Ročníkový projekt: Knihovna pro testování restartujících automatů a webové prostředí

Jan Urban

October 2019

Obsah

Úvod

Tato práce pojednává o tvorbě knihovny pro práci s restartujícími automaty. A vizualizaci restartujících automatů ve webovém prostředí. V první kapitole zadefinuji třídy restartujících automatů a vlastnosti, které podporuje tato knihovna. Druhá kapitola se zaměřuje na webové rozhraní. Ve třetí je podrobněji rozepsáno jak s knihovnou pracovat. A ve čtvrté je jak knihovna funguje.

0.1 Redukční analýza

Redukční analýza je metoda, která pomocí odstraňování a přesouvání slov ve větě, redukuje větu na kratší větu a zachová správnost věty.

0.2 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, tedy 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 Jaké typy automatů program rozpozná

Zde jsou zadefinované třídy restartujících automatů a vypsané vlastnosti, které má knihovna rozpoznávat.

Definice 1 (RLWW) *Nechť M je RLWW automat pak platí $M = (Q, \Sigma, \Gamma, \#, \$, k, I, q_0, Q_A)$.*

- *Q je konečná množina stavů.*
- *Σ a Γ jsou konečné množiny symbolů, vstupní abeceda a pracovní abeceda v tomto pořadí, pro které platí že $\Sigma \subseteq \Gamma$.*
- *k je nezáporné číslo udávající velikost okna.*
- *I je konečná množina pravidel následujících čtyř typů, pro které platí $u, v \in \Sigma^*$ a $q, q' \in Q$:*
  1. *$(q, u) \to (q',\ MVR)$ (posun doprava)*
  2. *$(q, au) \to (q', MVL)$ (posun doleva)*
  3. *$(q, u) \to (q', v)$ - neboli Rewrite(v)*
  4. *$(q, u) \to (q_0, Restart)$*
- *q_0 je počáteční stav a platí $q_0 \in Q$.*
- *Q_A jsou akceptující stavy $Q_A \subseteq Q$.*

- *Jako první symbol pásky (levá zarážka) se v literatuře používá znak ¢. Cent se běžně na dnešní klávesnici nevyskytuje, tak v knihovně volím pro levou zarážku znak #. Který se dá lépe najít na klávesnici. A poslední znak pásky(pravá zarážka) je $.*

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

1.0.1 Něco více o pravidlech

1. MVR/MVL – automat M změní stav na q' a posune okno doprava na políčko pásky.
2. $(q, u) \to (q', v)$ – přepíše u na v, kde v musí být kratší než u. To se docílí smazáním alespoň jednoho symbolu a nahrazením dalších symbolů z u.
3. *Restart* změní stav na počáteční a přesune hlavu na začátek pásky, tak že první znak v okně bude levá zarážka (#).

1.1 RRWW

Vychází z RLWW-automatu ale nepoužívá MVL – posun doleva

1.2 RRW (RLW)

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

1.3 RR (RL)

Vychází z RRW (RLW)-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

1.6 Determinismus

Pokud má automat pro každou dvojici (q, u), kde $q \in Q$, $u \in \Sigma^*$, nejvýše jedno pravidlo s levou stranou (q, u), tak říkáme, že automat je deterministický.

1.7 možno doplnit

monotonii

2 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)

3 Jak funguje tato knihovna

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.

3.1 Instrukce automatu

Tento automat implementuje několik instrukcí:

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

3.2 Jak inicializovat automat

Inicializovat automat můžeme dvěma způsoby, ze souboru a nebo vytvořením nového automatu. Soubor do kterého se soubor ukládá je typu json, tedy nějaký slovník klíčů a nějakých dat.

Listing 1: Inicializace automatu

```
from automata import automaton
a = automaton()
a.definition["size_of_window"] = 3
a.definition["s0"] = ["q0", 0]
a.add_to_alphabet("#", "$", "a", "b", "c", "d")
a.add_accepting_state("Accept")
```

3.3 Jak přidat jednotlivé pravidla

Pokud chceme přidat nějaké pravidlo, například ze stavu q_0 a s viditelnou páskou odpovídající #ab chceme zůstat ve stavu q_0 a posunout se doprava tedy: $(q_0, \#ab) -> (q_0, \text{MVR})$. Stačí napsat:

```
a.add_instr("q0", "#ab", "q0", "MVR")
```

Kvůli tomu že dovoluji, že znak abecedy může být i nějaká n-tice symbolů, tedy klidně i "VíceSymbolů"braný jako znak. Pokud by jsme chtěli použít nějaký nepísmenný znak u automatu s velikostí okna 3, tak použijeme tento zápis:

```
a.add_instr("q0", "[ '#', 'Tohle je jeden znak', 'b']",
  "q0", "MVR", value_as_list=True)
```

