$.

Čas potřebný na rozdělení na čtvrtiny
a na spojení „čtvrtinových“ úloh je $\Theta(n^2)$.

Složitost rekurzivních algoritmů

Příklad

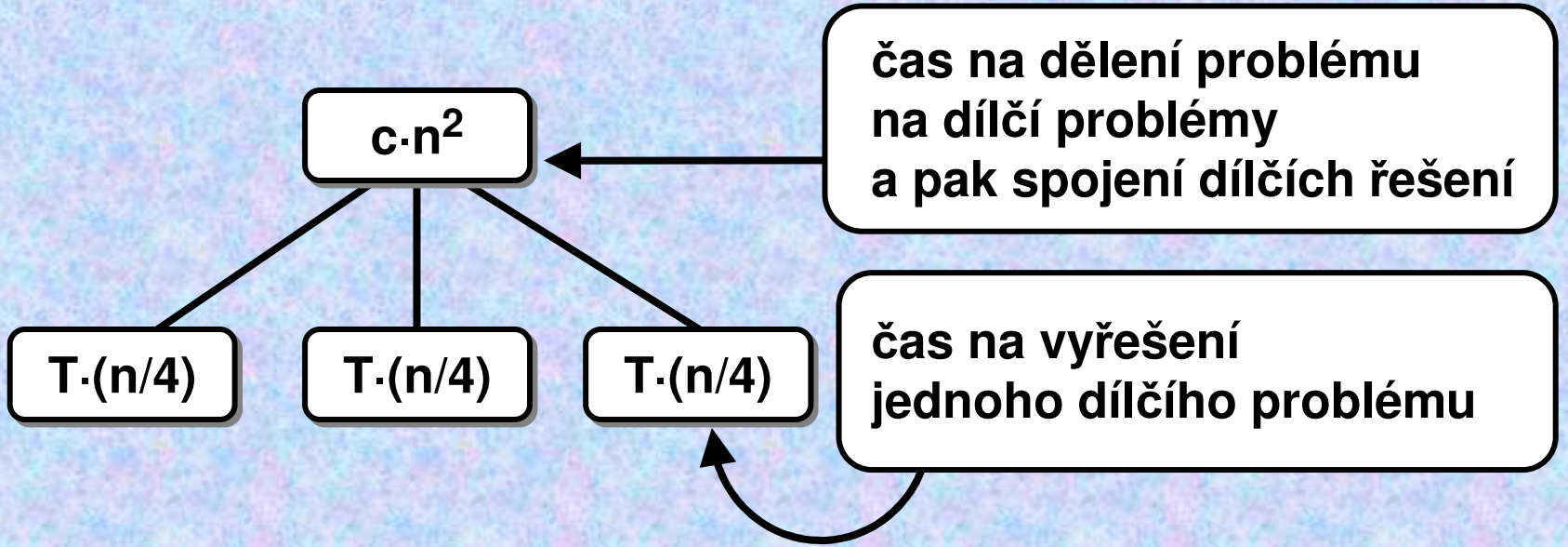
$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

