# Ročníkový projekt: Knihovna pro testování lehkých restartujících automatů a webové prostředí

Jan Urban

October 2019

Tato práce pojednává o tvorbě knihovny pro práci s restartujícími automaty

### 0.1 Příklad použití restartujících automatů

Příklad od lingvistů, kde restartující automat mohou použít na redukční analýzu věty:

```
"Trosky zaměstnance, který tam plnil své úkoly, zcela zavalily"
"Trosky zaměstnance, který plnil své úkoly, zcela zavalily"
"Trosky zaměstnance, který plnil úkoly, zcela zavalily"
"Trosky zaměstnance zcela zavalily"
"Trosky zaměstnance zavalily"
```

Což je korektní věta a z toho vyplývá, že počáteční věta je syntakticky správná. Na rozdíl od "Trosky zavalily" která je věta neúplná. Stejně nekorektní (neúplná je následující větev úprav)

```
"Trosky zaměstnance zcela zavalily"
"Trosky zcela zavalily"
```

Z čehož je už možné zjistit že věta je špatně, nebo po další redukci na "Trosky zavalily"

## 1 Třídy restartujících automatů

### 1.1 RRWW

Restartující automat je n-tice $M = (Q, \Sigma, \Gamma, \#, \$, k, I, q_0, Q_A, Q_R)$.

- $Q$ je konečná množina stavů
- $\Sigma$ *a* $\Gamma$ jsou konečné množiny symbolů, vstupní abeceda a pracovní abeceda v tomto pořadí, pro které platí že sigma subset of gamma.

- $k$ je nezáporné číslo udávající velikost dopředného okna, kolik znaků před hlavu automatu vidíme. (neboli velikost celého okna je k+1)
- $I$ je konečná množina pravidel následujících tří typů:
  1. $(q, au) \rightarrow (q', MVR)$ nebo $(q, au) \rightarrow (q', MVL)$
  2. $(q, au) \rightarrow (q', Rewrite(v))$
  3. $(q, au) \rightarrow Restart$
- $q_0$ je počáteční stav a platí $q_0 \in Q$
- $Q_A$ jsou akceptující stavy $Q_A \subseteq Q$
- $Q_R$ jsou odmítající stavy $Q_R \subseteq Q$
- První symbol je normálně ¢(v mém případě volím # - dá se lépe najít na klávesnici) a posledním symbolem je $ u dané pásky/slova

$Q$ je rozdělena do dvou množin a to do: 1. nekoncové stavy $Q - (Q_A \cup Q_R)$ které mají alespoň jedno pravidlo pro každý vstup do kterého se mohou dostat 2. koncové stavy $Q_A \cup Q_R$ které ukončují celou práci automatu

### 1.1.1 Něco více o pravidlech

1. $MVR$ - M změní stav na $q'$ a posune hlavu doprava na další symbol (symbol nemusí být nutně jedno písmeno nebo znak, ale klidně nějaká n-tice znaků) 2. $(q, au) \rightarrow (q', Rewrite(v))$ - přepíše $au$ na $v$, kde $v$ je kratší než $au$. To se docílí smazáním alespoň jednoho symbolu a možným nahrazením dalších symbolů z $au$ 3. *Restartuj* změní stav na počáteční a přesune hlavu na začátek pásky, tak že první znak v okně bude levá zarážka.

## 1.2 RRW

Vychází z RRWW-automatu ale má totožnou pracovní abecedu s vstupní abecedou

## 1.3 RR

Vychází z RRW-automatu, ale zakazuje přepisovat, tedy může jen mazat

## 1.4 R

Vychází z RR automatu, ale restartuje se ihned po přepsání (smazání)

## 1.5 RWW

Vychází z RRWW automatu, ale po přepsání se ihned restartuje

# 2 Jaké typy automatů program rozpozná

## 2.1 Determinismus

projde všechny instrukce a pokud má každá maximálně jednu pravou stranu tak je deterministický (nebo má jen hvězdičku která odpovídá všem možnostem a ta má jen jednu pravou stranu)

## 2.2 možno doplnit monotonii, a rozpoznávání tříd restartujících automatů

# 3 Webové prostředí

- Možnost zadání automatu buď v textové formě nebo nahráním souboru. - Vypsat přijímané slova do nějaké velikosti zadané uživatelem. - Otestovat zda dané slovo je přijímáno a jak bylo přijato (přes jaké stavy ...) - nějaká možnost si zapsat program instrukcemi podporované pythonem a mojí knihovničkou - nejspíše nějakou webovou konzolí - ptát se co to je za automat - vybírat možnosti výstupu (vypsat celou cestu graficky, vypsat jen stavy)

# 4 Jak funguje tento program / automat

Tento automat je nedeterministický z důvodů že musí umět přijímat i nedeterministické automaty a je zásobníkového typu. Tedy rozhoduje se zda přijme, nebo nepřijme dané slovo na základě stavu automatu po vyprázdnění zásobníku.

## 4.1 Instrukce automatu

Tento automat implementuje několik instrukcí:

1. MVR (move right, posun doprava)
2. MVL (move left, posun doleva)
3. Restart
4. Accept (přijmi, vlastně restart)
5. "[ 'x' ]" (přepiš svojí pozici na "x")

### 4.1.1 Jak přidat pravidla/instrukce

Listing 1: Příklad použití

```
from automata import automaton
a = automaton()

a.mess["size_of_window"] = 3 # velikost okna bude 3

# počáteční stav a počáteční pozice okna
a.mess["s0"] = ["q0", 0]
```

```
# přidání abecedy
a.add_to_alphabet("#", "$", "a", "b", "c", "d")

# přidání akceptujícího stavu (v našem případě "Accept")
a.add_accepting_state("Accept")

# přidání pravidla (q0, #ab) -> (q0, MVR)
# /neboli ze stavu q0 a po přečtení "#ab" přejde
# automat do stavu q0 a posune se o jednu pozici doprava
# strtolist=True udává zda se přečtené data mají převést
# do listu tedy každý znak odpovídá jednomu
# symbolu abecedy -> ["#","a","b"]
a.add_instruction("q0", "#ab", "q0", "MVR", strtolist=True)

# nějaký způsob jak to vyplňovat v loopu
for i in ["aaa", "aab", "abb", "abc", "bbb", "bbc", "bbd"]:
  a.add_instruction("q0", i, "q0", "MVR", strtolist=True)

# a spustíme automat a na text "#aaabbbc$" a vrátí True a vypíše cestu s bare
a.iterate_text("#aaabbbc$")
# příklad z Example0.py
```