

Úvod do teoretické informatiky

Martin Kot Zdeněk Sawa

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

1. března 2006

Jméno: Zdeněk Sawa

E-mail: zdenek.sawa@vsb.cz

Místnost: KP314

WWW: <http://www.cs.vsb.cz/sawa/>

Tel.: 1424

Jméno: Martin Kot

E-mail: martin.kot@vsb.cz

Místnost: KP314

WWW: <http://www.cs.vsb.cz/kot/>

Tel.: 1424

Povinné:

- **Zápočet** (35 bodů):
 - Zápočtová písemka (20 bodů)
 - Vypracování zápočtového příkladu (15 bodů):
 - Písemné vypracování (ve formátu PDF) (10 bodů)
 - Předvedení na cvičení (5 bodů)

- **Zkouška** (65 bodů)

Nepovinné: (bonusové body)

- Vypracování prémiového příkladu (15 bodů)

Webové stránky k předmětu naleznete na adrese:

<http://www.cs.vsb.cz/kot/uti>

Na těchto stránkách najdete:

- Informace o předmětu
- Sylaby (autorem je doc. Petr Hliněný)
- Zadání příkladů na cvičení
- Zadání zápočtových příkladů
- Zadání prémiových příkladů
- Slidy z přednášek
- Aktuální informace

- Přednášky jsou číslovány od 1 do 11.

- Cvičení jsou číslována od 0 do 11.

Cvičení číslo i se vztahuje k přednášce číslo i a mělo by být následující týden po této přednášce.

- Čísla cvičení uvedená u zápočtových příkladů udávají, na kterém cvičení budete příklad předvádět.

K vypracování zápočtových příkladů ze cvičení číslo i by měly postačovat znalosti z přednášky a cvičení číslo $i - 1$.

Na vypracování zápočtového příkladu budete mít víc než týden času.

Zápočtové příklady začínají cvičením číslo 2.

- Body dostane vždy první student v ročníku, který daný příklad vyřeší.
- Řešení posílejte e-mailem přednášejícím, ne cvičícím:
 - Zdeněk Sawa (zdenek.sawa@vsb.cz)
 - Martin Kot (martin.kot@vsb.cz)
- Příklady budou zveřejňovány na webových stránkách postupně během semestru.
- Každý student může tímto způsobem získat 15 bodů navíc nejvýše jednou.

V tomto předmětu se budeme zabývat dvěma oblastmi teoretické informatiky:

- **Teorii formálních jazyků**
- **Výpočetní složitostí**

Teorie formálních jazyků

- **Přednáška 1:** Úvod, základní pojmy
- **Přednáška 2:** Konečné automaty
- **Přednáška 3:** Regulární výrazy
- **Přednáška 4:** Minimalizace automatů, omezení konečných automatů
- **Přednáška 5:** Bezkontextové gramatiky
- **Přednáška 6:** Zásobníkové automaty, Chomského hierarchie

Výpočetní složitost

- **Přednáška 7:** Algoritmy, výpočetní modely, Turingovy stroje
- **Přednáška 8:** Výpočetní složitost, analýza algoritmů
- **Přednáška 9:** Asymptotická notace, řešení rekurentních rovnic
- **Přednáška 10:** Třídy P a NP, NP-úplné problémy
- **Přednáška 11:** Důkazy NP-úplnosti některých problémů

Motivace 1: Vyhledávání v textu

Potřebujeme řešit následující problém:

- Máme řadu různých textů (např. soubory na disku nebo webové stránky apod.).
- Potřebujeme zjistit, které z těchto textů obsahují nějaké dané slovo či frázi, případně nějakou kombinaci slov apod.

Požadujeme, aby řešení bylo:

- **Rychlé** – můžeme prohledávat mnoho MB dat
- **Dostatečně obecné** – chceme mít možnost formulovat dostatečně obecné dotazy (např. mít možnost používat booleovské spojky)

Při popisu libovolného programovacího jazyka (Java, C, C++, Pascal, ...) musí být řečeno:

1 Co jsou jeho **lexikální elementy (tokeny)** –

- identifikátory
- klíčová slova
- literály (číselné a řetězcové konstanty)
- operátory
- oddělovače
- komentáře
- ...

a jak přesně vypadají.

2 Které sekvence těchto lexikálních elementů tvoří (syntakticky) dobře utvořené programy.

Chceme řešit následující problémy:

- Jak přesně (jednoznačně) popsat jednotlivé typy lexikálních elementů?
- Jak implementovat v překladači rozpoznávání těchto jednotlivých typů?
- Jak přesně popsat všechny možné způsoby jakými je možné z lexikálních elementů „poskládat“ syntakticky správně napsaný program?
- Jak implementovat v překladači rozpoznání dobře utvořených výrazů, příkazů, procedur, metod apod.?