Vztah pro výpočet

$$T(n) = 3T(n/4) + c \cdot n^2$$

Strom rekurze

$$T(n) = 3T(n/4) + c \cdot n^2$$

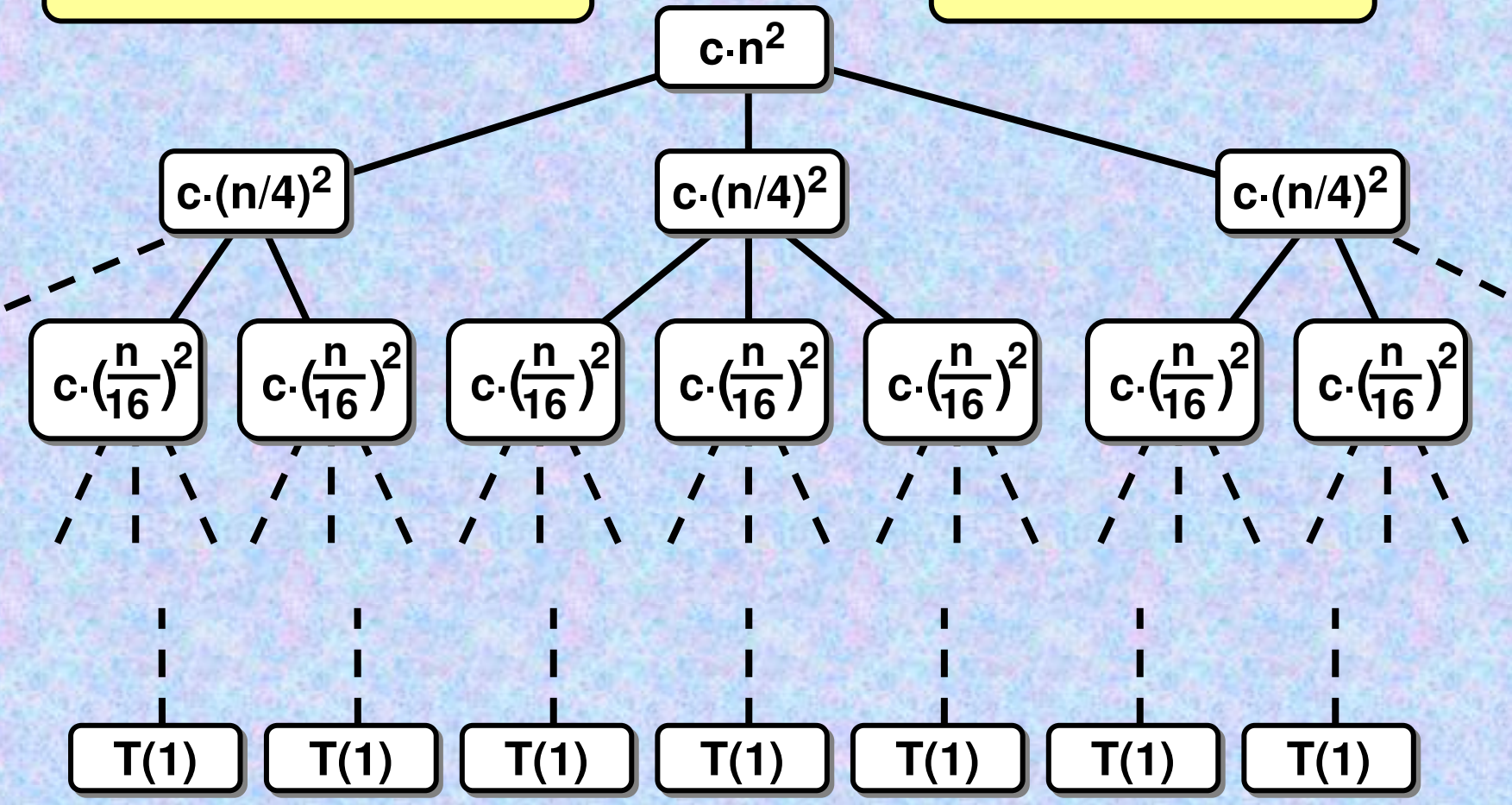


Strom rekurze je ale větší ...