Můžeme zapsat pravidlo i v regulárních výrazech (Regular Expressions). Neboli * pro cokoliv, ? pro jeden znak ... (ještě není implementováno, bude doplněno) Jen je potřeba psát to "listovým"zápisem (viz výše).

```
a.add_instr("q0", "[ '#', '*', '?a*']",
  "q0", "MVR", value_as_list=True)
```

3.4 Otestování pásky, zda bude přijata automatem

Otestování na výstup buď zamítne (napsáním False), nebo přijme (True) a když dané slovo přijme, tak vypíše na konzoli jak automat postupoval. Páska by měla být ohraničená zarážkami # a $. Zapsat můžeme tímto způsobem:

```
a.iterate_text("#aaabbbc$")
```

Více symbolové znaky se na pásku dají zapsat pomocí ohraničení do hranatých závorek. Berou se jen vnější závorky a vše co je uvnitř se chápe jako jeden znak. Ale pokud se uvnitř použijí hranaté závorky musí být vždy v páru.

```
a.iterate_text("#aa[Znak]bbc$")
a.iterate_text("#aa[[a]cb]bbc$")
```

4 Jak funguje knihovna

V této sekci bych probral z jakých částí je složena tato knihovnička.

1. Konfiguraci - v jakém je stavu, co vidí, jakou má pásku, což je důležité u nedeterministických možností.
2. Iterace textu - neboli pásky, hlavní část programu.
3. Pravidla - jak se vyhodnocují pravidla a nedeterminismus programu.

4.1 Konfigurace

Konfigurace je samostatná třída obsahující stav automatu, pozici na pásce, verzi textu a svého předchůdce pro vykreslování cesty. Což je vše co si daná verze běžícího automatu potřebuje pamatovat. Tento způsob není moc efektivní, ale na malé příklady postačující.

```
class configuration:
  def __init__(self, state: str, position: int,
   text_version: int, father=None)
```

4.2 Iterace textu

Zkusí dosadit do s nový status, pokud se je zásobník (configs) prázdný tak se vyhodí výjimku a konfigurace "conf"zůstane nezměněná. Poté přečte okno ze správné verze pásky. A zavolá funkci __move s oknem a konfigurací "conf". Pokud poslední stav byl akceptující tak se vrátí true a vytiskne se postup automatu.

```
while True:
  try:
    conf = self.configs.pop()
    if conf.state == "Accept":
      raise Exception("Accepting_state")
    window = self.__get_window(
        self.texts[conf.text_version], conf.position)
    self.__move(window, conf)
  except:
    if self.is_accepting_state(conf.state):
      self.pretty_printer(conf)
      return True
    elif self.configs.__len__() == 0:
      return False
```

4.2.1 funkce vyhodnocování (__move()) / nedetrministická část

Funkce se podívá do slovníku s pravidly a zkusí použít všechny pravidla které může. K tomu použije další funkci __make_instruction(). Jelikož vybírá všechny možnosti, které se ve slovníku nacházejí s tímto stavem a obsahem okna, se stává automat nedeterministický.

```
def __move(self, window, conf):
  possibilities = self.definition["tr_function"][conf.state]
  if "['*']" in possibilities:
    for possibility in possibilities["['*']"]:
      self.__make_instruction(possibility[1], possibility[0], conf)
  for possibility in possibilities[window]:
    self.__make_instruction(possibility[1], possibility[0], conf)
```

4.3 Pravidla

Dopsat....

```
pos = stat.position
end_of_pos = self.size_of_window + pos
```

4.3.1 MVR / MVL

```
if instruction == "MVR":  # move right
  conf = configuration(new_state, pos + 1, stat.text_version, stat)
  self.configs.append(conf)
elif instruction == "MVL":  # move right
  conf = configuration(new_state, pos - 1, stat.text_version, stat)
  self.configs.append(conf)
```

4.3.2 Restart

```
elif instruction == "Restart":  # restart
  conf = configuration(new_state, 0, stat.text_version, stat)
  self.configs.append(conf)
```

4.3.3 Accept

```
elif instruction == "Accept":
  conf = configuration(new_state, 0, stat.text_version, stat)
  self.configs.append(conf)
```

4.3.4 Rewrite

Není moc bezpečné kvůli eval, chtělo by to něco lepšího.

```
  elif re.match(r"^\[.*\]$", instruction):
  new_list = self.texts[stat.text_version].copy()
  new_values = eval(instruction)
  new_list[pos: end_of_pos] = new_values

  self.texts.append(new_list)
  conf = configuration(new_state, stat.position, len(self.texts) - 1, stat)
  self.configs.append(conf)
```