Lexikální analýza – činnost překladače, kdy rozpoznává v textu jednotlivé lexikální elementy.

Syntaktická analýza – činnost překladače, kdy rozpoznává v dané sekvenci lexikálních elementů např. aritmetické výrazy, příkazy, podprogramy nebo i celé programy.

Co mají všechny dosud zmíněné problémy společného?

- Pracujeme se sekvencemi **znaků** (říkáme též **symbolů**).
- Znak patří do nějaké konečné **abecedy** (typicky např. ASCII nebo Unicode).
- Musíme rozpoznávat ty sekvence znaků, které nějakou vlastnost mají a ty co ji nemají.

Poznámka: V teorii formálních jazyků se sekvencím znaků říká **slova**. Při programování máme na mysli například řetězce (stringy) nebo třeba soubory na disku apod.

Množině slov z nějaké abecedy se říká **jazyk**.

Definice

Abeceda je libovolná (neprázdná) konečná množina **symbolů (znaků)**.

Poznámka: Abeceda se často označuje řeckým písmenem Σ (velké sigma).

Definice

Slovo v dané abecedě je libovolná posloupnost symbolů z této abecedy.

Příklad 1:

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

Slova v abecedě Σ : AHOJ ABRACADABRA ERROR

Příklad 2:

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

Slovo v abecedě Σ : HELLO_WORLD

Příklad 3:

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Slova v abecedě Σ : 0, 314159, 666, 65536

Příklad 4:

Slova v abecedě $\Sigma = \{0, 1\}$: 011010001, 111, 1010101010101010

Příklad 5:

Slova v abecedě $\Sigma = \{a, b\}$: *aababb*, *abbabbba*, *aaab*

Příklad 6:

Abeceda Σ je množina všech ASCII znaků.

Příklad slova:

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

```
#include_<stdio.h> ↔↔ int_<main() ↔ { ↔ <<<<<printf("He...
```

Délka slova je počet znaků ve slově.

Například délka slova *abaab* je 5.

Délku slova w označujeme $|w|$.

Pokud tedy např. $w = abaab$, pak $|w| = 5$.

Prázdné slovo je slovo délky 0, tj. neobsahující žádné znaky.

Prázdné slovo se označuje řeckým písmenem ε (epsilon).

(Pozn.: Někteří autoři používají pro označení prázdného slova místo ε řecké písmeno λ (lambda).)

$$|\varepsilon| = 0$$

Se slovy je možné provádět operaci **zřetězení**:

Například zřetězením slov **OST** a **RAVA** vznikne slovo **OSTRAVA**.

Operace zřetězení se označuje symbolem \cdot (podobně jako násobení). Tento symbol je možné vypouštět.

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

Zřetězení je **asociativní**, tj. pro libovolná tři slova u , v a w platí

$$(u \cdot v) \cdot w = u \cdot (v \cdot w)$$

což znamená, že při zápisu více zřetězení můžeme vypouštět závorky a psát například $w_1 \cdot w_2 \cdot w_3 \cdot w_4 \cdot w_5$ místo $(w_1 \cdot (w_2 \cdot w_3)) \cdot (w_4 \cdot w_5)$

Zřetězení slov

Zřetězení není **komutativní**, tj. obecně pro dvojici slov u a v neplatí rovnost

$$u \cdot v = v \cdot u$$

Příklad:

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

Zjevně pro libovolná slova v a w platí:

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

Pro libovolné slovo w také platí:

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

Množinu všech slov tvořených symboly z abecedy Σ označujeme Σ^* .

Definice

(Formální) jazyk v abecedě Σ je nějaká libovolná podmnožina množiny Σ^* .

Příklad 1: Množina $\{00, 01001, 1101\}$ je jazyk v abecedě $\{0, 1\}$

Příklad 2: Množina všech syntakticky správných programů v jazyce C je jazyk v abecedě tvořené množinou všech ASCII znaků.

Příklad 3: Množina všech textů obsahujících sekvenci znaků **ABRACADABRA** je jazyk v abecedě tvořené množinou všech ASCII znaků.

Příklad 4: Uvažujme abecedu Σ tvořenou množinou všech Unicode znaků.

Množina všech komentářů v jazyce Java tvoří jazyk:

- jednořádkové komentáře začínající dvojicí znaků `//` a končící znakem konce řádku (nebo koncem souboru).
- víceřádkové komentáře začínající dvojicí znaků `/*` a končící dvojicí znaků `*/`, přičemž uvnitř se nesmí nacházet žádná další dvojice znaků `*/`.