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

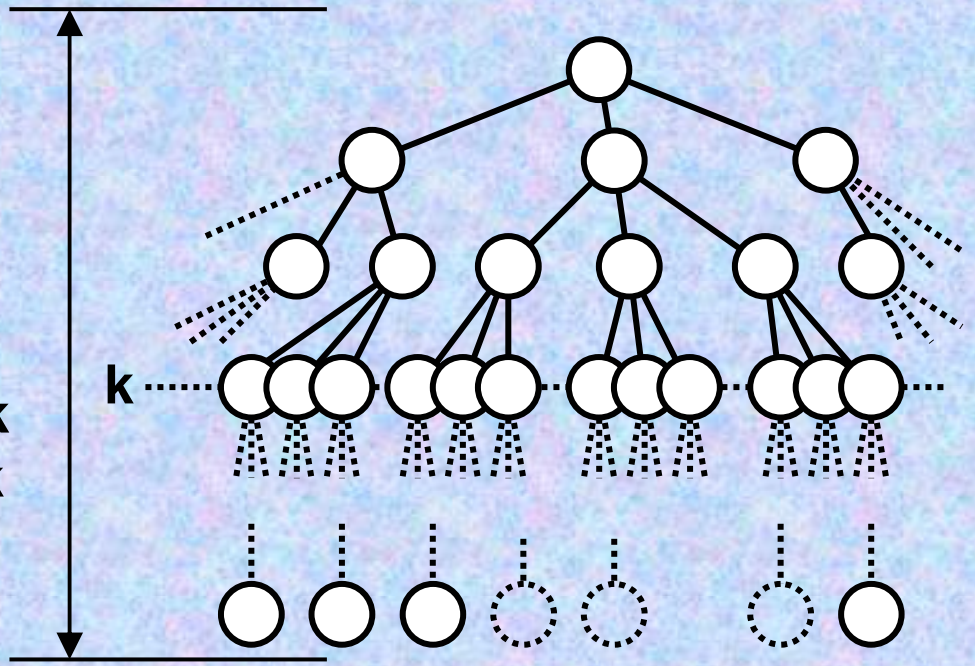
Strom rekurze



Strom rekurze

Průběh výpočtu

1. Nakresli strom rekurze
2. Spočti jeho hloubku
3. Spočti jeho šířku v patře k
4. Spočti cenu uzlu v patře k
5. Sečti ceny uzlů v patře k
6. Sečti ceny všech pater



cena uzlu = asymptotická složitost zpracování podproblému odpovídajícího uzlu ve stromu rekurze.

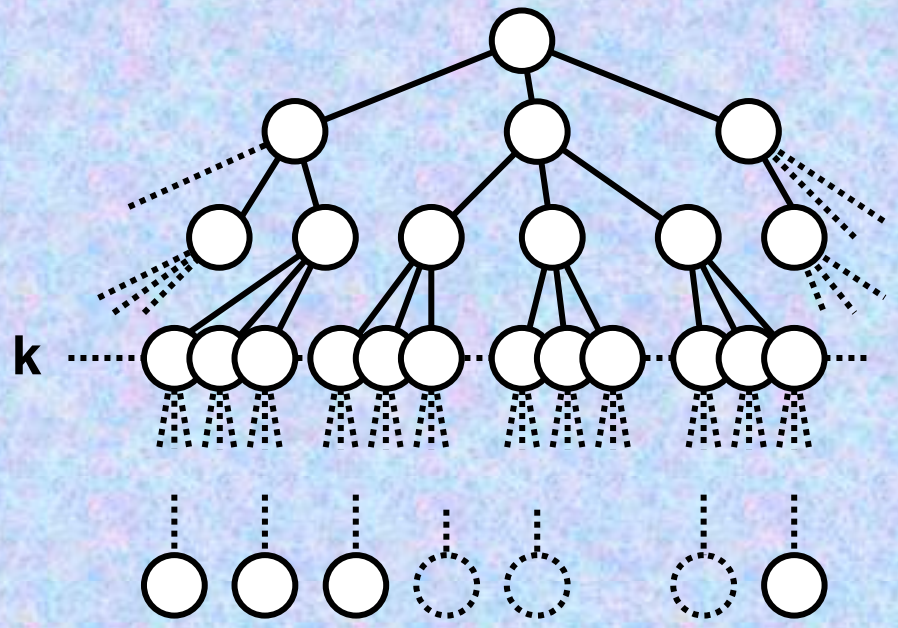
cena stromu = asymptotická složitost zpracování celé úlohy.

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

1. Nakresli strom rekurze
2. Spočti jeho hloubku
...



V hloubce k
je velikost podproblému = $n/4^k$.

Velikost podproblému je tedy = 1, když $n/4^k = 1$, tj $k = \log_4(n)$.