Pokud bychom množinu všech jednořádkových komentářů označili jako jazyk L_1 a množinu všech víceřádkových komentářů jako jazyk L_2 , můžeme množinu všech komentářů označit jako jazyk L , definovaný jako

$$L = L_1 \cup L_2$$

Množinové operace na jazycích

Vzhledem k tomu, že jazyky jsou množiny, můžeme s nimi provádět množinové operace:

Sjednocení – $L_1 \cup L_2$ je jazyk tvořený slovy, která patří buď do jazyka L_1 nebo do jazyka L_2 (nebo do obou).

Průnik – $L_1 \cap L_2$ je jazyk tvořený slovy, která patří současně do jazyka L_1 i do jazyka L_2 .

Doplňěk – \bar{L} je jazyk tvořený těmi slovy ze Σ^* , která nepatří do L .

Rozdíl – $L_1 - L_2$ je jazyk tvořený slovy, která patří do L_1 , ale nepatří do L_2 .

Poznámka: Při operacích nad jazyky předpokládáme, že jazyky, se kterými operaci provádíme, používají tutéž abecedu Σ .

Příklad:

Uvažujme množinu všech textů tvořených ASCII znaky. Jestliže:

- L_1 je množina všech textů, ve kterých se vyskytuje sekvence znaků **FOO**, a
- L_2 je množina všech textů, ve kterých se vyskytuje sekvence znaků **BAR**,

pak

- $L_1 \cup L_2$ jsou všechny texty, ve kterých se vyskytuje **FOO** nebo **BAR**,
- $L_1 \cap L_2$ jsou všechny texty, ve kterých se vyskytuje **FOO** i **BAR**,
- $\overline{L_1}$ jsou všechny texty, ve kterých se nevyskytuje **FOO**,
- $L_1 - L_2$ jsou všechny texty, ve kterých se vyskytuje **FOO**, ale nevyskytuje **BAR**.

Definice

Zřetězení jazyků L_1 a L_2 je jazyk

$$L = \{uv \mid u \in L_1, v \in L_2\}$$

tj. jazyk všech slov, která začínají slovem z L_1 a pokračují slovem z L_2 .
Zřetězení jazyků L_1 a L_2 označujeme $L_1 \cdot L_2$.

Příklad:

$$L_1 = \{abb, ba\}$$

$$L_2 = \{a, ab, bbb\}$$

Jazyk $L_1 \cdot L_2$ obsahuje slova:

abba

abbab

abbbbb

baa

baab

babbb

Příklad:

Pro nějaký programovací jazyk chceme definovat, jak mohou vypadat konstanty reprezentující čísla v plovoucí řádové čárce (floating-point), např.:

1e1 2.0 .3 0.0 3.14 1E-9 4.5e137

Abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., e, E, +, -\}$

Příklad:

Pro nějaký programovací jazyk chceme definovat, jak mohou vypadat konstanty reprezentující čísla v plovoucí řádové čárce (floating-point), např.:

1e1 2.0 .3 0.0 3.14 1E-9 4.5e137

Abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., e, E, +, -\}$

Pokud zvolíme L_{num} jako množinu všech (neprázdných) slov tvořených pouze číslicemi, a $L_{dot} = \{.\}$, můžeme konstanty jako například 3467.982, 3.141592 nebo 0.0 popsat takto:

$$L_{num} \cdot L_{dot} \cdot L_{num}$$

Definice jazyka L všech možných konstant v plovoucí řádové čárce by pak mohla vypadat například takto:

$$\begin{aligned} L = & L_{num} \cdot L_{dot} \cdot (L_{num} \cup \{\varepsilon\}) \cdot (L_{exp} \cup \{\varepsilon\}) \cup \\ & L_{dot} \cdot L_{num} \cdot (L_{exp} \cup \{\varepsilon\}) \cup \\ & L_{num} \cdot L_{exp} \end{aligned}$$

kde

$$\begin{aligned} L_{num} & - \text{neprázdné sekvence číslic} \\ L_{dot} & = \{.\} \\ L_{exp} & = L_E \cdot L_{sign} \cdot L_{num} \\ L_E & = \{E, e\} \\ L_{sign} & = \{\varepsilon, +, -\} \end{aligned}$$

Používáme následující zápis:

$$\begin{aligned}L^2 &= L \cdot L \\L^3 &= L \cdot L \cdot L \\L^4 &= L \cdot L \cdot L \cdot L \\L^5 &= L \cdot L \cdot L \cdot L \cdot L \\&\dots\end{aligned}$$

Příklad: Pokud $L = \{aa, b\}$, pak L^3 obsahuje slova:

aaaaaa aaaab aabaa aabb baaaa baab bbaa bbb

Definujeme

$$\begin{aligned}L^1 &= L \\L^0 &= \{\varepsilon\}\end{aligned}$$

Iterace jazyka

Induktivní definice pro libovolné $k \geq 0$:

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

Definice

Iterace jazyka L je jazyk

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$$

tj. jazyk tvořený slovy vzniklými zřetěžením libovolného počtu slov z jazyka L .

Příklad: $L = \{aa, b\}$

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

Příklad: $L_{dig} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

$$L_{num} = L_{dig} \cdot L_{dig}^*$$

Poznámka: Používá se také následující značení:

$$L^+ = L^1 \cup L^2 \cup L^3 \cup L^4 \cup \dots$$

Řešení předchozího příkladu bychom tedy také mohli zapsat stručněji:

$$L_{num} = L_{dig}^+$$

Když chceme nějaký jazyk popsat, máme několik možností:

- Můžeme vyjmenovat všechna jeho slova (což je ale použitelné jen pro konečné jazyky).

Příklad: $L = \{aab, babba, aaaaaa\}$

- Můžeme specifikovat nějakou vlastnost, kterou mají právě ta slova, která do tohoto jazyka patří:

Příklad: Jazyk nad abecedou $\{0, 1\}$, obsahující všechna slova, ve kterých je počet výskytů symbolu 1 sudý.

V teorii formálních jazyků se používají především následující dva přístupy:

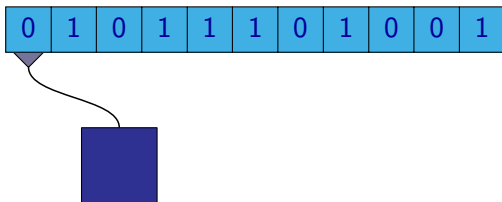
- Popsat (idealizovaný) stroj, zařízení, algoritmus, který rozpozná slova patřící do daného jazyka – vede k použití tzv. **automatů**.
- Popsat postup, jak mechanicky generovat všechna možná slova patřící do daného jazyka – vede k tzv. **gramatikám** a **regulárním výrazům**. (budeme se jim věnovat později)