Takže **strom má log(4) + 1 pater.**

Strom rekurze

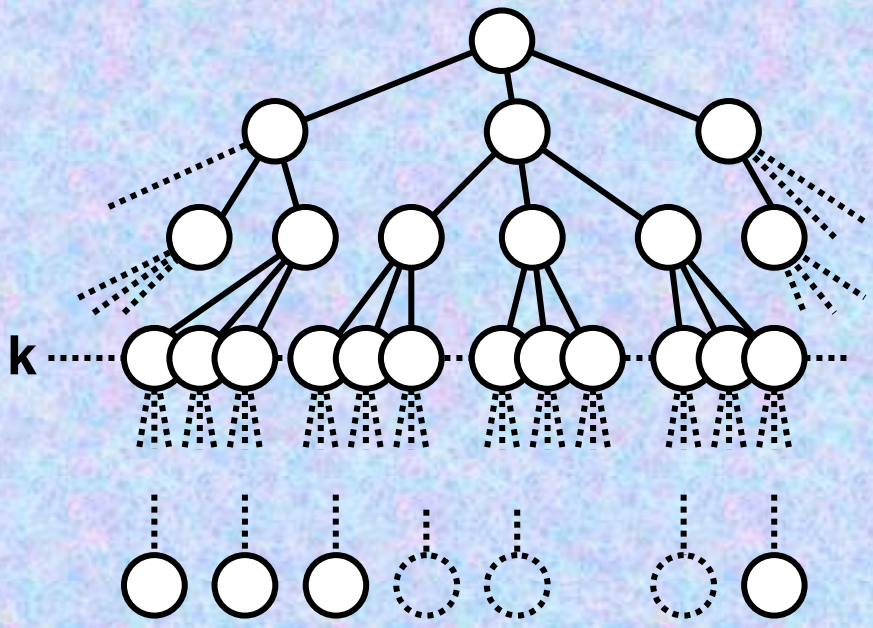
$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

...

3. Spočti jeho šířku v patře k

...



0. patro – 1 uzel
1. patro – 3 uzly
2. patro – 3·3 = 9 uzlů
3. patro – 3·3·3 = 27 uzlů

...

k. patro – 3·3· ... ·3·3 = 3^k uzlů

Strom rekurze

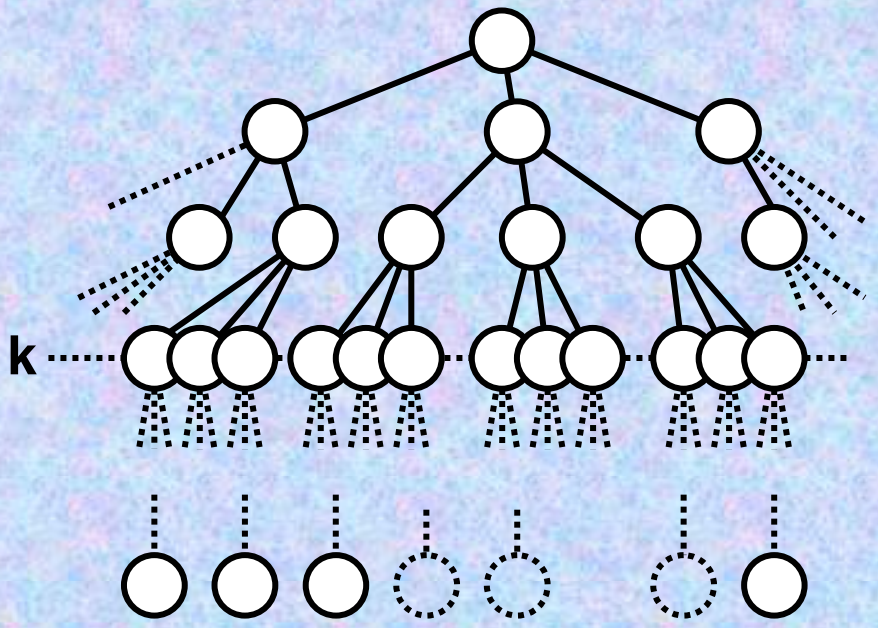
$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

...

4. Spočti cenu uzlu v patře k

...



0. patro – $c \cdot n^2$
1. patro – $c \cdot (n/4)^2$
2. patro – $c \cdot (n/16)^2$
3. patro – $c \cdot (n/64)^2$

...

k. patro – $c \cdot (n/4^k)^2$

Strom rekurze

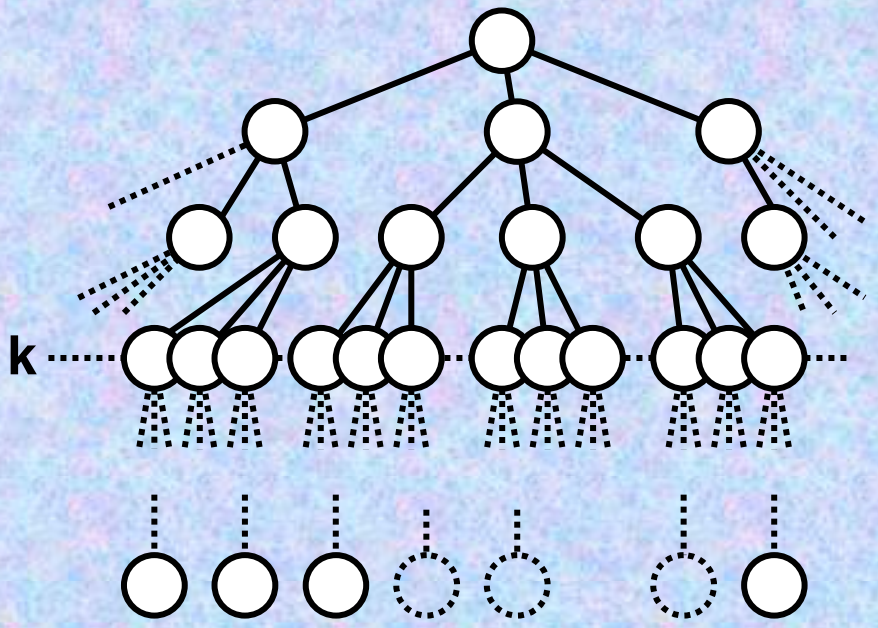
$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

...

5. Sečti ceny uzlů v patře k

...



V patře k je 3^k uzlů, každý má cenu $c \cdot (n/4^k)^2$.

Celková cena patra k je

$3^k \cdot c \cdot (n/4^k)^2 = (3/16)^k \cdot c \cdot n^2$

Pozor na poslední patro:

Strom rekurze

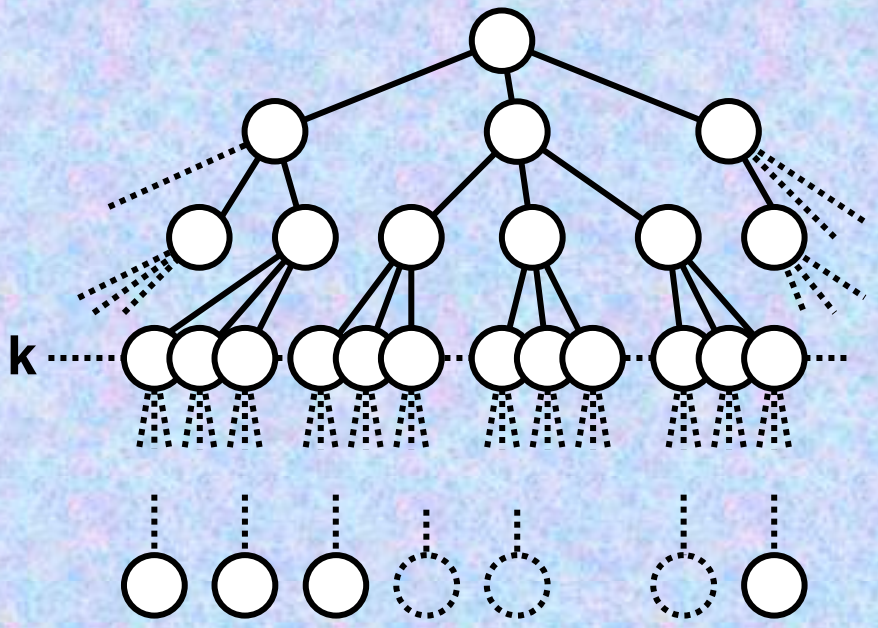
$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

...

5. Sečti ceny uzlů v patře k

...



Poslední patro je v hloubce $\log_4(n)$ a má tedy

$3^{\log_4(n)} = n^{\log_4(3)}$ uzlů.

Každý přispívá konstantní cenou, takže cena posledního patra je

$n^{\log_4(3)} \cdot konst = \Theta(n^{\log_4(3)})$

Strom rekurze

Průběh výpočtu
...
6. Sečti ceny všech pater

$$T(n) = 3T(n/4) + c \cdot n^2$$

Celková cena =

$$cn^2 + 3/16 \cdot cn^2 + (3/16)^2 \cdot cn^2 + \ldots + (3/16)^{\log_4(n-1)} \cdot cn^2 + \Theta(n^{\log_4(3)}) =$$

$$(1 + 3/16 \cdot + (3/16)^2 + \ldots + (3/16)^{\log_4(n-1)}) \cdot cn^2 + \Theta(n^{\log_4(3)}) =$$

Geometrickou posloupnost nahradíme přibližně geometrickou řadou (zbytek řady je zanedbatelný). Získáváme horní odhad součtu.

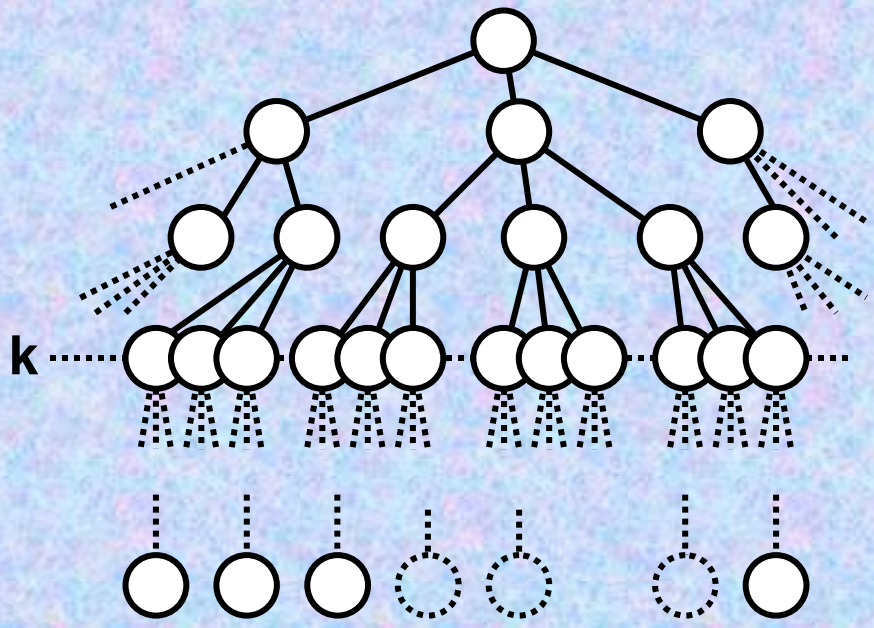
$$(1 + 3/16 \cdot + (3/16)^2 + (3/16)^3 + \ldots \text{ ad inf. }) \cdot cn^2 + \Theta(n^{\log_4(3)}) =$$

$$(1 / (1 - 3/16)) \cdot cn^2 + \Theta(n^{\log_4(3)}) = 16/13 \cdot cn^2 + \Theta(n^{\log_4(3)}) =$$

Strom rekurze

Průběh výpočtu
...
6. Sečti ceny všech pater

$T(n) = 3T(n/4) + c \cdot n^2$



$2 > \log_4(3)$

$16/13 \cdot cn^2 + \Theta(n^{\log_4(3)}) = \Theta(n^2)$

Asymptotická složitost celého algoritmu A.

Substituční metoda

Příklad

Rekurentní vztah popisující asymptotickou složitost algoritmu B

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Náš odhad složitosti

$$T(n) = O(n \cdot \log_2(n))$$

Zdroj odhadu: zkušenost, podobnost s jinými úlohami
úvaha, intuice ... :-)

Chceme dokázat: Náš odhad platí

Metoda: Běžná matematická indukce, do níž dosadíme (substituujeme) daný rekurentní vztah

Substituční metoda

Chceme dokázat : $T(n) = O(n \cdot \log_2(n))$,
to jest : $T(n) \leq c \cdot n \cdot \log_2(n)$, pro vhodné $c > 0$

Krok II (obecný krok) matematické indukce:

Dokážeme, že pokud nerovnost platí pro $\lfloor n/2 \rfloor$, platí i pro n.

Víme: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Předpokládáme: $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor)$

Substituce: $T(n) \leq 2 \cdot c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor) + n$

úpravy: $\leq cn \cdot \log_2(n/2) + n = cn \cdot \log_2(n) - cn \cdot \log_2(2) + n$

$= cn \cdot \log_2(n) - cn + n$

$\leq cn \cdot \log_2(n)$, pokud $c \geq 1$

Substituční metoda

Krok I matematické indukce:

nerovnost $T(n) \leq c \cdot n \cdot \log_2(n)$, platí pro nějaké konkrétní malé n.

Nelze dokazovat pro n = 1,

neboť bychom dokazovali $T(1) \leq c \cdot 1 \cdot \log_2(1) = 0$, což neplatí, protože jistě je $T(1) > 0$.

Pozorování

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Pokud n > 3, v rekurentním vztahu se T(1) neobjeví,
tedy pokud dokážeme indukční krok I pro n = 2 a n = 3,
je důkaz hotov pro všechna $n \geq 2$.

Jde nám ale o **asymptotickou složitost,** tudíž důkaz pro $n \geq 2$ stačí.

Substituční metoda

Krok I matematické indukce pro n = 2 a n = 3

Ať $T(1) = k$.

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Z rekurentního vztahu plyne
$T(2) = 2k+2$
$T(3) = 2k+3$

Chceme mít:
$T(2) \leq c \cdot 2 \cdot \log_2(2)$
$T(3) \leq c \cdot 3 \cdot \log_2(3)$

Stačí tedy volit $c \geq \max \{ (2k+2)/2,\ (2k+3)/(3 \cdot \log_2(3)) \}$.

Mistrovská metoda

Přímočará aplikace věty:

Nechť $a \geq 1$ a $b > 1$ jsou konstanty, $f(n)$ je funkce a nechť asymptotická složitost daného algoritmu je definována rekurentním vztahem

$$T(n) = aT(n/b) + f(n),$$

kde podíl n/b lze libovolně interpretovat jako $\lfloor n/b \rfloor$ nebo $\lceil n/b \rceil$.

Potom platí

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak $T(n) = \Theta(n^{\log_b(a)})$.

   // Podmínka 1. říká, že $f(n)$ musí růst polynomiálně pomaleji
   // než funkce $n^{\log_b(a)}$.

Mistrovská metoda

2. Pokud $f(n) = \Theta(n^{\log_b(a)})$, pak $T(n) = \Theta(n^{\log_b(a)})$.

3. Pokud $f(n) = \Omega(n^{\log_b(a)+e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 0$ a pro všechna dostatečně velká n, pak
   $T(n) = \Theta(f(n))$.

// Podmínka 3. říká, že f(n) musí růst polynomiálně rychleji
// než funkce $n^{\log_b(a)}$.

Mistrovská metoda

...

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak $T(n) = \Theta(n^{\log_b(a)})$.

...

Příklad.

$T(n) = 9T(n/3) + n$ $\quad a = 9, b = 3, f(n) = n.$

Platí $\quad n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2)$

$f(n) = O(n^{\log_3(9) - e})$, kde $e = 1$

Celkem tedy $T(n) = \Theta(n^2)$

Mistrovská metoda

...

3. Pokud $f(n) = \Omega(n^{\log_b(a)+e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 0$ a pro všechna dostatečně velká n, pak
   $T(n) = \Theta(f(n))$.

Příklad.

$T(n) = 2T(n/2) + n \cdot \log_2(n)$

$a = 2$, $b = 2$, $f(n) = n \cdot \log_2(n)$.

Platí $n^{\log_b(a)} = n$

$f(n) = n \cdot \log_2(n)$ roste asymptoticky rychleji než $n^{\log_b(a)} = n$.

Pozor, neroste ale polynomiálně rychleji. Poměr $n \cdot \log_2(n) / n = \log_2(n)$ roste pomaleji než každá rostoucí polynomiální funkce. $n \cdot \log2(n) \notin \Omega(n1+e)$ pro každé kladné e. Předpoklad 3. není splněn.

DSA

Různé algoritmy mají různou složitost