Rozpoznávání jazyka

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

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

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

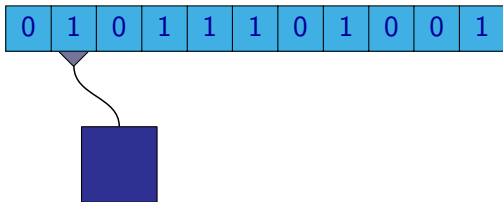


Rozpoznávání jazyka

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

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

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

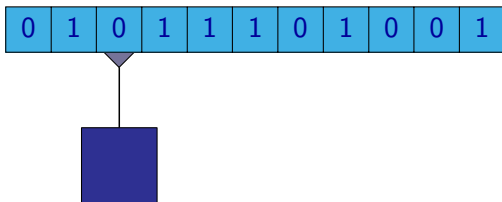


Rozpoznávání jazyka

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

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

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

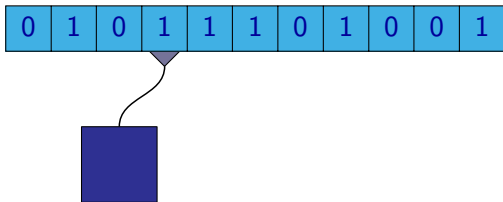


Rozpoznávání jazyka

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

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

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

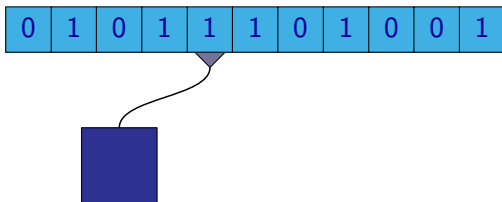


Rozpoznávání jazyka

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

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

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

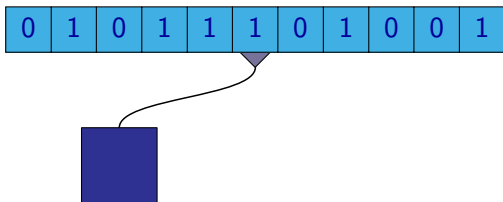


Rozpoznávání jazyka

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

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

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

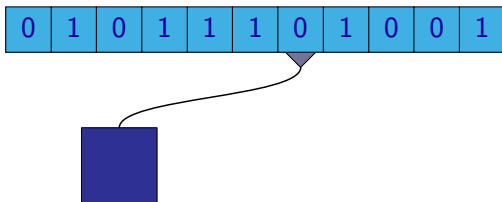


Rozpoznávání jazyka

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

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

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

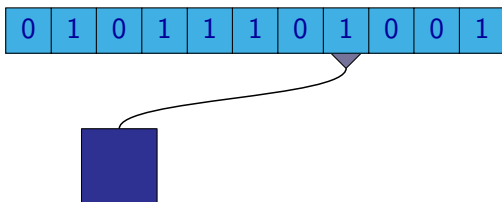


Rozpoznávání jazyka

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

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

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

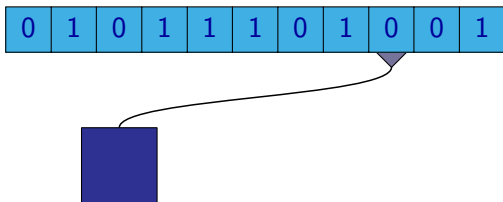


Rozpoznávání jazyka

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

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

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

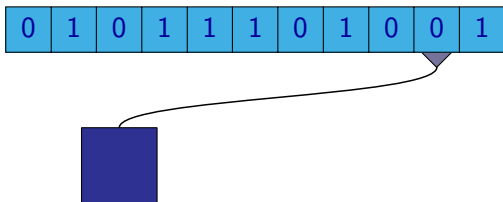


Rozpoznávání jazyka

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

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

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

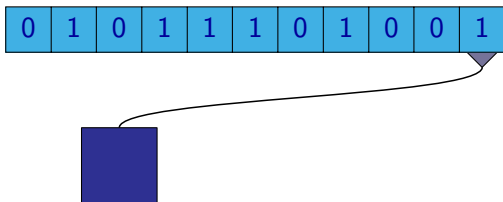


Rozpoznávání jazyka

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

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

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

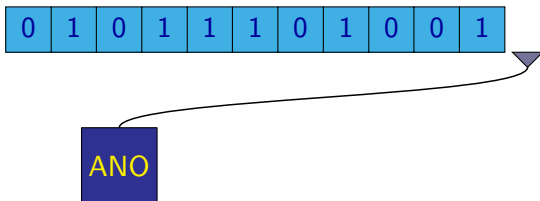


Rozpoznávání jazyka

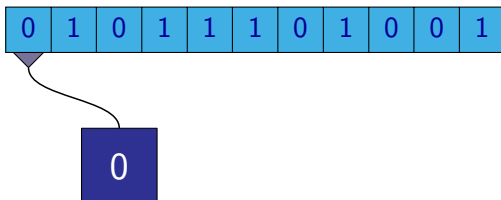
Příklad: Uvažujme slova nad abecedou $\{0, 1\}$.

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

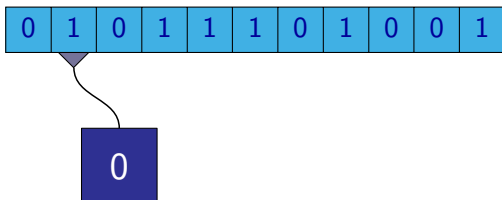
Chceme navrhnout zařízení, které přečte slovo, a sdělí nám, zda toto slovo patří do jazyka L či ne.



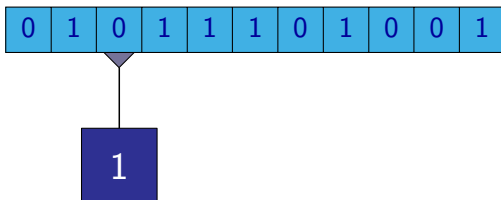
První nápad: Počítat počet výskytů symbolů 1.



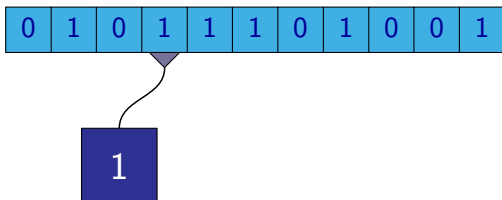
První nápad: Počítat počet výskytů symbolů 1.



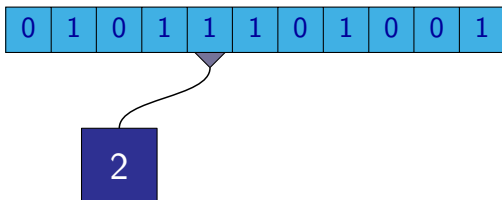
První nápad: Počítat počet výskytů symbolů 1.



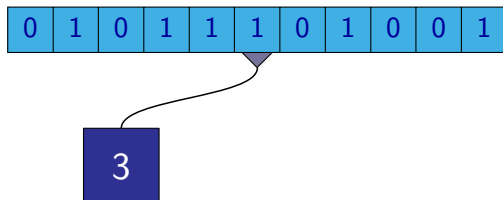
První nápad: Počítat počet výskytů symbolů 1.



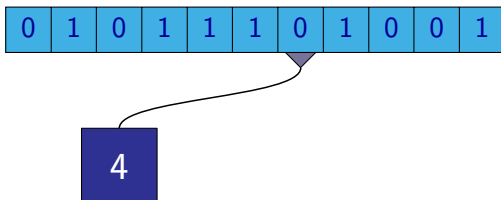
První nápad: Počítat počet výskytů symbolů 1.



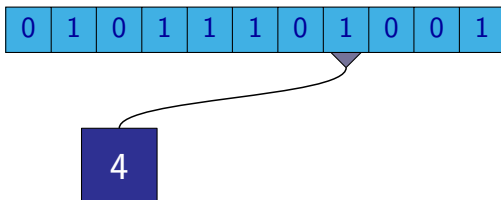
První nápad: Počítat počet výskytů symbolů 1.



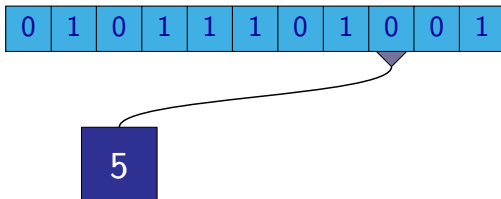
První nápad: Počítat počet výskytů symbolů 1.



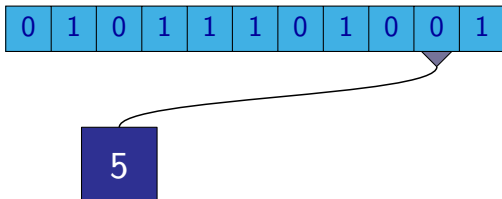
První nápad: Počítat počet výskytů symbolů 1.



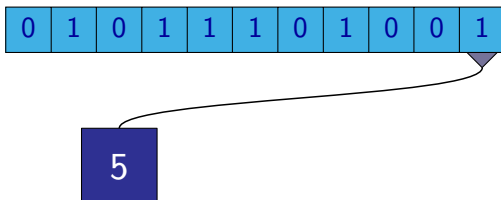
První nápad: Počítat počet výskytů symbolů 1.



První nápad: Počítat počet výskytů symbolů 1.



První nápad: Počítat počet výskytů symbolů 1.



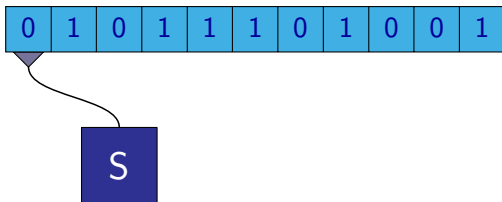
První nápad: Počítat počet výskytů symbolů 1.

0	1	0	1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---

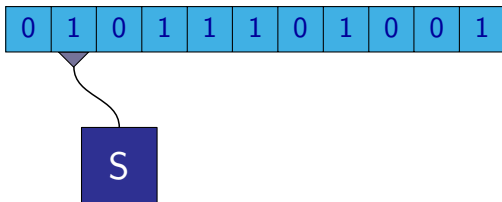
6

ANO – 6 je sudé číslo

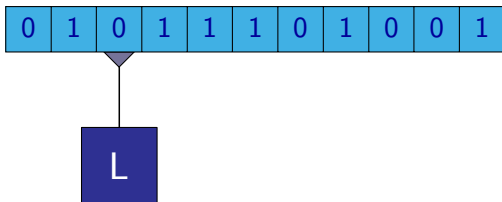
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



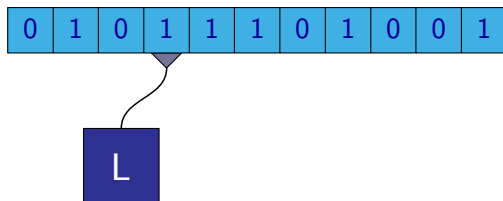
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



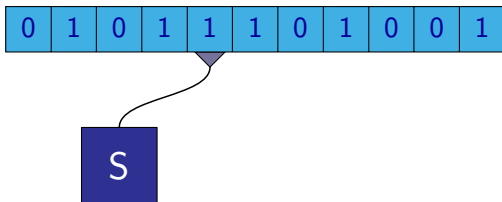
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



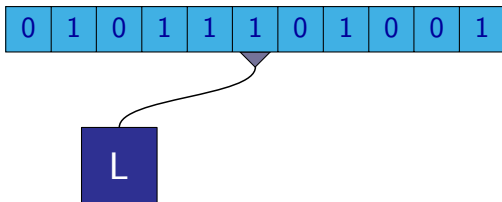
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



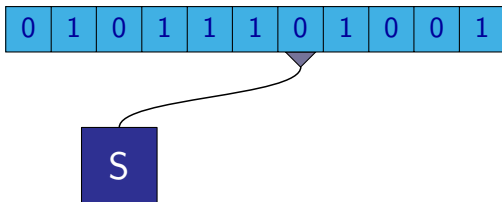
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



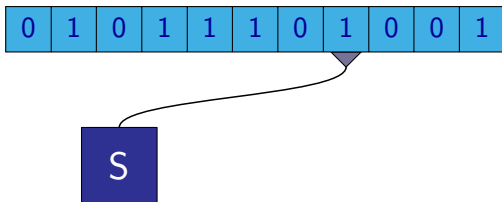
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



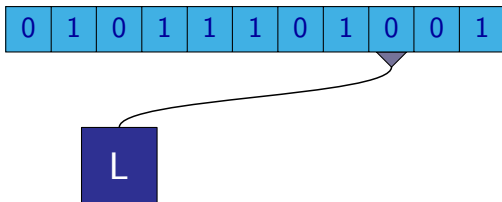
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



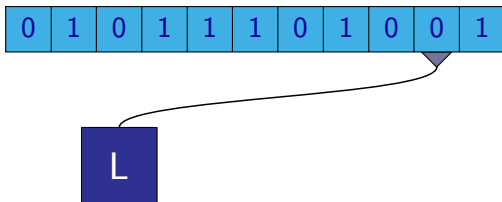
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



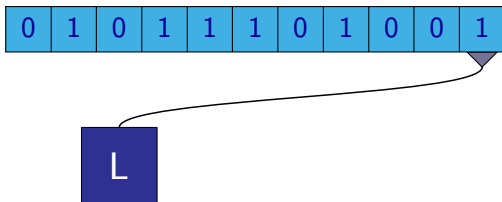
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



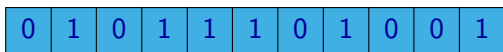
Druhý nápad: Ve skutečnosti nás zajímá pouze, zda počet dosud přečtených symbolů **1** je sudý nebo lichý (místo čísla si stačí pamatovat jen jeho poslední bit).



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



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



S ANO

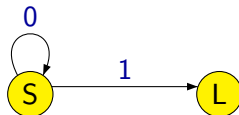
Chování tohoto zařízení můžeme popsat grafem:



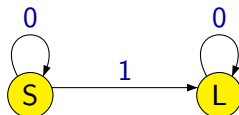
Chování tohoto zařízení můžeme popsat grafem:



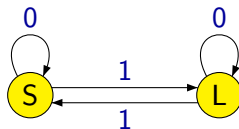
Chování tohoto zařízení můžeme popsat grafem:



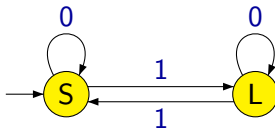
Chování tohoto zařízení můžeme popsat grafem:



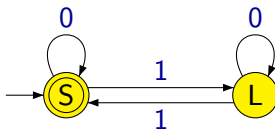
Chování tohoto zařízení můžeme popsat grafem:



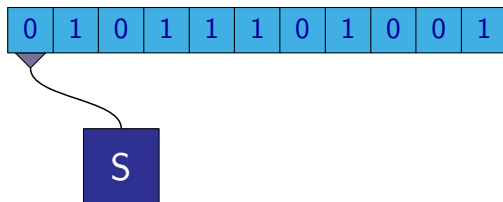
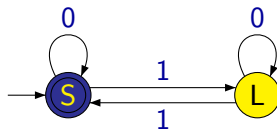
Chování tohoto zařízení můžeme popsat grafem:



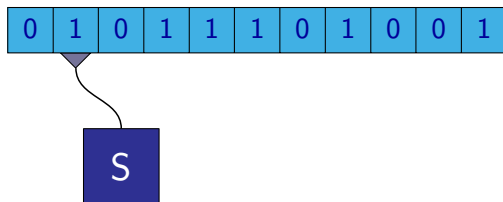
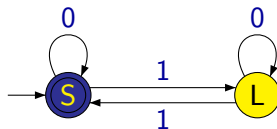
Chování tohoto zařízení můžeme popsat grafem:



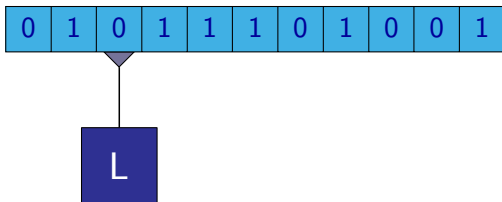
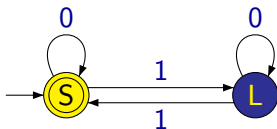
Chování tohoto zařízení můžeme popsat grafem:



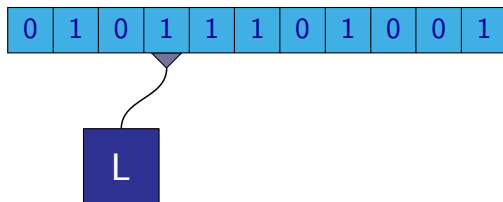
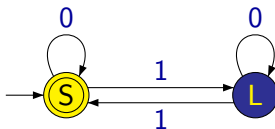
Chování tohoto zařízení můžeme popsat grafem:



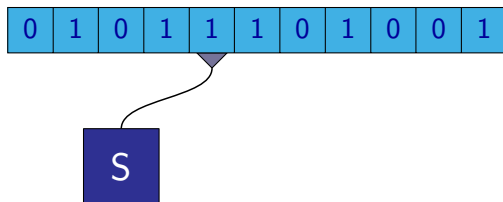
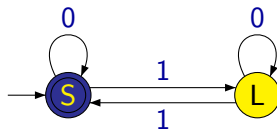
Chování tohoto zařízení můžeme popsat grafem:



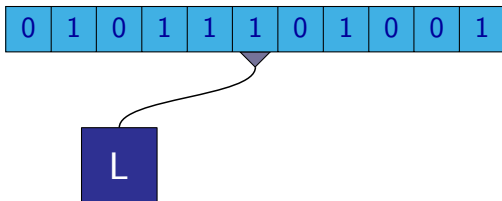
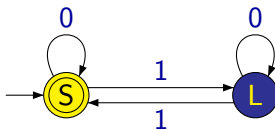
Chování tohoto zařízení můžeme popsat grafem:



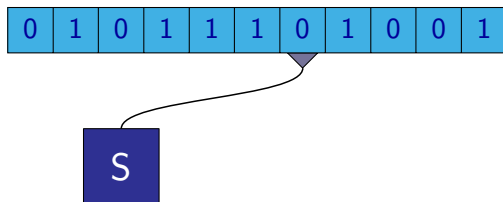
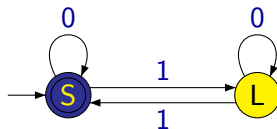
Chování tohoto zařízení můžeme popsat grafem:



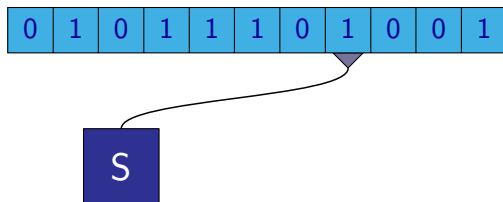
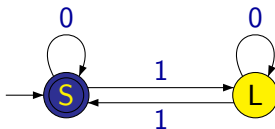
Chování tohoto zařízení můžeme popsat grafem:



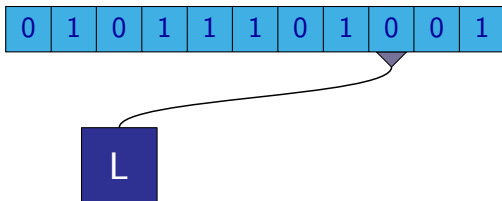
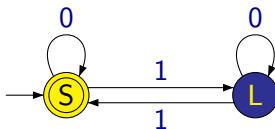
Chování tohoto zařízení můžeme popsat grafem:



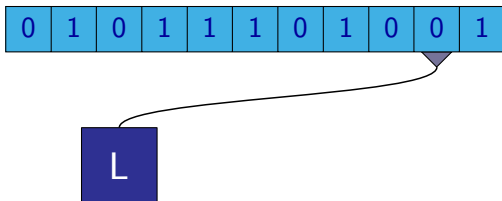
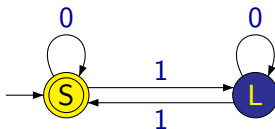
Chování tohoto zařízení můžeme popsat grafem:



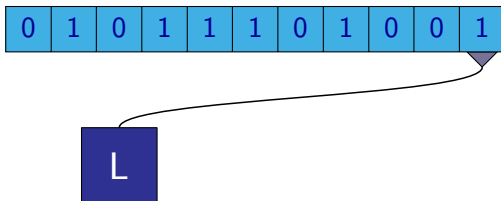
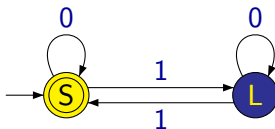
Chování tohoto zařízení můžeme popsat grafem:



Chování tohoto zařízení můžeme popsat grafem:



Chování tohoto zařízení můžeme popsat grafem:



Chování tohoto zařízení můžeme popsat grafem:

