
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**
**DEPARTMENT OF INFORMATION SYSTEMS**


# ANALÝZA A TRANSFORMACE KÓDŮ ZALOŽENÁ NA PŘEVODNÍCÍCH

**CODE ANALYSIS AND TRANSFORMATION BASED ON TRANSDUCERS**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **MARIÁN PUKANČÍK**
**AUTHOR**

**VEDOUCÍ PRÁCE** **prof. RNDr. ALEXANDER MEDUNA, CSc.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

# Zadání bakalářské práce


Student: **Pukančík Marián**
Program: Informační technologie
Název: **Analýza a transformace kódů založená na převodnících**
**Code Analysis and Transformation Based on Transducers**
Kategorie: Překladače

Zadání:

1. Seznamte se s metodami analýzy a transformace kódů, které jsou prováděny převodníky.
2. Dle instrukcí vedoucího navrhněte vlastní metody, které představují modifikované verze metod z předchozího bodu.
3. Dle instrukcí vedoucího studujte vlastnosti vlastních metod navržených v předchozím bodě. Porovnejte jejich vlastnosti s vlastnostmi jiných metod.
4. Dle instrukcí vedoucího aplikujte metody navržené v bodě 2 ve vhodné oblasti informatiky, např v kryptografii či bioinformatice. Implementujte a testujte ji.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Meduna, A.: Automata and Languages, Springer, London, 2000

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

# Abstrakt

Táto práca sa zaoberá analýzou a transformáciou kódu založenej na prevodníkoch. Práca definuje všetky potrebné teoretické pojmy týkajúce sa konečných a zásobníkových prevodníkov, a následne sa venuje návrhu a implementácii aplikácie založenej na týchto druhoch prevodníkov, so zameraním na oblasť bioinformatiky.



# Abstract

This thesis is concerning with Code Analysis and Transformation Based on Transducers. Thesis defines all necessary terms regarding finite and pushdown transducers and then discusses design and implementation of an application based on these types of transducers, oriented on the area of bioinformatics.



# Kľúčové slová

konečný prevodník, zásobníkový prevodník, rozšírený zásobníkový prevodník, konfigurácia, výpočtový krok, aminokyselina, kodón, transkripcia, translácia, DNA, RNA

# Keywords

finite transducer, pushdown transducer, extended pushdown transducer, configuration, step, amino acid, codon, transcription, translation, DNA, RNA


# Citácia

PUKANČÍK, Marián. *Analýza a transformace kódů založená na převodnících.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. RNDr. Alexander Meduna, CSc.

# Analýza a transformace kódů založená na převodnících

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána prof. RNDr. Alexandra Medunu CSc. Ďalšie informácie mi boli poskytnuté pani Ing. Ivanou Burgetovou Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Marián Pukančík
27. mája 2020


## Poďakovanie

Rád by som sa poďakoval pánovi prof. RNDr. Alexandrovi Medunovi CSc. za jeho odbornú pomoc pri tvorbe tejto práce. Tiež by som sa chcel poďakovať pani Ing. Ivane Burgetovej Ph.D. za jej cenné rady v oblasti molekulárnej biológie, a taktiež celej mojej rodine, ktorá za mnou po celú dobu stála.

# Obsah

# Kapitola 1

# Úvod

Táto práca je venovaná štúdiu vlastností prevodníkov a ich využití pri analýze a transformácii kódu v oblasti bioinformatiky.

Mojím cieľom bolo využiť získané znalosti o konečných a zásobníkových automatoch nadobudnutých v predmete Formálne jazyky a prekladače pri štúdiu prevodníkov a následne vytvoriť aplikáciu, ktorá by bola schopná demonštrovať silu prevodníkov ako výpočtového modelu. A ako som už uviedol vyššie, rozhodol som sa aplikovať tieto znalosti v oblasti bioinformatiky, konkrétne som sa zameral na proces proteosyntézy u prokaryotických buniek.

Zameranie na oblasť bioinformatiky som zvolil z dôvodu, že je to v súčasnosti veľmi rýchlo sa rozvíjajúca oblasť, a zároveň pracuje s dátami (sekvencie DNA/mRNA), ktorých spracovanie je možné automatizovať aj aplikáciami založenými práve na prevodníkoch, ktoré sú zameraním tejto práce.

Výsledkom je aplikácia schopná prijať reťazec DNA (prípadne mRNA) a následne symulovať procesy transkripcie a translácie, ktorých produktom je proteín alebo skupina proteínov reprezentovaných ako reťazec (alebo reťazce) aminokyselín. Aplikácia prijíma tieto sekvencie dvomi spôsobmi: zadaním priamo na vstupe alebo ako obsah priloženého súboru, ktorý by mal spĺňať štandardy formátu FASTA – najčastejšie používaného formátu v oblasti bioinformatiky a biochémie. (Avšak je možné zadať aj iné formáty súborov.) Zároveň aplikácia poskytuje možnosť vygenerovania sekvencie DNA a jej následné spracovanie.

V kapitole 2 sa nachádzajú definície základných pojmov potrebných pre definovanie prevodníkov, odvodzovanie pravidiel, na základe ktorých tieto výpočtové modely pracujú, a taktiež pre základné uvedenie do problematiky v oblasti množín a formálnych jazykov.

Nasledujúce kapitoly sú venované skúmaným druhom prevodníkov (3) a základom molekulárnej biológie (4). Zatiaľ čo kapitola 3 definuje jednotlivé druhy prevodníkov, ich vlastnosti (napr.: konfigurácia, výpočtový krok...) a princíp ich práce, tak v kapitole 4 sú prezentované základné pojmy z oblasti molekulárnej biológie, ktoré sú nevyhnutné pre pochopenie procesov, ktoré výsledná aplikácia vykonáva. Rovnako tu sú prezentované ako aj definície, tak aj ilustračné príklady procesov *transkripcie* a *translácie* genetickej informácie.

Kapitola 5 predstavuje konceptuálny návrh už vyššie spomínanej aplikácie, detailne popisuje implementáciu jej jednotlivých častí a následne sa venuje testovaniu funkcionality a validácii získaných dát.

Táto aplikácia implementuje tri prevodníky. Dva konečné, z ktorých prvý slúži ako *scanner*, čiže je schopný určiť druh a správnosť vstupných sekvencií, a druhý, ktorý priamo implementuje proces *transkripcie* vstupných sekvencií (mení *tymín* na *uracil*). Tretí prevodník je zásobníkový, ktorého zásobník je na svojom vrchole schopný pracovať s reťazcom dĺžky dva symboly, a teda priamo implementuje proces *translácie*. Následne sú prezentované zís-

kané výsledky, ktoré sú porovnávané s výsledkami získanými z webovej aplikácie Sequence Manipulation Suite[1], ktorá mimo iného je tiež schopná prekladať sekvencie DNA/mRNA na sekvencie aminokyselín.

Poslednou kapitolou je záver 7, v ktorom sú zhrnuté výsledky tejto práce, rovnako ako vyhliadky do budúcna pre ďalší vývoj navrhnutej aplikácie.

[1]Sequence Manipulation Suite – dostupná na https://www.bioinformatics.org/sms2/index.html

# Kapitola 2

# Matematický základ

Táto kapitola definuje základné pojmy, ktoré sú ďalej využité pre definície rôznych typov prevodíkov, množín pravidel týchto prevodníkov alebo demonštráciu príkladov použitia. Definície týchto pojmov sú prevzaté z [10], [9] a [2].

### Množina

Množina je súhrn dobre rozlíšiteľných entít, ktorý chápeme ako celok. Algebrické množiny označujeme veľkými písmenami latinskej abecedy. Entity, ktoré množina obsahuje, sa nazývajú prvky množiny. Označujeme ich malými písmenami latinskej abecedy. Napr.:$A = \{c, d, 5, d, g\}$.

### Rozdelenie množín

1. Konečné – majú konečný počet prvkov.
2. Nekonečné – spočitateľné, nespočitateľné.
3. Prázdne – neobsahuje žiaden prvok. $A = \emptyset$.
4. Neprázdne – obsahujú $1..n$ prvkov. $A \neq \emptyset$.
5. Disjunktné – nemajú spoločné prvky, ich prienik je prázdna množina: $A \cap B = \emptyset$.
6. Nedisjunktné – $A \cap B \neq \emptyset$.

### Množinové operácie

1. Zjednotenie množín $A$ a $B$ je množina všetkých spoločných prvkov z množiny $U$, ktoré patria aspoň do jednej z množín $A$ alebo $B$ ($A \cup B$).
2. Prienik množín $A$ a $B$ je množina všetkých prvkov, ktoré patria do množiny $A$ a zároveň do množiny $B$ ($A \cap B$).
3. Rozdiel množín $A$ a $B$ je množina všetkých prvkov, ktoré patria množine $A$, ale nepatria množine $B$ ($A - B$).
4. Doplnok (komplement) množiny $A$ je množina všetkých prvkov patriacich množine $U$, ktoré nepatria množine $A$, označuje sa $A' = U - A$.

5. Karteziánsky súčin množín $A$, $B$ je množina všetkých usporiadaných dvojíc $[x, y]$, kde $x \in A$ a $y \in B$. Inými slovami: je to množina všetkých usporiadaných dvojíc, ktorých prvá zložka je z prvej množiny (teda z množiny $A$) a druhá zložka z druhej množiny (teda z množiny $B$).

### Abecedy a symboly

Abeceda je konečná, neprázdna množina elementov, ktoré nazývame symboly. Ak označíme abecedu $\Sigma$, potom $\Sigma = \{u, v, 1, 2\}$ je abeceda obsahujúca štyri symboly.

### Reťazec

Nech $\Sigma$ je abeceda.

1. $\epsilon$ je reťazec na abecedou $\Sigma$.
2. Ak $x$ je reťazec nad $\Sigma$ a $z \in \Sigma$, potom $xz$ je reťazec nad abecedou $\Sigma$.

### Dĺžka reťazca

Nech $x$ je reťazec nad abecedou $\Sigma$. Dĺžka reťazca $x$, $\mid x \mid$, je definovaná:

1. Ak $x = \epsilon$, potom $\mid x \mid = 0$.
2. Ak $x = a_1..a_n$, potom $\mid x \mid = n$, pre $n \geq 1$ a $a_i \in \Sigma$ pre všetky $i = 1, .., n$.

### Konkatenácia reťazcov

Nech $x$ a $y$ sú dva reťazce nad abecedou $\Sigma$. Konkatenácia $x$ a $y$ je reťazec $xy$.

### Podreťazec

Nech $x$ a $y$ sú dva reťazce nad abecedou $\Sigma$. $x$ je podraťazec $y$, ak existujú reťazce $z_1, z_2$ nad abecedou $\Sigma$, pričom platí: $z_1 x z_2 = y$.

### Jazyk

Nech $\Sigma^*$ značí množinu všetkých reťazcov nad $\Sigma$. Každá podmnožina $L \subseteq \Sigma^*$ je jazyk nad $\Sigma$.

# Kapitola 3

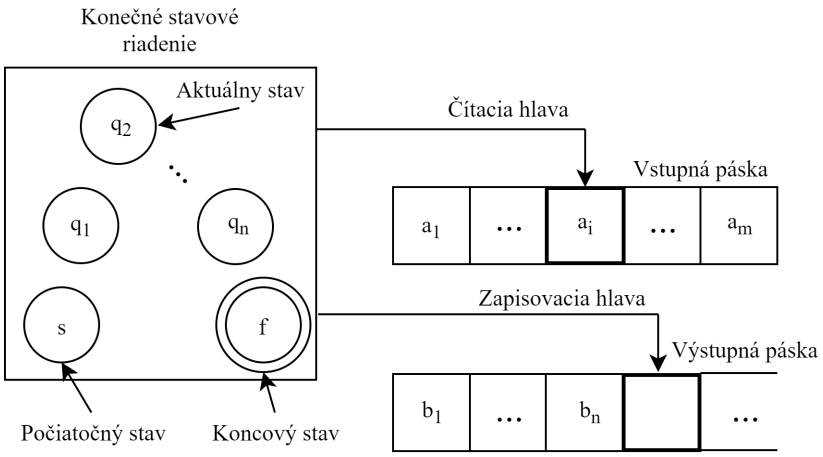
# Prevodníky

Táto kapitola bola prevzatá z [10], [4], [9] a [1] a definuje jednotlivé druhy prevodníkov, ich vlastnosti (napr.: konfigurácia, výpočtový krok...) a princíp ich práce.

## 3.1 Konečný prevodník

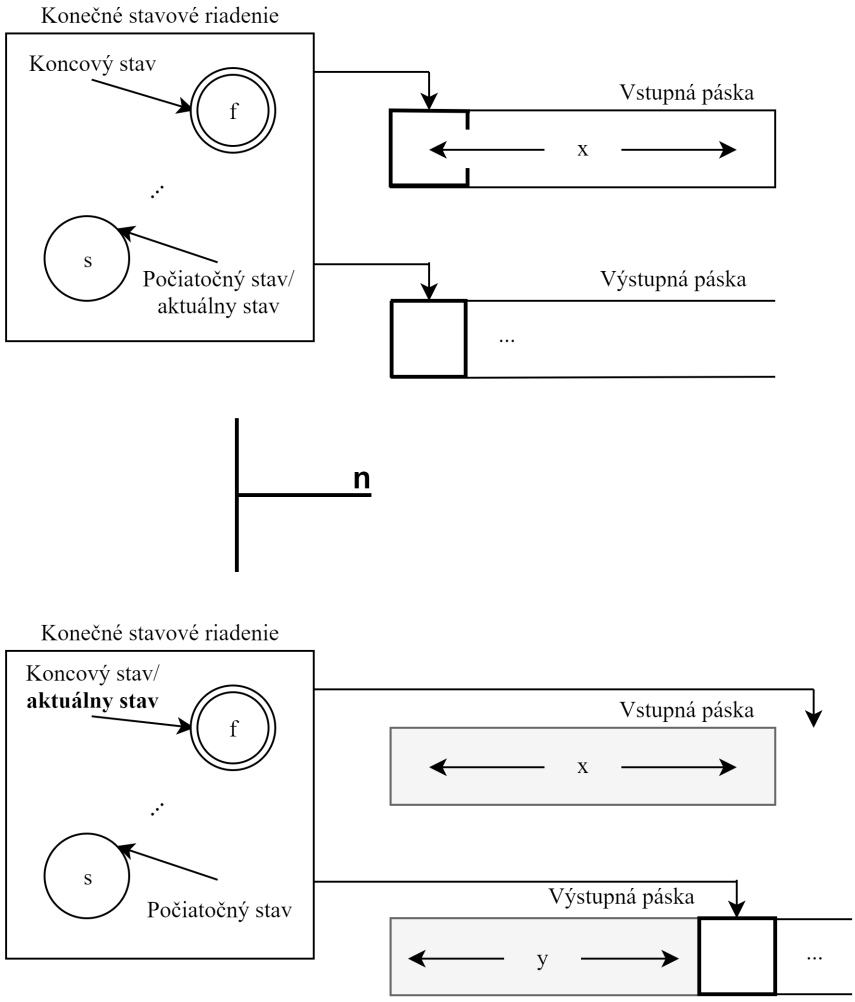
Konečný prevodník je výpočtový model vychádzajúci z konečného automatu. Avšak oproti konečnému automatu, disponujúcemu len jednou (vstupnou) páskou, disponuje dvomi páskami – vstupnou páskou, určenou len pre čítanie a páskou výstupnou, určenou len pre výpis na výstup. Konečný prevodník $M$ sa teda skladá zo vstupnej/výstupnej pásky, čítacej/zapisovacej hlavy a konečného stavového riadenia.

Obidve pásky sú rozdelené na štvorce. Každý štvorec vstupnej pásky obsahuje jeden symbol vstupného slova, $a_1...a_i...a_m$. Symbol $a_i$ reprezentuje aktuálne prečítaný symbol. Výstupná páska je nekonečná smerom vpravo. Ak výstupná páska obsahuje slovo $b_1...b_n$, tak potom sa zapisovacia hlava nachádza jeden štvorec za štvorcom obsahujúci $b_n$. Konečné stavové riadenie je reprezentované ako konečná množina stavov a konečnou reláciou reprezentovanou ako množina výpočtových pravidiel. Graficky je konečný prevodník a jeho jednotlivé časti znázornený na obrázku 3.1.




Obr. 3.1: Konečný prevodník

Konečný prevodník, $M$ pracuje vykonávaním výpočtových krokov podľa definovaných výpočtových pravidiel. V priebehu výpočtového kroku, $M$ zmení aktuálny stav, prečíta nula alebo jeden vstupný symbol a vypíše výstupné slovo. Ak $M$ prečíta vstupný symbol $a_i$, tak následne posunie svoju čítaciu hlavu o jeden štvorec vpravo. Avšak, ak $M$ neprečíta žiaden symbol, tak čítacia hlava zostáva statická. V prípade zapisovacej hlavy sa hlava posunie o jeden štvorec za výstupné slovo. Jeden zo stavov konečného prevodníka $M$ je defnovaný ako pačiatočný a niektoré stavy ako stavy koncové. Nech vstupná páska prevodníka $M$ obsahuje slovo $x$ a výstupná páska je prázdna. $M$ začne preklad slova $x$ z pačiatočného stavu. Ak dokáže $M$ vykonať sekvenciu výpočtových krokov takých, že prečíta celé $x$, vypíše výstupné slovo $y$ a vstúpi do koncového stavu, tak potom $M$ prekladá $x$ do $y$. Množina, ktorá obsahuje všetky páry slov preložených týmto spôsobom, tvorí preklad definovaný $M$. [10]




Obr. 3.2: Práca konečného prevodníka

### 3.1.1 Definícia – konečný prevodník

Konečný prevodník je pätica $M = (Q, \Sigma, R, s, F)$, kde:

- $Q$ je konečná množina stavov,
- $\Sigma$ je abeceda, pre ktorú platí: $\Sigma \cap Q = \Delta$, a zároveň: $\Sigma = \Sigma_I \cup \Sigma_O$, kde $\Sigma_I$ je vstupná abeceda a $\Sigma_O$ je výstupná abeceda,
- $R \subseteq Q(\Sigma \cup \{\epsilon\}) \times Q\Sigma_I^*$ je konečná relácia,
- $s \in Q$ je počiatočný stav,
- $F \subseteq Q$ je množina konečných stavov.

Uvažujme konečný prevodník $M = (Q, \Sigma, R, s, F)$. Členy $R$ sa nazývajú pravidlá, čiže $R$ obsahuje konečnú množinu definovaných pravidiel. Majme napríklad pravidlo $(pa, qz) \in R$, pričom $p, q \in Q, a \in (\Sigma_I \cup \{\epsilon\})$ a $z \in (\Sigma_O \cup \{\epsilon\})$. Namiesto zápisu $(pa, qz)$, môžeme pravidlo zapísať nasledovne:

$$r : pa \vdash qz$$

kde $pa$ je ľavá strana pravidla $r$ a $qz$ tvorí jeho pravú stranu. [10]

### 3.1.2 Definícia – konfigurácia

Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Konfigurácia $M$ je trojica $(q, x, y)$, kde:

- $q \in Q$ je aktuálny stav,
- $x \in \Sigma_I$ je zvyšná časť vstupného reťazca na vstupnej páske, pričom prvý symbol $x$ z ľava je pod čítacou hlavou,
- $y \in \Sigma_O$ je výstupný reťazec, ktorý bol zapísaný na výstupnú pásku do tohto momentu. [1], [9]

### 3.1.3 Definícia – výpočtový krok

Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Ďalej, nech

$$\chi_1 = (q_1, x, y) \text{ a } \chi_2 = (q_2, w, yz)$$

sú dve konfigurácie konečného prevodníka $M$ a nech $r : q_1 a \vdash q_2 z \in R$. Tak $M$ vykoná výpočtový krok z $\chi_1$ do $\chi_2$ na základe pravidla $r$ ([1], [9]), symbolicky zapísané ako:

$$\chi_1 \vdash \chi_2 \ [r]$$

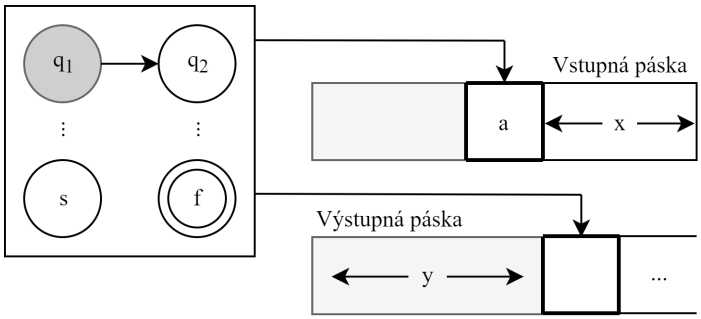
### 3.1.4 Príklad – výpočtový krok

Uvažujme konečný prevodník $M = (Q, \Sigma, R, s, F)$ a konfiguráciu

$$\chi_1 = (q_1, ax, y),$$

kde $q_1 \in Q$, $a \in \Sigma_I$, $x \in \Sigma_I^*$ a $y \in \Sigma_O^*$. Ako obr. 4.3 ukazuje, $(q_1, ax, y)$ reprezentuje aktuálnu konfiguráciu $M$:

1. $q_1$ je aktuálny stav, v ktorom sa $M$ nachádza.
2. $ax$ je doposiaľ neprečítaný suffix vstupného slova.
3. $y$ je výstupné slovo vypísané do tohto momentu na výstupnú pásku. Výstupná hlava ukazuje na prvý štvorec za výstupným slovom.

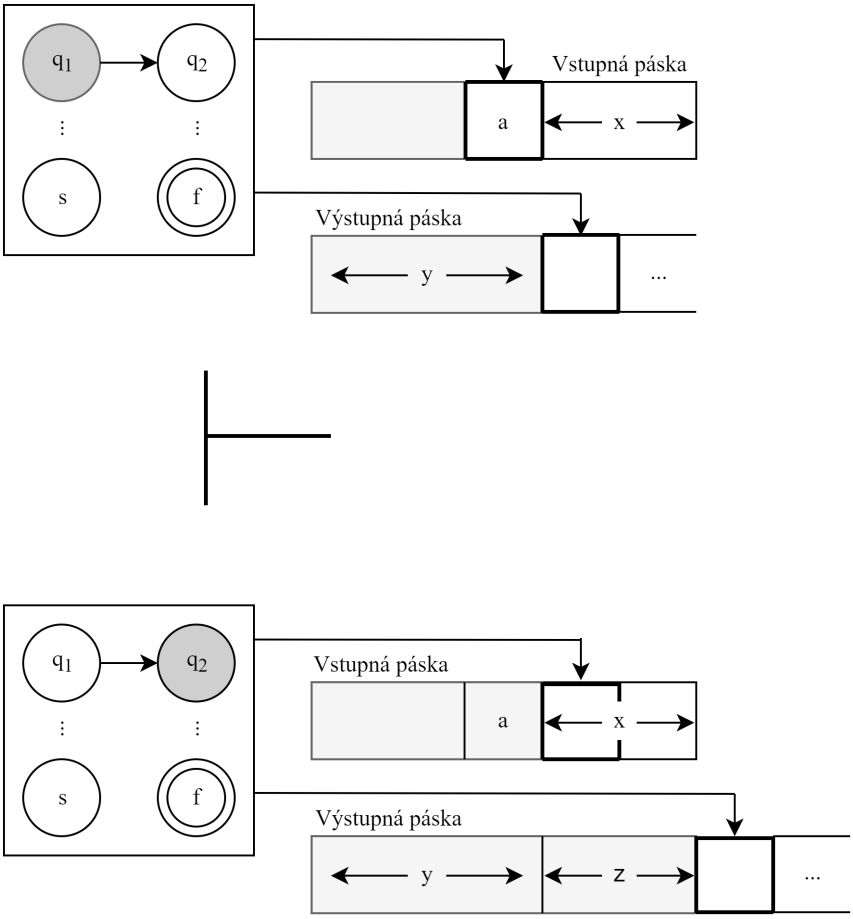



Obr. 3.3: Znázornenie aktuálneho stavu konečného prevodníka

Ďalej uvažujme výpočtový krok zodpovedajúci pravidlu: $r : q_1 a \vdash q_2 z \in R$. Keďže obe $a$ i $z$ sa môžu rovnať $\varepsilon$, tak rozlišujeme tieto nasledovné prípady:

1. $a \in \Sigma_I \wedge z \in \Sigma_O^+$
2. $a = \varepsilon \wedge z \in \Sigma_O^+$
3. $a \in \Sigma_I \wedge z = \varepsilon$
4. $a = \varepsilon \wedge z = \varepsilon$

1. Za predpokladu, že platí $a \in \Sigma_I \wedge z \in \Sigma_O^+$ , tak $M$ zmení stav z $q_1$ na $q_2$, prečíta symbol $a$ zo vstupnej pásky, posunie vstupnú hlavu o jeden štvorec vpravo, zapíše $z$ na výstupnú pásku a nastaví výstupnú hlavu tak, aby ukazovala na prvý štvorec za $z$.
2.Ak $a = \varepsilon \wedge z \in \Sigma_O^+$, tak pravidlo $r$ je možné zapísať nasledovne: $r : q_1 \vdash q_2 z$. Podľa tohto pravidla, $M$ zmení stav z $q_1$ na $q_2$, pričom zapíše $z$ na výstupnú pásku a nastaví výstupnú hlavu tak, aby ukazovala na prvý štvorec za $z$. Avšak vstupná hlava ostáva v pôvodnej polohe.
3.V prípade, že $a \in \Sigma_I \wedge z = \varepsilon$, tak pravidlo $r$ je možné zapísať nasledovne: $r : q_1 a \vdash q_2$. $M$ zmení stav z $q_1$ na $q_2$, posunie vstupnú hlavu o jeden štvorec vpravo, avšak na výstup nie je vypísané nič a výstupná hlava ostáva statická.
4. Ak $a = \varepsilon \wedge z = \varepsilon$, tak $r$ je možné upraviť nasledovne: $r : q_1 \vdash q_2$. V tomto prípade $M$ zmení stav z $q_1$ na $q_2$, avšak nič neprečíta ani nezapíše, a zároveň vstupná a výstupná hlava zotrvajú na svojich miestach.




Obr. 3.4: výpočtový krok

### 3.1.5 Sekvencia krokov

Opäť uvažujme konečný prevodník $M = (Q, \Sigma, R, s, F)$. Nech $\chi$ je akákoľvek konfigurácia $M$. $M$ vykoná nula výpočtových krokov pri prechode z $\chi$ do $\chi$ podľa $\epsilon$:

$$\chi \vdash^0 \chi \ [\epsilon]$$

Ďalej uvažujme sekveciu konfigurácií

$$\chi_0, ..., \chi_n$$

pre $n \geq 1$ takú, pre ktorú platí:

$$\chi_{i-1} \vdash^n \chi_i \ [r_i]$$

kde $r_i \in R, i = 1, ..., n$. Tak potom $M$ vykoná $n$ výpočtových krokov z $\chi_0$ do $\chi_n$ podľa pravidiel $r_1 ... r_n$, čo je možné zapísať ako:

$$\chi_0 \vdash^n \chi_n \ [r_1 ... r_n]$$

Následne uvažujme dve konfigurácie $M$, a to $\chi_1$ a $\chi_2$.
1. Ak existuje $n \geq 1$ také, že platí: $\chi_1 \vdash^n \chi_2[\rho]$, tak môžeme uvažovať, že: $\chi_1 \vdash^+ \chi_2[\rho]$.
2. Ak existuje $n \geq 0$ také, že platí: $\chi_1 \vdash^n \chi_2[\rho]$, tak môžeme uvažovať, že: $\chi_1 \vdash^* \chi_2[\rho]$.
kde $\rho$ reprezentuje sekvenciu $n$ pravidiel z množiny $R$, pričom platí $\rho = \epsilon$ ak $n = 0$. [10]

### 3.1.6 Preklad definovaný konečným prevodníkom

Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník, $x \in \Sigma_I^*$ a $y \in \Sigma_O^*$. $M$ prekladá $x$ na $y$ len ak:

$$(s, x, \epsilon) \vdash^* (f, \epsilon, y)$$

pre $f \in F$. Tak potom preklad definovaný $M$, $T(M)$ je:

$$T(M) = \{(x, y) : x \in \Sigma_I^*, y \in \Sigma_O^*, \text{ a } M \text{ prekladá } x \text{ na } y\}$$

Príslušný vstupný jazyk k $T(M)$ je definovaný ako:

$$L_I(M) = \{x : (x, y) \in T(M) \text{ pre nejaké } y \in \Sigma_O^*\}$$

Príslušný výstupný jazyk k $T(M)$ je definovaný ako:

$$L_O(M) = \{x : (x, y) \in T(M) \text{ pre nejaké } y \in \Sigma_I^*\}$$

Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Potom, ako bolo demonštrované v tejto podsekcii, obidva jazyky $L_I(M)$ aj $L_O(M)$ sú regulárne. [10]

## 3.2 Zásobníkový prevodník

Zásobníkový prevodník je výpočtový model založený na zásobníkovom automate, pričom rozdiel medzi ním a konečným prevodníkom je, že zásobníkový prevodník obsahuje okrem vstupnej a výstupnej pásky ešte aj zásobník.

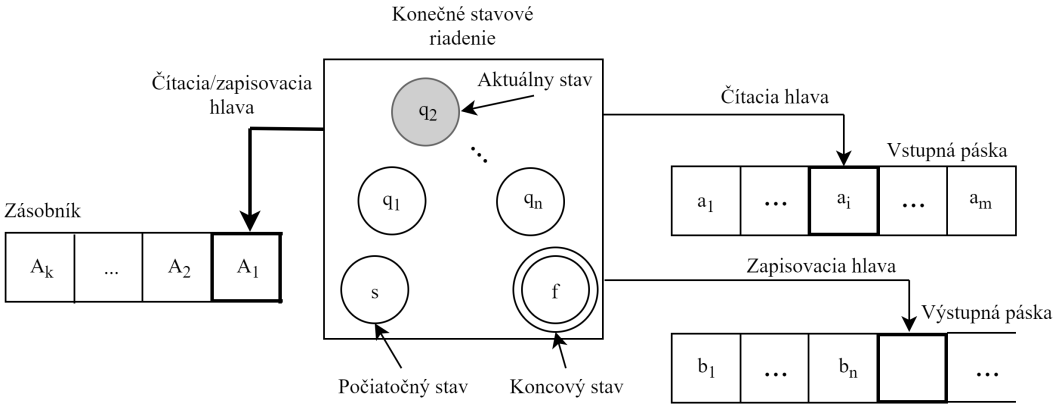
Zásobník má rovnaký význam ako u zásobníkového automatu. Hlavnú úlohu zohráva pri výpočtovom kroku, kedy prevodník prečíta nula alebo jeden symbol zo vstupnej pásky, zmení stav, nahradí vrchný symbol na zásobníku iným symbolom alebo slovom a vypíše výstupné slovo na výstupnú pásku.

Na začiatku prekladu obsahuje vstupná páska slovo $x$, zásobník obsahuje len počiatočný symbol $S$, výstupná páska je úplne prázdna a prevodník sa nachádza v počiatočnom stave. Ak zásobníkový prevodník $M$ dokáže z tejto počiatočnej konfigurácie prečítať celé vstupné slovo $x$, zapísať výstupné slovo $y$ a dosiahnuť koncového stavu, tak potom $M$ prekladá $x$ na $y$. Množina obsahujúca všetky páry slov preložených týmto spôsobom tvorí preklad definovaný $M$. [4], [10]

### 3.2.1 Definícia – zásobníkový prevodník

Zásobníkový prevodník je usporiadaná pätica $M = (Q, \Sigma, R, s, F)$, kde:

- $Q$ je konečná množina stavov;
- $\Sigma$ je abeceda, pre ktorú platí: $\Sigma \cap Q = \emptyset$, a zároveň: $\Sigma = \Sigma_I \cup \Sigma_O \cup \Sigma_{PD}$, kde $\Sigma_I$ je vstupná abeceda, $\Sigma_O$ je výstupná abeceda $\Sigma_{PD}$ je zásobníková abeceda obsahujúca počiatočný symbol $S$;
- $R \subseteq \Sigma_{PD} Q (\Sigma_I \cup \{\epsilon\}) \times \Sigma_{PD}^* Q \Sigma_O^*$ je konečná relácia;
- $s \in Q$ je počiatočný stav;
- $F \subseteq Q$ je množina konečných stavov. [10]




Obr. 3.5: Zásobníkový prevodník

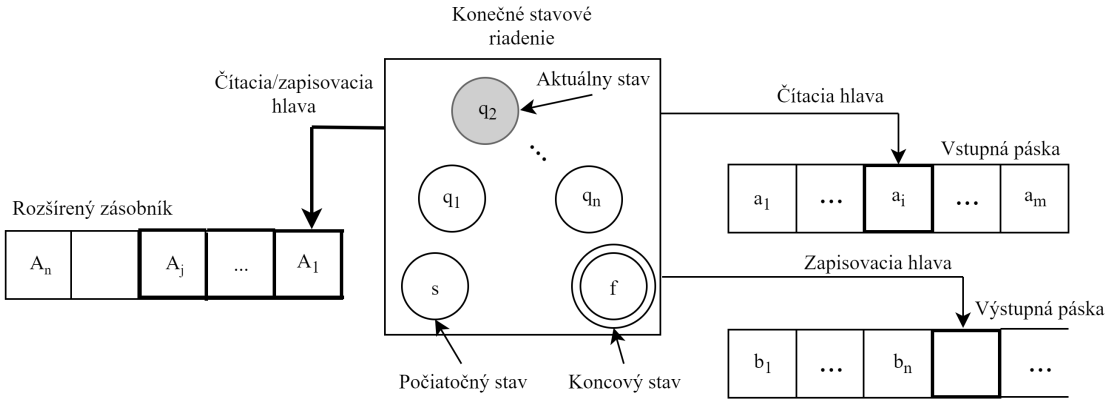
## 3.3 Rozšírený zásobníkový prevodník

Rozšírený zásobníkový prevodník je výpočtový model rozširujúci možnosti zásobníkového prevodníka tým, že umožňuje pracovať s viac ako jedným symbolom na vrchole zásobníka.

### 3.3.1 Definícia – rozšírený zásobníkový prevodník

$$M = (Q, \Sigma, R, s, F)$$

kde
$R \subseteq \Sigma^*_{PD} Q(\Sigma_I \cup \{\epsilon\}) \times \Sigma^*_{PD} Q \Sigma^*_O$ je konečná relácia;
Ostatné časti – $Q, \Sigma, s, F$ majú rovnaký význam ako v prípade zásobníkového prevodníka. [10]

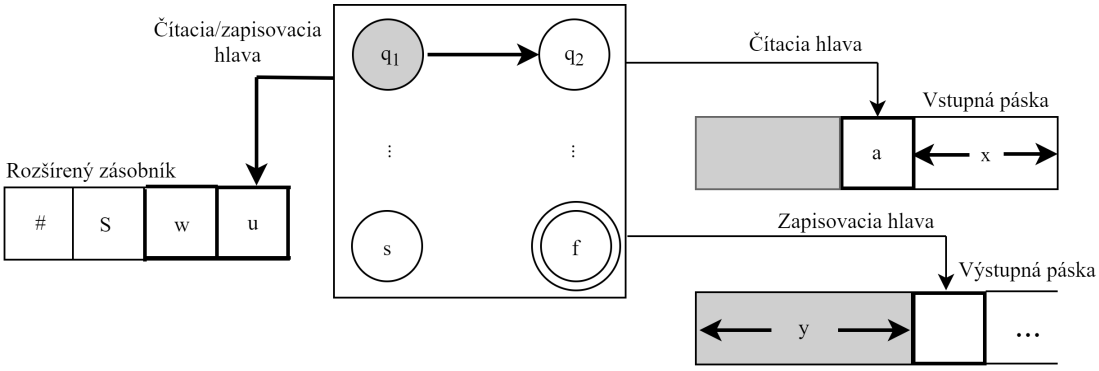



Obr. 3.6: Rozšírený zásobníkový prevodník

### 3.3.2 Definícia – konfigurácia

Nech $M = (Q, \Sigma, R, s, F)$ je rozšírený zásobníkový prevodník. Konfigurácia $M$ je štvorica $(q, x, \alpha, y)$, kde:

- $q \in Q$ je aktuálny stav;
- $x \in \Sigma_I$ je zvyšná časť vstupného reťazca na vstupnej páske, pričom prvý symbol $x$ z ľava je pod čítacou hlavou;
- $\alpha \in \Sigma_{PD}^*$ je n-tica symbolov tvoriacich reťazec na vrchole zásobníka, pričom pre každý symbol $n$ tohto reťazca platí, že $n \in \Sigma_{PD}$;
- $y \in \Sigma_O$ je výstupný reťazec, ktorý bol zapísaný na výstupnú pásku do tohto momentu. [1], [9]




Obr. 3.7: Konfigurácia rozšíreného zásobníkového prevodníka

### 3.3.3 Definícia – výpočtový krok

Nech $M = (Q, \Sigma, R, s, F)$ je rozšírený zásobníkový prevodník. Ďalej, nech

$$\chi_1 = (q_1, x, \alpha_1, y) \text{ a } \chi_2 = (q_2, w, \alpha_2, yz)$$

sú dve konfigurácie $M$ a nech $r : uAq_1a \vdash vBq_2z \in R$, kde
$q_1, q_2 \in Q, a \in \Sigma_I, z \in \Sigma_O, u, v \in \Sigma_{PD}$, $A, B \in \Sigma_{PD}^*$. Tak $M$ vykoná výpočtový krok z $\chi_1$ do $\chi_2$ na základe pravidla $r$ ([1], [9]), symbolicky zapísané ako:

$$\chi_1 \vdash \chi_2 \ [r]$$

### 3.3.4 Príklad – výpočtový krok

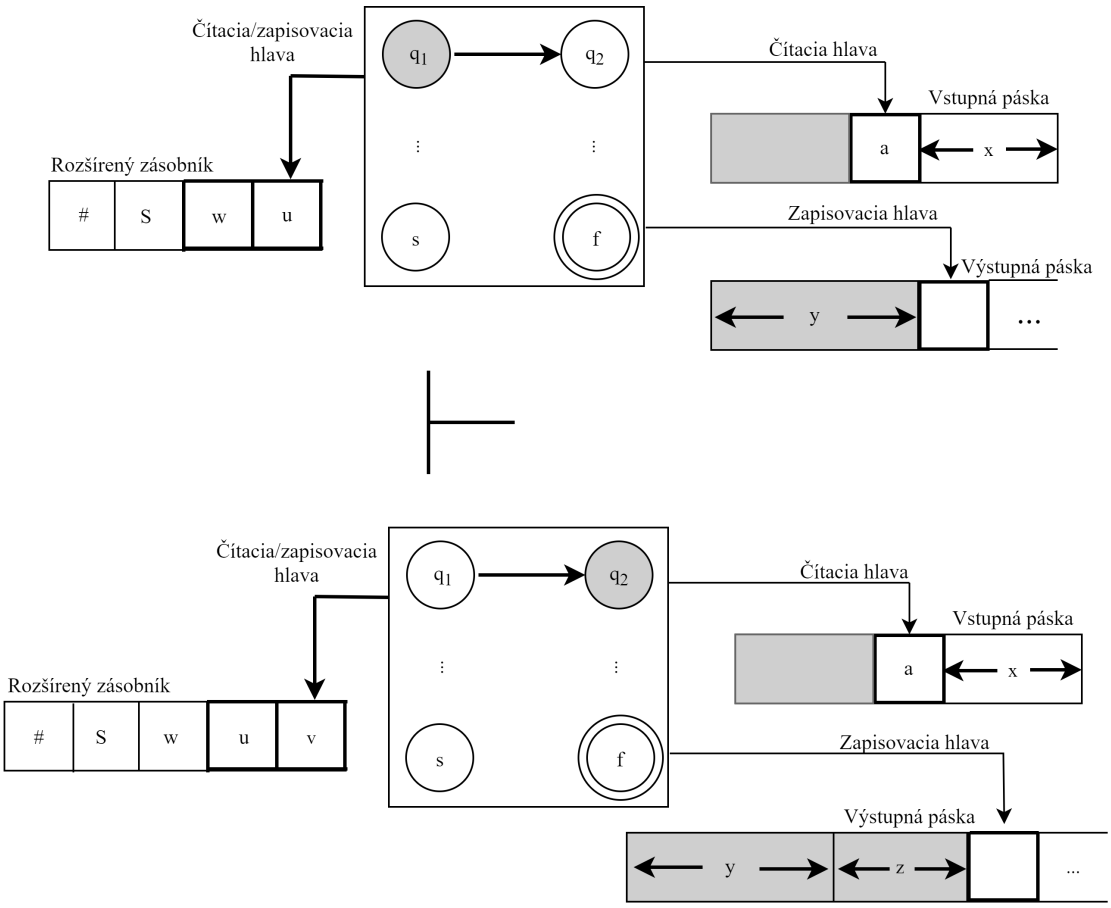
Uvažujme rozšírený zásobníkový prevodník $M = (Q, \Sigma, R, s, F)$, ktorý je schopný pracovať s dvomi vrcholnými symbolmi na zásobníku zároveň a konfiguráciu

$$\chi_1 = (q_1, ax, \alpha_1, y),$$

kde $q_1 \in Q$, $a \in \Sigma_I$, $x \in \Sigma_I^*$, $\alpha_1 \in \Sigma_{PD}^*$ a $y \in \Sigma_O^*$. Ako obr. 3.7 ukazuje, $(q_1, ax, \alpha_1, y)$ reprezentuje aktuálny stav $M$:

1. $q_1$ je aktuálny stav, v ktorom sa $M$ nachádza.
2. $ax$ je doposiaľ neprečítaný suffix vstupného slova.
3. $\alpha_1$ je dvojica najvyšších symbolov zásobníka.
4. $y$ je výstupné slovo vypísané do tohto momentu na výstupnú pásku. Výstupná hlava ukazuje na prvý štvorec za výstupným slovom.

Ďalej uvažujme výpočtový krok zodpovedajúci pravidlu: $r : uwq_1a \vdash vuq_2z$. Taktiež uvažujme, že $\{u, w, v, a, z\} \neq \{\epsilon\}$. $M$ vykoná výpočtový krok ilustrovaný na obr. 3.8 podľa pravidla $r$.




Obr. 3.8: Výpočtový krok rozšíreného zásobníkového prevodníku

### 3.3.5 Sekvencia krokov

Opäť uvažujme rozšírený zásobníkový prevodník $M = (Q, \Sigma, R, s, F)$. Nech $\chi$ je akákoľvek konfigurácia $M$. $M$ vykoná nula výpočtových krokov pri prechode z $\chi$ do $\chi$ podľa $\epsilon$:

$$\chi \vdash^0 \chi \ [\epsilon]$$

Ďalej uvažujme sekveciu konfigurácií

$$\chi_0, ..., \chi_n$$

pre $n \geq 1$ takú, pre ktorú platí:

$$\chi_{i-1} \vdash^n \chi_i \ [r_i]$$

kde $r_i \in R, i = 1, ..., n$. Tak potom $M$ vykoná $n$ výpočtových krokov z $\chi_0$ do $\chi_n$ podľa pravidiel $r_1 ... r_n$, čo je možné zapísať ako:

$$\chi_0 \vdash^n \chi_n \ [r_1 ... r_n]$$

Následne uvažujme dve konfigurácie $M$, a to $\chi_1$ a $\chi_2$.
1. Ak existuje $n \geq 1$ také, že platí: $\chi_1 \vdash^n \chi_2[\rho]$, tak môžeme uvažovať, že: $\chi_1 \vdash^+ \chi_2[\rho]$.
2. Ak existuje $n \geq 0$ také, že platí: $\chi_1 \vdash^n \chi_2[\rho]$, tak môžeme uvažovať, že: $\chi_1 \vdash^* \chi_2[\rho]$.
kde $\rho$ reprezentuje sekvenciu $n$ pravidiel z množiny $R$, pričom platí $\rho = \epsilon$ ak $n = 0$. [10]

### 3.3.6 Preklad definovaný rozšíreným zásobníkovým prevodníkom

Nech $M = (Q, \Sigma, R, s, F)$ je rozšírený zásobníkový prevodník, $x \in \Sigma_I^*$ a $y \in \Sigma_O^*$. $M$ prekladá $x$ na $y$ len ak:

$$(s, x, S, \epsilon) \vdash^* (f, \epsilon, z, y)$$

pre $f \in F$ a $z \in \Sigma_{PD}^*$. Tak potom preklad definovaný $M$, $T(M)$ je:

$$T(M) = \{(x, y) : x \in \Sigma_I^*, \ y \in \Sigma_O^*, \text{ a } M \text{ prekladá } x \text{ na } y\}$$

Príslušný vstupný jazyk k $T(M)$ je definovaný ako:

$$L_I(M) = \{x : (x, y) \in T(M) \text{ pre nejaké } y \in \Sigma_O^*\}$$

Príslušný výstupný jazyk k $T(M)$ je definovaný ako:

$$L_O(M) = \{x : (x, y) \in T(M) \text{ pre nejaké } y \in \Sigma_I^*\}$$

Nech $M = (Q, \Sigma, R, s, F)$ je konečný prevodník. Potom, ako bolo demonštrované v tejto podsekcii, obidva jazyky $L_I(M)$ aj $L_O(M)$ sú regulárne. [10]

# Kapitola 4

# Základy molekulárnej biológie

Táto kapitola bola prevzatá z [7], [11], [8], [6] a [3].

## 4.1 Bunka

Najjednoduchší organizmus schopný samostatného života, ktorý:

- je vnútorne rozdelený na jadro ako nositeľa genetickej informácie a cytoplazmu, v ktorej sa uskutočňujú metabolické procesy,
- proti okoliu je ohraničený plazmatickou membránou,
- vyznačuje sa všetkými základnými životnými funkciami (autoreprodukcia, metabolizmus, autoregulácia v závislosti na prostredí) s rozvinutým mechanizmom prenosu genetickej informácie (replikácia DNA, transkripcia, translácia). [7]

### 4.1.1 Prokaryotická bunka

Bunka, ktorej základným znakom je prítomnosť prokaryotického jadra (*nukleoidu*) v cytoplazme a neprítomnosť mitochondrií, plastidov, Golgiho systému a endoplazmatického retikula. [7]

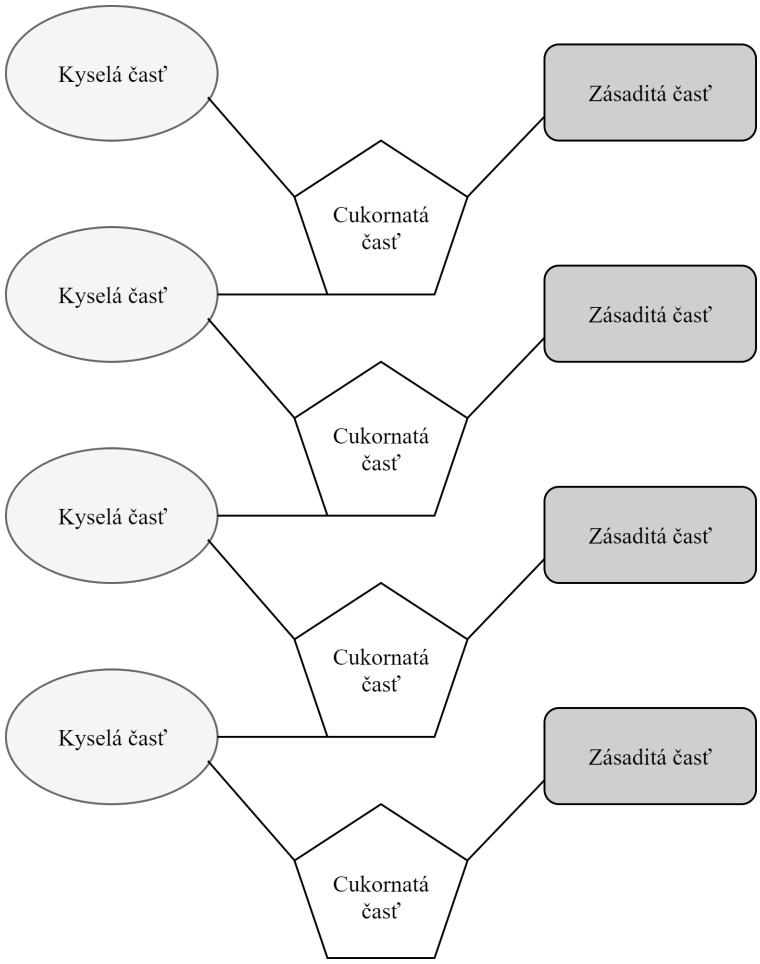
### 4.1.2 Eukaryotická bunka

Bunka, ktorej základným znakom je prítomnosť eukaryotického jadra oddeleného od cytoplazmy obalom jadra a prítomnosť mitochondrií (v živočíšnych bunkách) alebo mitochondrií a plastidov (v rastlinných bunkách), Golgiho systému a endoplazmatického retikula. [7]

## 4.2 Nukleové kyseliny

Nukleové kyseliny sú makromolekulové zlúčeniny prítomné vo všetkych bunkách (najčastejšie v jadre, v ktorom boli objavené ako prvé). Ich význam spočíva v prenose a uchovaní genetickej informácie a určení priebehu biosyntézy bielkovín v bunkách. Každá nukleová kyselina je zložená z troch základných častí:

- kyselá časť (zvyšok kyseliny fosforečnej $H_3PO_4$),
- zásaditá časť (pyrimidinová báza - Uracil(U), Cytozín(C), Tymín(T), alebo purinová báza - Adenín(A), Guanín(G)),
- cukornatá časť ($\beta$-D-ribofuranosu, 2-deoxy-$\beta$-D-ribofuranosu). [3]




Obr. 4.1:

## 4.3 Nukleotidy

Ak sa k *nukleosidu* (spojenie zásaditej s cukornatou zložkou) naviaže kyselá časť (zvyšok kyseliny fosforečnej), vznikne nukleotid. Keďže prichádza k reakcii kyseliny s hydroxilovou skupinou naviazanou na uhľovodíkovom raťazci, nastáva proces nazývaný *esterifikácia* (reakcia kyseliny s alkoholom $R-OH$ pri súčasnom odštiepení vody). Jednotlivé nukleotidy sú spolu prepojené a vytvárajú polynukleotidový reťazec, ktorý je základom štruktúry nukleových kyselín. Názov nukleotidu je tvorený z názvu nukleosidu a koncovky *fosfát*. Nukleové kyseliny obsahujú $10^3 - 10^6$ nukleotidov a ich relatívna molárna hmotnosť dosahuje až $10^9$. [3]

## 4.4 Typy nukleových kyselín

Rozlišujeme dva typy nukleových kyselín, a to **ribonukleové kyseliny**(RNA) a **deoxyribonukleové kyseliny**(DNA). O zaradení jednotlivej kyseliny rozhoduje v nej prítomný sacharid.

### 4.4.1 Ribonukleové kyseliny(RNA)

Ak je v molekule kyseliny prítomný výhradne sacharid $\beta$-D-ribofuranosa, ide o RNA. RNA môže obsahovať (zo zásaditých zložiek) výhradne *adenín*, *uracil*, *cytozín* a *guanín*. Molekuly RNA sú väčšinou jednovláknové, a preto rozlišujeme 3 základné typy ribonukleových kyselín:

- Mediátorová RNA (mRNA) – obsahuje prepis genetickej informácie.
- Transférová RNA (tRNA) – zaisťuje prenos aminokyselín do miest biosyntézy bielkovín.
- Ribozómová RNA (rRNA) – podieľa sa na tvorbe ribozómov. [3], [11]

### 4.4.2 Deoxyribonukleové kyseliny(DNA)

Ak je v molekule kyseliny prítomný sacharid 2-deoxy-$\beta$-D-ribofuranosa, ide o deoxyribonukleové kyseliny DNA. Zpomedzi zásaditých zložiek môže DNA obsahovať *adenín*, *tymín*, *cytozín* alebo *guanín*.

DNA je nositeľom genetickej informácie. Poradie jednotlivých zložiek – $A, T, C, G$ – v molekule udáva primárnu štruktúru DNA. Samotná molekula DNA má tvar dvojšrúbovnice. Usporiadanie molekuly do tohto tvaru umožňujú vodíkové väzby, ktoré vznikajú medzi atómami zásaditých zložiek. Vázanie vodíkovými väzbami je ovplyvnené komplementaritou bází, kedy sa *adenín* viaže s *tymínom* pomocou dvoch vodíkových vazieb (A=T) a *cytozín* s *guanínom* pomocou troch vodíkových vazieb ($C \equiv G$). Spojenie je prevedené tak, že hlavné vlákno má orientáciu od $5'$ ku $3'$ koncu a druhé komplementárne vlákno vo smere od $3'$ ku $5'$ koncu. Z to vyplýva, že na základe znalosti štruktúry jedného z reťazcov dvojšrúbovnice môžeme odvodiť štruktúru jemu komplementárneho reťazca. [3], [11]

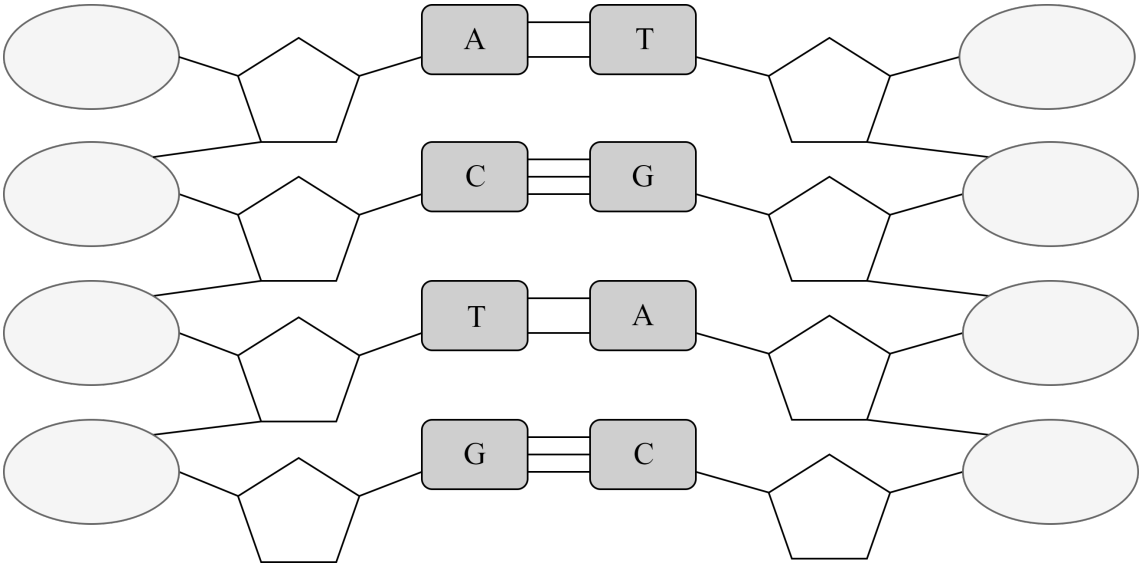
**Príklad – princíp komplementarity [8]**

Majme jeden z dvoch reťazcov dvojšrúbovnice DNA $r_1$ orientovaný od $5'$ ku $3'$ koncu. Dopočítajme reťazec $r_2$ podľa princípu komplementarity.

$r_1$: CCTTCACCAATTAAGGGG

Ak vychádzame z princípov komplementarity uvedených v podsekcii 4.4.2, podľa ktorých sa adenín viaže s tymínom a cytozín s guanínom, tak reťazen $r_2$ bude vyzerať nasledovne:

$r_2$: GGAAGTGGTTAATTCCCC




Obr. 4.2:

## 4.5 Prenos genetickej informácie

### 4.5.1 Replikácia

Replikácia DNA je proces tvorby kópií molekuly DNA, prevádzanej najmä pri benečnom delení. Ústredným prvkom je pri tom enzým *polymeráza*. Dvojšrúbovnica DNA je rozpletená na dve oddelené vlákna a ku každému vláknu sa v mieste zvanom *replikačná vidlica* postupne tvorí nové komplementárne vlákno. Enzým *polymeráza* taktiež zabezpečuje, že replikácia DNA prebehne bez chýb a budú vytvorené dve identické molekuly DNA. [11]

### 4.5.2 Transkripcia

Transkripcia je proces, pri ktorom sú špecifické časti DNA tzv. gény kopírované do molekuly RNA. Samotný proces začína rozpletením krátkeho úseku dvojšrúbovnice DNA, jeden z reťazcov potom slúži ako šablóna pre syntézu RNA. Rovnako ako pri replikácii DNA, je aj nukleotidová sekvencia RNA určená komplementárnym párovaním bází. Ak sa voľný ribonukleotid spáruje s deoxyribonukleotidom v reťazci DNA, ktorý slúži ako šablona, je kovalentne pripojený k rastúcemu reťazcu RNA v enzýmom katalyzovanej reakcii. Reťazec RNA vznikajúci transkripciou - *transkript* - je teda postupne predlžovaný a má rovnakú sekvenciu ako vlákno DNA, ktoré je komplementárne k reťazci DNA, slúžiacemu ako šablóna.

Transkripcia sa však líši od replikácie v niekoľkých zásadných bodoch. Oproti novovzniknutej DNA nezostáva RNA spojená so šablónou DNA vodíkovými väzbami. Namiesto toho dochádza ihneď za miestom, kde bol pridaný ribonukleotid, k obnoveniu dvojšrúbovitej štruktúry DNA a vytesnení vlákna RNA. Preto sú molekuly RNA jednovláknové. Vzhľadom na to, že dochádza k prepisu len malej časti DNA, sú molekuly RNA oveľa kratšie než DNA.

Enzým, ktorý prepisuje DNA do RNA, sa nazýva *RNA–polymeráza*, ktorá sa viaže na oblasť tzv. *promotoru* – špecifická oblasť pred počiatkom génu (cca 30 bází). Rovnako ako *DNA–polymeráza*, ktorá katalyzuje replikáciu, katalyzuje *RNA–polymeráza* vznik fosfodierestovej väzby, ktorá spojuje jednotlivé nukleotidy a vytvára tak cukor–fosfátovú kostru RNA. Výsledná molekula sa označuje ako mRNA (*mediátorová*). [11]

### 4.5.3 Translácia

Translácia je preklad genetickej informácie z poradia nukleotidov v mRNA (získanej procesom transkripcie) do poradia aminokyselím v polypeptidovom reťazci prostredníctvom genetického kódu. Aminokyseliny sa na miesto syntézy (do ribozómov) transportujú pomocou tRNA. Druh aminokyseliny určuje *kodón* (sekvencia troch po sebe nasledujúcich nukleotidov) mRNA a *komplementárny antikodón* tRNA. Samotný proces translácie sa skladá z nasledujúcich štyroch fáz:

- Naviazanie mRNA na ribozóm – na malú podjednotku ribozómu sa napája mRNA. Ribozóm sa po molekule presúva, pričom sa na voľný koniec mRNA môže nadviazať ďalší ribozóm. (vzniká *polyribozóm*, t. j. komplex molekuly mRNA, po ktorej sa posúvajú viaceré ribozómy) V ribozóme sa nachádzajú dve oblasti veľkosti *kodónu*, cez ktoré sa presúva mRNA a na ktoré sa viaže tRNA: *aminoacylové* miesto väzby (tu sa viaže tRNA nesúca svoju aminokyselinu) a *peptidylové* miesto väzby (tu sa viaže tRNA nesúca svoju aminokyselinu + už nadviazaný polypeptidový reťazec).

- Iniciácia translácie – Translácia začne, keď sa posunom na *peptidylové* miesto väzby dostane *iniciačný kodón* molekuly mRNA (AUG), na ktorý sa naviaže iniciačná tRNA s *antikodónom* (UAC) s nadviazanou iniciačnou aminokyselinou, *metionínom*.

- Proteosyntéza (Transport) – Posunom mRNA sa na *aminoacylové* miesto väzby dostane nasledujúci *kodón*, na ktorý sa napojí príslušný komplex *aminoacyl-tRNA*. Medzi obidvoma aminokyselinami vzniká peptidová väzba (vzniká *dipeptid*). Nastane ďalší posun o jeden *kodón* a tRNA sa z *aminoacylového* miesta presunie s nadviazaným *dipeptidom* na *peptidylové* miesto, odkiaľ vytesní predchádzajúcu tRNA. Na *aminoacylové* miesto sa dostane nový *kodón*, na ktorý sa napojí ďalší *aminoacyl-tRNA* s *komplementárnym antikodónom* a príslušnou aminokyselinou. Medzi aminokyselinami vzniká peptidová väzba (vzniká *tripeptid*). tRNA sa presunie na *peptidylové* miesto a vytesní predchádzajúcu tRNA. Tento proces sa opakuje a polypeptid narastá.

- Terminácia translácie – Translácia sa skončí, keď na *aminoacylové* miesto príde *terminačný kodón* mRNA (UGA, UAA, UAG), pre ktorý neexistuje *komplementárny antikodón* tRNA, nemôže sa teda napojiť ďalšia aminokyselina. K *terminačnému kodónu* sa naviaže *terminačný faktor*, ktorý hydrolyzuje väzbu medzi peptidovým reťazcom a tRNA. Novovytvorený polypeptidový reťazec sa z ribozómu uvoľní. [11], [5]

| | U | | C | | A | | G | |
|---|---|---|---|---|---|---|---|---|
| **U** | UUU | fenylalanín | UCU | serín | UAU | tyrosín | UGU | cysteín |
| | UUC | fenylalanín | UCC | serín | UAC | tyrosín | UGC | cysteín |
| | UUA | leucín | UCA | serín | UAA | **stop** | UGA | **stop** |
| | UUG | leucín | UCG | serín | UAG | **stop** | UGG | tryptofan |
| **C** | CUU | leucín | CCU | prolín | CAU | histidín | CGU | arginín |
| | CUC | leucín | CCC | prolín | CAC | histidín | CGC | arginín |
| | CUA | leucín | CCA | prolín | CAA | glutamín | CGA | arginín |
| | CUG | leucín | CCG | prolín | CAG | glutamín | CGG | arginín |
| **A** | AUU | izoleucín | ACU | treonín | AAU | asparagín | AGU | serín |
| | AUC | izoleucín | ACC | treonín | AAC | asparagín | AGC | serín |
| | AUA | izoleucín | ACA | treonín | AAA | lysín | AGA | arginín |
| | AUG | **metionín** | ACG | treonín | AAG | lysín | AGG | arginín |
| **G** | GUU | valín | GCU | alanín | GAU | kyselina asparagová | GGU | glycín |
| | GUC | valín | GCC | alanín | GAC | | GGC | glycín |
| | GUA | valín | GCA | alanín | GAA | kyselina glutámová | GGA | glycín |
| | GUG | valín | GCG | alanín | GAG | | GGG | glycín |

Obr. 4.3: Genetický kód

### 4.5.4 Príklad – prenos genetickej informácie

Majme jeden z dvoch reťazcov dvojšrúbovnice DNA $r_1$ orientovaný od $5'$ ku $3'$ koncu. Ďalej uvažujme, že ide o sekvenciu DNA prokaryotickej buňky. Vykonajte proces transkripcie a translácia a následne uveďte získaný proteín reprezentovaný ako reťazec aminokyselín $r_2$.

$r_1$: GCAATGTACTTCCATCTGGAATAGAGA

Po prebehnutí procesu transkripcie, získame mRNA v tvare:

$mRNA$: GCAAUGUACUUCCAUCUGGAAUAGAGA

Následne je potrebné nájsť v sekvencii *start* a *stop* kodón(y)

$mRNA$: GCA**AUG**UACUUCCAUCUGGAA**UAG**AGA

Nakoniec procesom translácie získame reťazec aminokyselín $r_2$

$r_2$: MetTyrPheHisLeuGlu

## 4.6 Formát FASTA

Formát FASTA je textový formát využívaný prevažne v oblastiach bioinformatiky a biochémie pre reprezentáciu sekvenciií nukleotidov alebo aminokyselín. Formát taktiež umožňuje definovanie názvov a komentárov k jednotlivým sekvenciám. V súčasnosti sa tento formát stal univerzálnym štandartom v oblasti bioinformatiky.

### 4.6.1 Vlastnosti formátu FASTA

Sekvencia vo formáte FASTA sa začína vždy jednoriadkovým popisom nasledovaným 1..n riadkov dlhou sekvenciou dát. Popisný riadok sa vždy musí začať symbolom ">" a predchádzať mu môže komentár, ktorého každý riadok sa musí začínať symbolom ";".

V samotnej sekvencii dát sa nesmie nachádzať prázdny riadok a žiaden riadok by nemal obsahovať viacej ako 80 znakov. Dátové sekvencie musia byť tiež zadané v štandardných IUB/IUPAC kódoch pre nukleové kyseliny a aminokyseliny. Tieto kódy môžu byť zadané malými alebo veľkými písmenami latinskej abecedy. Sekvencie je možné uložiť v súboroch s príponou *.fasta*, *.fa* pre ľubovolný druh sekvencie, *.fna* pre sekvenciu nukleových kyselín, .faa pre sekvenciu aminokyselín a mnohými ďalšími.

Organizácia NCBI(National Center for Biotechnology Information) definuje štandard pre identifikátory používané v riadkoch definujúcich jednotlivé sekvencie nukleových kyselín alebo aminokyselín. Tieto identifikátory umožňujú sekvenciám, ktoré boli získané z databázy, mať označenie a referenciu do danej databázy. Ale keďže tieto identifikátory nebudú ďalej v tejto práci využité pre demonštráciu príkladov, a taktiež ani aplikácia, ktorá bude výsledkom tejto práce, ich využitie podporavať nebude, tak ich syntax uvádzať nebudem. Je však dostupná na [6].

#### Príklad validnej sekvencie vo formáte FASTA

>validná sekvencia1
CGTTACATGAAGGTAGACCTTATCTCTCGTTACATGAAGGTAGACCT
CGTTACATGAAGGTAGACCTTATCTCTCGTTACATGAAGGTAGCTTA
ATGAAGGTAGACCTTATCTCTCGTTACATGAAGGT

*;volitelný komentár*
>validná sekvencia2
GCAAUGUACUUCCAUCUGGAAUAGGAATCTCTCGTTACATGAAATG
ATGAAGGTAGACCTTATCT

#### Príklad nevalidnej sekvencie vo formáte FASTA

>nevalidná sekvencia
*;volitelný komentár*
CGTTACATGAAGGTAGACCTTATCTCTCGTTACATGAAGGTAGACCT

CGTTACATGAAGGTAGACCTTATCTCTCGTTACATGAAGGTAGCTTA
ATGAAGGTAGACCTTATCTCTCGTTACATGAAGGT

| Typ | Význam | Typ | Význam | Typ | Význam |
|---|---|---|---|---|---|
| A | adenín | C | cytozín | G | guanín |
| T | tymín | N | A/G/C/T | U | uracil |
| K | G/T | S | G/C | Y | T/C |
| M | A/C | W | A/T | R | G/A |
| B | G/T/C | D | G/A/T | H | A/C/T |
| V | G/C/A | - | medzera | | |

Tabuľka 4.1: IUB/IUPAC kódy nukleových kyselín podporovaných formátom FASTA

| Typ | Význam | Typ | Význam | Typ | Význam |
|---|---|---|---|---|---|
| A | alanín | B | D/N | C | cysteín |
| D | kys. asparagová | E | kys. glutámová | F | fenylalanín |
| G | glycín | H | histidín | I | izoleucín |
| J | I/L | K | lysín | L | leucín |
| M | metionín | N | asparagín | O | pyrolysín |
| P | prolín | Q | glutamín | R | arginín |
| S | serín | T | treonín | U | selenocysteín |
| V | valín | W | tryptofan | Y | tyrosín |
| Z | kys. glutámová | X | ľubovolný | - | medzera |

Tabuľka 4.2: IUB/IUPAC kódy aminokyselín podporovaných formátom FASTA

# Kapitola 5

# Implementácia

## 5.1 Konceptuálny návrh

Ako som už uviedol v úvode 1, cieľom tejto práce je navrhnúť a implementovať aplikáciu pre prevod sekvencií DNA alebo mRNA na proteín (príp. proteínov) reprezentovaný vo forme reťazca (príp. reťazcov) aminokyselín. Pre implementáciu som zvolil programovací jazyk *Python 3.6* [1]. Výsledná aplikácia by mala byť spustiteľná na operačnom systéme Linux alebo Windows (pomocou podsystémov pre Linux).

Aplikácia je schopná spracovávať vstupný reťazec tromi rôznymi spôsobmi. Pri základnom, plno automatizovanom nastavení, program načíta reťazec, spracuje ho a vypíše výsledok na výstup. Druhým nastavením je automatické krokovanie, kedy program opäť načíta vstupný reťazec, ale v priebehu spracovania postupne vypisuje konfiguráciu použitého prevodníka v každom výpočetnom kroku. Posledným nastavením je manuálne krokovanie, kedy program načíta vstupný reťazec, vykoná jeden výpočtový krok, vypíše aktuálnu konfiguráciu a následne čaká na interakciu užívateľa (stisk klávesy) pre vykonanie ďalšieho kroku. Táto funkcionalita je však aplikovaná len na proces translácie, čiže prevodník $M_3$ (5.5.1).

Užívateľ je taktiež schopný ovplyvniť spôsob zadávania a formát vstupného reťazca, pričom je program schopný rozoznať prípadné zadanie sekvencie mRNA. Samotný vstupný reťazec je možné, buď zadať manuálne cez príkazový riadok ako parameter programu pri jeho spustení, ďalej ako obsah súboru uvedeného ako parameter programu pri spustení, pričom je možné v súbore uviesť viacero sekvencií (avšak tieto sekvencie musia spĺňať podmienky formátu FASTA), alebo je možné si nechať programom vygenerovať náhodnú sekvenciu DNA a s ňou následne pracovať.

Táto aplikácia využíva tri prevodníky. Konečný prevodník $M_1$ (5.3.1) reprezentujúci scanner (lexikálny analyzátor) overujúci validitu vstupnej sekvencie, a taktiež typ zadanej sekvencie (DNA alebo mRNA). Ďalej obsahuje konečný prevodník $M_2$ (5.4.1) vykonávajúci operáciu transkripcie a rozšírený zásobníkový prevodník $M_3$ (5.5.1) vykonávajúci operáciu translácie.

## 5.2 Triedy

Aplikácia pracuje s piatimi rôznymi triedami, pričom tri reprezentujú konfigurácie jednotlivých použitých prevodníkov. Všetky triedy sú definované v súbore *classes.py* a všetky obsahujú len triedne premenné bez konštruktorov a metód.

[1] Python 3.6 – dostupný na https://www.python.org/downloads/release/python-360/

Triedy $ScannerConfiguration$ a $TranscriptionConfiguration$ reprezentujú konfigurácie konečných prevodníkov $M_1$ (5.3.1) a $M_2$ (5.4.1). $ScannerConfiguration$ obsahuje dva atribúty: stav, v ktorom sa prevodník aktuálne nachádza a obsah vstupnej pásky zodpovedajúcej aktuálnej konfigurácii. Trieda $TranscriptionConfiguration$ obsahuje tri atribúty: stav, vstupnú a výstupnú sekvenciu, pričom položky stav a vstupná sekvencia sú významovo zhodné s atribútmi $M_1$ a atribút výstupná sekvencia obsahuje obsah výstupnej pásky v aktuálnej konfigurácii prevodníka $M_2$.

**Algoritmus 1:** Trieda ScannerConfiguration

```
Class contains
    stav;
    vstupnáSekvencia;
end
```

**Algoritmus 2:** Trieda TranscriptionConfiguration

```
Class contains
    stav;
    vstupnáSekvencia;
    výstupnáSekvencia;
end
```

Trieda $TranslationConfiguration$ reprezentuje konfiguráciu rozšíreného zásobníkového pravodníka $M_3$ (5.5.1). Obsahuje štyri atribúty: stav, vstupná a výstupná sekvencia a vrchol zásobníka, pričom prvé tri sú významovo zhodné s atribútami $M_2$. Atribút vrchol zásobníka obsahuje reťazec dĺžky dva znaky, reprezentujúce posledné dva prečítané symboly prevedníkom $M_3$.

**Algoritmus 3:** Trieda TranslationConfiguration

```
Class contains
    stav;
    vstupnáSekvencia;
    výstupnáSekvencia;
    vrcholZásobníka;
end
```

Ďalšou triedou je trieda $Token$ obsahujúca dva atribúty: typ tokenu a typ chyby. Jednotlivé typy hodnôt, ktoré môže trieda $Token$ nadobudnúť a ich význam sú reprezentované tabuľkou 5.1. Samotný token, ktorý je výstupom prevodníka $M1$ slúži pre rozpoznanie druhu vstupnej sekvencie (DNA, mRNA alebo chybná sekvencia). A na základe jeho hodnoty sa predá vstupná sekvencia najprv prevodníku $M_2$ (v rípade zadania DNA) pre proces transkripcie alebo priamo prevodníku $M_3$ (v prípade zadania mRNA) pre proces translácie.

**Algoritmus 4:** Trieda Token

```
Class contains
    tokenType;
    errorType;
end
```

Poslednou triedou je trieda $ArgParserOutput$ obsahujúca tri atribúty: zoznam typov tokenov, zoznam typov chýb a zoznam zadaných sekvencií. Keďže je možné na vstupe prog-

ramu zadať súbor vo formáte FASTA, ktorý môže obsahovať viacero sekvencií určených k prekladu, a taktiež dané sekvencie môžu byť rôzneho typu (DNA, mRNA, chybná), tak je nutné tieto vlastnosti detekovať, uchovať a podľa nich ďalej predávať sekvencie pre spracovanie prevodníkom $M_2$ a $M_3$.

**Algoritmus 5:** Trieda ArgParserOutput

```
Class contains
    tokenTypeList;
    errorTypeList;
    listOfSequences;
end
```

## 5.3 Prevodník $M_1$

### 5.3.1 Definícia

Nech $M_1 = (Q_1, \Sigma_1, R_1, s, F)$ je konečný prevodník, vykonávajúci lexikálnu analýzu vstupného reťazca, kde:

- $Q_1 = \{s, u, t, f\}$,
- $\Sigma_1 = \Sigma_{1I} \cup \Sigma_{1O}$,
- $\Sigma_{1I} = \{A, T, U, C, G, \#, i\}$, pričom $i$ je prvkom množiny symbolov $\beta$, pre ktorú platí: $\beta \cap \Sigma_{1I} = \emptyset$,
- $\Sigma_{1O} = \{(1,0), (2,0), (3,1), (3,2), (3,3)\}$, význam jednotlivých prvkov množiny sa nachádza v tabuľke 5.1,
- $R_1 = \{1 : sA \vdash s, 2 : sG \vdash s, 3 : sC \vdash s, 4 : sT \vdash t, 5 : sU \vdash u, 6 : si \vdash f(3,3), 7 : s\# \vdash f(1,0), 8 : uA \vdash u, 9 : uC \vdash u, 10 : uG \vdash u, 11 : uU \vdash u, 12 : uT \vdash f(3,2), 13 : u\# \vdash f(2,0), 14 : ui \vdash f(3,3), 15 : tA \vdash t, 16 : tC \vdash t, 17 : tG \vdash t, 18 : tT \vdash t, 19 : tU \vdash f(3,1), 20 : t\# \vdash f(1,0), 21 : ti \vdash f(3,3)\}$,
- $s \in Q_1$ je počiatočný stav a $f \in Q_1$ je koncový stav.

| Typ | Význam |
|---|---|
| (1,0) | Validná DNA sekvencia |
| (2,0) | Validná mRNA sekvencia |
| (3,1) | Chybný výskyt uracilu |
| (3,2) | Chybný výskyt tymínu |
| (3,3) | Nevalidný symbol v zadanej sekvencii |

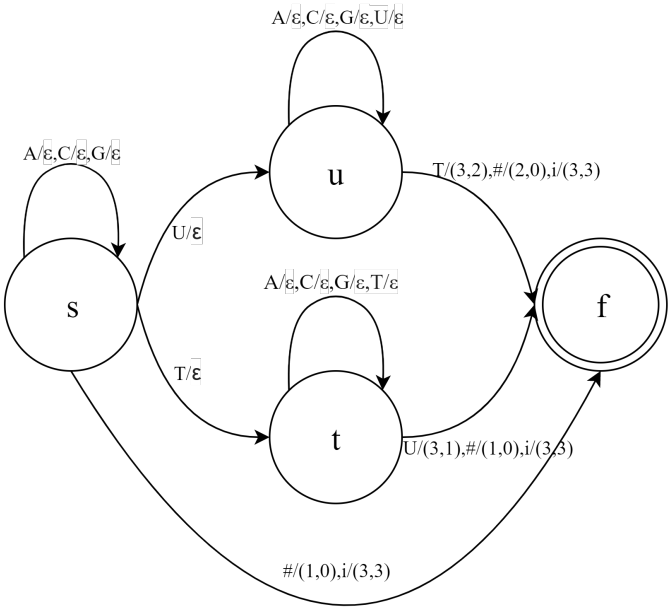
Tabuľka 5.1: Tokeny a ich význam

### 5.3.2 Funkcionalita $M_1$

Konečný prevodník $M_1$ slúži ako scanner (lexikálny analyzátor), ktorý validuje vstupnú sekvenciu, a zároveň rozhoduje o jej type (DNA alebo mRNA). Prevodník číta po jednom znaky zo vstupnej pásky a na základe pravidiel z množiny $R_1$ ich spracúva. Ak $M_1$ prejde do koncového stavu $f$, tak na výstup vráti príslušný token, viď 5.1. Pri spracovaní vstupnej sekvencie pracuje $M_1$ podľa nasledujúceho algoritmu:

**Algoritmus 6:** Funkcionalita $M_1$

```
Result: Príslušný token
Class Konfigurácia contains
    stav;
    vstupnáSekvencia;
end
Konfigurácia.vstupnáSekvencia += '#'; // vloženie zarážky na koniec
    vstupnej sekvencie
for symbol in Konfigurácia.vstupnáSekvencia do
    if symbol == i then
        nastav novú konfiguráciu; // význam 'i' viď 5.3.1
        nastav príslušný chybový token;
        return Token; // (3,3)
    else if symbol == '#' then
        nastav novú konfiguráciu; // narazili sme na zarážku, čiže celá
            sekvencia bola úspešne spracovaná
        nastav príslušný token;
        return Token; // (1,0), ak sme sa v predchádzajúcom výpočetnom
            kroku nachádzali v stave 's' alebo 't' a (2,0) ak v stave
            'u'
    else if ((symbol == 'T') and (sme v stave 'u')) then
        nastav novú konfiguráciu; // mRNA nemôže obsahovať zložku
            'T'(tymín), viď. 4.4.1
        nastav príslušný chybový token;
        return Token; // (3,2)
    else if ((symbol == 'U') and (sme v stave 't')) then
        nastav novú konfiguráciu; // DNA nemôže obsahovať zložku
            'U'(uracil), viď. 4.4.2
        nastav príslušný chybový token;
        return Token; // (3,1)
    else
        nastav novú konfiguráciu; // ak sa nachádzame v stave 's' a
            prečítame prvok z množiny {A,C,G}, v stave 'u' a prečítame
            prvok z množiny {A,C,G,U}, alebo ak sa nachádzame v stave
            't' a prečítame prvok z množiny {A,C,G,T}
    end
end
```




Obr. 5.1: Prevodník $M_1$

### 5.3.3 Príklad

**Spracovanie validného vstupu**

Majme vstupnú sekvenciu: $x = CGTTACAAGATCTCT$ a úvodnú konfiguráciu $M_1$: $(s, x, y)$, kde $s$ je počiatočný stav $M_1$ a $y$ reprezentuje obsah výstupnej pásky, ktorá je v tomto momente prázdna. Ak k $x$ pripojíme zarážku, jej spracovanie prevodníkom $M_1$ bude nasledovné:

$sC \vdash s$; $sG \vdash s$; $sT \vdash t$; $tT \vdash t$; $tA \vdash t$; $tC \vdash t$; $tA \vdash t$; $tA \vdash t$; $tG \vdash t$; $tA \vdash t$; $tT \vdash t$; $tC \vdash t$; $tT \vdash t$; $tC \vdash t$; $tT \vdash t$; $t\# \vdash f(1, 0)$

Prevodník $M_1$ úspešne spracoval celú sekvenciu, narazil na zarážku(#), prešiel do koncového stavu $f$ a na výstupe vrátil token $(1, 0)$ označujúci validnú DNA sekvenciu.

**Spracovanie nevalidného vstupu**

Tento raz majme vstupnú sekvenciu $x = CGTTACAAGAUCTCT$ a úvodnú konfiguráciu $M_1$: $(s, x, y)$, ktorá má tie isté vlastnosti ako konfigurácia v minulom príklade. Po pripojení zarážky # bude spracovanie $x$ nasledovné:

$sC \vdash s$; $sG \vdash s$; $sT \vdash t$; $tT \vdash t$; $tA \vdash t$; $tC \vdash t$; $tA \vdash t$; $tA \vdash t$; $tG \vdash t$; $tA \vdash t$; $tU \vdash f(3, 1)$

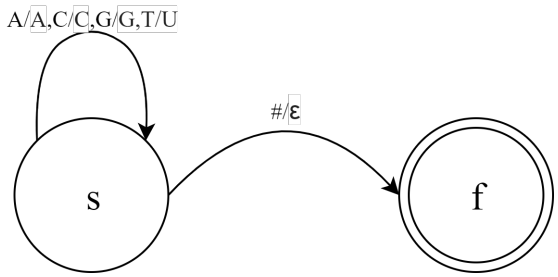
Prevodník $M_1$ prečítal symbol $U$(uracil) v stave $t$, ktorý značí, že už v jednom z predchádzajúcich krokov bol prečítaný symbol $T$ (tymín), čo znamená, že validná sekvencia môže byť len typu DNA, avšak $U$ sa nachádza len v sekvenciach typu mRNA. $M_1$, teda prejde do koncového stavu $f$ a vráti chybový token $(3, 1)$.

## 5.4 Prevodník $M_2$

### 5.4.1 Definícia

Nech $M_2 = (Q_2, \Sigma_2, R_2, s, F)$ je konečný prevodník, vykonávajúci operáciu transkripcie, kde:

- $Q_2 = \{s, f\}$,
- $\Sigma_2 = \Sigma_{2I} \cup \Sigma_{2O}$,
- $\Sigma_{2I} = \{A, T, C, G, \#\}$,
- $\Sigma_{2O} = \{A, U, C, G\}$,
- $R_2 = \{1 : sA \vdash sA, 2 : sT \vdash sU, 3 : sC \vdash sC, 4 : sG \vdash sG, 5 : s\# \vdash f\}$,
- $s \in Q_2$ je počiatočný stav,
- $f \in Q_2$ je koncový stav.




Obr. 5.2: Prevodník $M_2$

### 5.4.2 Funkcionalita $M_2$

Konečný prevodník $M_2$ vykonáva operáciu transkripcie 4.5.2 tak, že postupne spracúva vstupnú sekvenciu (DNA) a výsledkom jeho činnosti je sekvencia mRNA. Algoritmus 7 znázorňuje základný princíp činnosti $M_2$.

Prevodník $M_2$ obsahuje dva stavy, a to stav $s$, reprezentujúci počiatočný stav a stav $f$, ktorý je stavom konečným. Na začiatku spracovania sa $M_2$ nachádza v počiatočnom stave $s$, čítacia hlava ukazuje na prvý symbol vstupnej sekvencie a zapisovacia hlava ukazuje na prvé prázdne miesto na výstupnej páske (v tomto prípade je výstupná páska prázdna).

Ak prevodník $M_2$ v stave $s$ prečíta symbol z množiny $\{A, T, C, G\}$, tak zotrvá v stave $s$, čítacia hlava sa posunie o jeden symbol vpravo a na výstup vypíše príslušný symbol. Čiže po prečítaní $T$ je na výstupnú pásku vypísaný symbol $U$, inak sa vypísaný symbol zhoduje so symbolom aktuálne prečítaným zo vstupu.

V prípade, že v stave $s$ prevodník $M_2$ prečíta symbol # označujúci koniec vstupnej sekvencie, prejde $M_2$ do koncového stavu $f$ a vráti obsah výstupnej pásky.

**Algoritmus 7:** Funkcionalita $M_2$

```
Result: mRNA
Class Konfigurácia contains
    stav;
    vstupnáSekvencia; // výstup prevodníka M1 - validná sekvencia DNA
    výstupnáSekvencia; // sekvencia mRNA
end
Konfigurácia.vstupnáSekvencia += '#'; // vloženie zarážky na koniec
    vstupnej sekvencie
for symbol in Konfigurácia.vstupnáSekvencia do
    if symbol == '#' then
        prejdi do stavu 'f';
        skráť vstupnú sekvenciu;
        return Konfigurácia.výstupnáSekvencia;
    if symbol == 'T' then
        zotrvaj v stave 's';
        skráť vstupnú sekvenciu;
        pridaj 'U' na výstupnú pásku;
    else
        zotrvaj v stave 's';
        skráť vstupnú sekvenciu;
        uprav výstupnú sekvenciu;
    end
end
```

### 5.4.3 Príklad

Majme vstupnú sekvenciu: $x = CGTTACAAGATCTCT$ a úvodnú konfiguráciu $M_2$: $(s, x, y)$, kde $s$ je počiatočný stav $M_2$ a $y$ reprezentuje obsah výstupnej pásky, ktorá je v tomto momente prázdna. Ak k $x$ pripojíme zarážku, jej spracovanie prevodníkom $M_2$ bude nasledovné:

$sC \vdash sC$; $sG \vdash sG$; $sT \vdash sU$; $sT \vdash sU$; $sA \vdash sA$; $sC \vdash sC$; $sA \vdash sA$; $sA \vdash sA$; $sG \vdash sG$; $sA \vdash sA$; $sT \vdash sU$; $sC \vdash sC$; $sT \vdash sU$; $sC \vdash sC$; $sT \vdash sU$; $s\# \vdash f$;

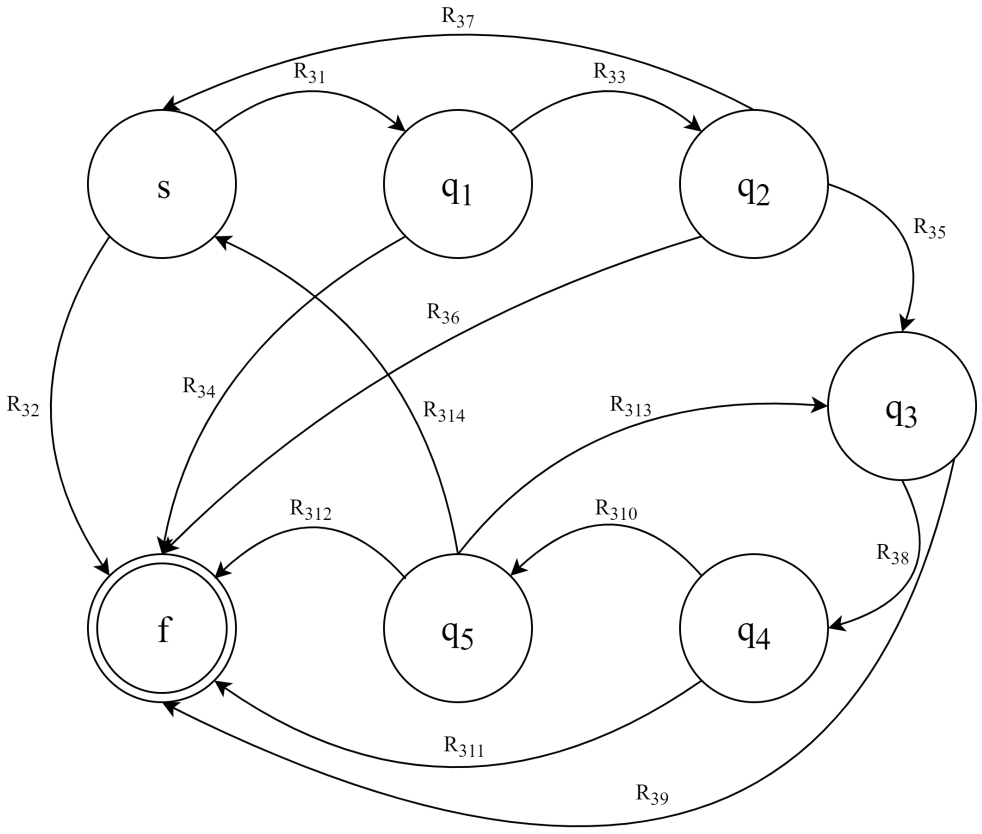
Prevodník $M_2$ v tomto prípade spracoval celú vstupnú sekvenciu $x$, narazil na zarážku $\#$, prešiel do koncového stavu $f$ a vrátil obsah výstupnej pásky $y = CGUUACAAGAUCUCU$.

## 5.5 Prevodník $M_3$

### 5.5.1 Definícia

Nech $M_3 = (Q_3, \Sigma_3, R_3, s, F)$ je rozšírený zásobníkový prevodník, vykonávajúci operáciu translácie, kde:

- $Q_3 = \{s, q_1, q_2, q_3, q_4, q_5, f\}$,
- $\Sigma_3 = \Sigma_{3I} \cup \Sigma_{3O} \cup \Sigma_{PD}$,
- $\Sigma_{3I} = \{A, U, C, G, \#\}$,
- $\Sigma_{3O} =$ *{Phe, Leu, Ile, Val, Ser, Pro, Met, Thr, Ala, Tyr, His, Gln, Asn, Lys, Asp, Glu, Cys, Trp, Arg, Gly}*,
- $\Sigma_{PD} = \{A, U, C, G, S, \$\}$,
- $R_3 = R_{31} \cup R_{32} \cup R_{33} \cup R_{34} \cup R_{35} \cup R_{36} \cup R_{37} \cup R_{38} \cup R_{39} \cup R_{310} \cup R_{311} \cup R_{312} \cup R_{313} \cup R_{314}$, pričom $\forall R_{3n}$, kde $n = 1..14$ platí, že $R_{3n} \cap (R_3 - R_{3n}) = \emptyset$,
- $s \in Q_3$ je počiatočný stav,
- $f \in Q_3$ je koncový stav.




Obr. 5.3: Prevodník $M_3$

**Množiny pravidiel** $R_{36}, R_{39}, R_{311}, R_{312}$

Majme dve množiny $X, Y$ obsahujúce zhodné štyri prvky $\{U, C, A, G\}$. Vykonaním operácie karteziánskeho súčinu nad týmito množinami získame množinu $V$ obsahujúcu 16 usporiadaných dvojích takých, že prvý prvok každej dvojice je prvkom množiny $X$ a druhý prvkom množiny $Y$. Ďalej uvažujme predpis pravidla $r : xyq\# \vdash S\$p$, pre ktoré platí:

- $x \in (X \cap \Sigma_{PD})$,
- $y \in (Y \cap \Sigma_{PD})$,
- $\# \in \Sigma_{I3}$ je zarážka označujúca koniec vstupnej sekvencie,
- $S, \$ \in \Sigma_{PD}$ je dvojica symbolov, ktorých konkatenácia reprezentuje počiatočný reťazec na vrchole zásobníka,
- $q, p \in Q_3$,
- $\vdash \notin (\Sigma_{I3} \cup \Sigma_{O3} \cup \Sigma_{PD})$ je špeciálny symbol označujúci prechod z jednej konfigurácie do druhej.

Následne, ak v pravidle $r$ nahradíme $q$ symbolom $q_2 \in Q_3$, $p$ symbolom $f \in Q_3$ a dvojicu $[x, y]$ postupne nahradíme všetkými usporiadanými dvojicami množiny $V$, získame množinu pravidiel $R_{36}$. Obdobným spôsobom je možné získať aj množiny pravidiel $R_{39}$(výmenou $q, p$ za $q_3, f \in Q_3$), $R_{311}$(výmenou $q, p$ za $q_4, f \in Q_3$) a $R_{312}$(výmenou $q, p$ za $q_5, f \in Q_3$).

**Množiny pravidiel** $R_{37}, R_{38}, R_{310}$

Majme tri množiny $X, Y, Z$ obsahujúce rovnaké štyri prvky $\{U, C, A, G\}$. Ak vykonáme operáciu karteziánskeho súčinu nad týmito tromi množinami, získame množinu $V$ obsahujúcu 64 usporiadaných trojíc takých, že prvý prvok každej trojice je prvkom možiny $X$, druhý množiny $Y$ a tretí prvkom množiny $Z$. Ďalej uvažujme predpis pravidla $r : xyqz \vdash zxp$, kde $x, y, z, \vdash, q, p$ pre ktoré platí:

- $x \in (X \cap \Sigma_{PD})$,
- $y \in (Y \cap \Sigma_{PD})$,
- $z \in (Z \cap \Sigma_{3I} \cap \Sigma_{PD})$,
- $q, p \in Q_3$,
- $\vdash \notin (\Sigma_{I3} \cup \Sigma_{O3} \cup \Sigma_{PD})$ je špeciálny symbol označujúci prechod z jednej konfigurácie do druhej.

Ak v pravidle $r$ nahradíme $q$ symbolom $q_3 \in Q_3$ a $p$ symbolom $q_4 \in Q_3$ a trojicu $[x, y, z]$ postupne nahradíme všetkými usporiadanými trojicami množiny $V$, získame množinu pravidiel $R_{38}$, obsahujúcu 64 pravidiel. Obdobným spôsobom je možné získať množinu pravidiel $R_{310}$, pri nahradení $q, p$ za $q_4, q_5 \in Q_3$.

Ďalej, ak pravidlo $r$ upravíme do podoby: $r : xyqz \vdash S\$p$, kde $S\$ \in \Sigma_{PD}$ je dvojica symbolov, ktorých konkatenácia reprezentuje počiatočný reťazec na vrchole zásobníka. A následne $q, p$ nahradíme $q_2, s \in Q_3$, a trojicu $[x, y, z]$ postupne nahradíme všetkými usporiadanými trojicami množiny $V$, získame množinu 64 pravidiel $R'$. Ak potom vykonáme operáciu $R'$ - $UAq_2G \vdash GUq_2$ získame množinu pravidiel $R_{37}$.

| Množina | Zoznam pravidiel |
|---|---|
| $R_{31}$ | $1 : S\$sA \vdash ASq_1$, $2 : S\$sU \vdash USq_1$, $3 : S\$sC \vdash CSq_1$, $4 : S\$sG \vdash GSq_1$ |
| $R_{32}$ | $1 : S\$s\# \vdash S\$f$ |
| $R_{33}$ | $1 : ASq_1A \vdash AAq_2$, $2 : ASq_1U \vdash UAq_2$, $3 : ASq_1C \vdash CAq_2$<br>$4 : ASq_1G \vdash GAq_2$, $5 : USq_1A \vdash AUq_2$, $6 : USq_1U \vdash UUq_2$<br>$7 : USq_1C \vdash CUq_2$, $8 : USq_1G \vdash GUq_2$, $9 : CSq_1A \vdash ACq_2$<br>$10 : CSq_1U \vdash UCq_2$, $11 : CSq_1C \vdash CCq_2$, $12 : CSq_1G \vdash GCq_2$<br>$13 : GSq_1A \vdash AGq_2$, $14 : GSq_1U \vdash UGq_2$, $15 : GSq_1C \vdash CGq_2$,<br>$16 : GSq_1G \vdash GGq_2$ |
| $R_{34}$ | $1 : USq_1\# \vdash S\$f$, $2 : CSq_1\# \vdash S\$f$, $3 : ASq_1\# \vdash S\$f$, $4 : GSq_1\# \vdash S\$f$ |
| $R_{35}$ | $1 : UAq_2G \vdash GUq_3Met$ |
| $R_{313}$ | $1 : UUq_5U \vdash UUq_3Phe$, $2 : UUq_5C \vdash CUq_3Phe$, $3 : UUq_5A \vdash AUq_3Leu$<br>$4 : UUq_5G \vdash AUq_3Leu$, $5 : UCq_5U \vdash UUq_3Leu$, $6 : UCq_5C \vdash CUq_3Leu$<br>$7 : UCq_5A \vdash AUq_3Leu$, $8 : UCq_5G \vdash GUq_3Leu$, $9 : UAq_5U \vdash UUq_3Ile$<br>$10 : UAq_5C \vdash CUq_3Ile$, $11 : UAq_5A \vdash AUq_3Ile$, $12 : UAq_5G \vdash GUq_3Met$<br>$13 : UGq_5U \vdash UUq_3Val$, $14 : UGq_5C \vdash CUq_3Val$, $15 : UGq_5A \vdash AUq_3Val$<br>$16 : UGq_5G \vdash GUq_3Val$, $17 : CUq_5U \vdash UCq_3Ser$, $18 : CUq_5C \vdash CCq_3Ser$<br>$19 : CUq_5A \vdash ACq_3Ser$, $20 : CUq_5G \vdash GCq_3Ser$, $21 : CCq_5U \vdash UCq_3Pro$<br>$22 : CCq_5C \vdash CCq_3Pro$, $23 : CCq_5A \vdash ACq_3Pro$, $24 : CCq_5G \vdash GCq_3Pro$<br>$25 : CAq_5U \vdash UCq_3Thr$, $26 : CAq_5C \vdash CCq_3Thr$, $27 : CAq_5A \vdash ACq_3Thr$<br>$28 : CAq_5G \vdash GCq_3Thr$, $29 : CGq_5U \vdash UCq_3Ala$, $30 : CGq_5C \vdash CCq_3Ala$<br>$31 : CGq_5A \vdash ACq_3Ala$, $32 : CGq_5G \vdash GCq_3Ala$, $33 : AUq_5U \vdash UAq_3Tyr$<br>$34 : AUq_5C \vdash CAq_3Tyr$, $35 : ACq_5U \vdash UAq_3His$, $36 : ACq_5C \vdash CAq_3His$<br>$37 : ACq_5A \vdash AAq_3Gln$, $38 : ACq_5G \vdash GAq_3Gln$, $39 : AAq_5U \vdash UAq_3Asn$<br>$40 : AAq_5C \vdash CAq_3Asn$, $41 : AAq_5A \vdash AAq_3Lys$, $42 : AAq_5G \vdash GAq_3Lys$<br>$43 : AGq_5U \vdash UAq_3Asp$, $44 : AGq_5C \vdash CAq_3Asp$, $45 : AGq_5A \vdash AAq_3Glu$<br>$46 : AGq_5G \vdash GAq_3Glu$, $47 : GUq_5U \vdash UGq_3Cys$, $48 : GUq_5C \vdash CGq_3Cys$<br>$49 : GUq_5G \vdash GGq_3Trp$, $50 : GCq_5U \vdash UGq_3Arg$, $51 : GCq_5C \vdash CGq_3Arg$<br>$52 : GCq_5A \vdash AGq_3Arg$, $53 : GCq_5G \vdash GGq_3Arg$, $54 : GAq_5U \vdash UGq_3Ser$<br>$55 : GAq_5C \vdash CGq_3Ser$, $56 : GAq_5A \vdash AGq_3Arg$, $57 : GAq_5G \vdash GGq_3Arg$<br>$58 : GGq_5U \vdash UGq_3Gly$, $59 : GGq_5C \vdash CGq_3Gly$, $60 : GGq_5A \vdash AGq_3Gly$<br>$61 : GGq_5G \vdash GGq_3Gly$ |
| $R_{314}$ | $1 : AUq_5A \vdash S\$s$, $2 : AUq_5G \vdash S\$s$, $3 : GUq_5A \vdash S\$s$ |

Tabuľka 5.2: Pravidlá $M_3$

### 5.5.2 Funkcionalita $M_3$

Rozšírený zásobníkový prevodník $M_3$ vykonáva operáciu translácie tak, že postupne spracúva vstupnú sekvenciu (mRNA - výstup prevodníka $M_2$) a výsledkom je jeden alebo viacero proteínov vo forme sekvencie aminokyselín. Nasledujúce riadky detailne opisujú spracovanie vstupnej sekvencie, pričom algoritmus 8 zobrazuje len základný princíp spracovania vstupnej sekvencie.

$M_3$ obsahuje sedem rôznych stavov, z čoho je jeden počiatočný ($s$) a jeden koncový ($f$). Na začiatku spracovania sa prevodník nachádza v počiatočnom stave ($s$), čítacia hlava ukazuje na prvý znak vstupnej sekvencie (mRNA), zapisovacia hlava ukazuje na prvé prázdne pole (v tomto prípade je celá páska prázdna) a zásobník obsahuje len reťazec $S\$$ označujúci vrchol zásobníka.

Zo stavu $s$ sa može $M_3$ dostať len do stavu $q_1$ a to tak, že prečíta symbol, pre ktorý platí $\Sigma_{I3} - \#$, pričom sa prečítaný znak vloží na vrchol zásobníka, čítacia hlava sa posunie o jeden symbol vpravo a zapisovacia hlava ostane na svojom mieste. V prípade prečítania symbolu $\#$ označujúceho koniec vstupnej sekvencie, $M_3$ prejde do koncového stavu $f$, vyprázdni zásobník a vráti obsah výstupnej pásky.

Prevodník $M_3$ sa v stave $q_1$ pri prechode do $q_2$ alebo $f$ správa obdobne ako v stave $s$ opísanom vyššie. Ak sa $M_3$ nachádza v stave $q_2$, na vrchole zásobníka sa nachádza reťazec $UA$ a čítacia hlava ukazuje na symbol $G$, tak $M_3$ prejde do stavu $q_3$, upravý reťazec na vrchole zásobníka, posunie čítaciu hlavu o jeden symbol vpravo a na výstupnú pásku vypíše $Met$ (Metionín - start kodón pre aminokyselinu). Ak v stave $q_2$ prečíta symbol $\#$ označujúci koniec vstupnej sekvencie, prejde do koncového stavu $f$, vyprázdni zásobník a vráti obsah výstupnej pásky. V ostatných prípadoch zodpovedajúcim pravidlám v množine $R_{37}$ vyprázdni zásobník až po počiatočný reťazec $S\$$, posunie čítaciu hlavu smerom vpravo a presunie sa do stavu $s$.

V prípade, že sa $M_3$ nachádza v stavoch $q_3$ alebo $q_4$, tak prevodník prečíta symbol zo vstupnej pásky, ak ide o symbol $\#$, prejde do stavu $f$, vyprázdni zásobník a vráti obsah výstupnej pásky. Avšak v tomto prípade bude posledný proteín nekompletný, pretože zadaná sekvencia (mRNA) neobsahuje rovnaký počet *start* a *stop* kodónov. Ak $M_3$ v jednom z týchto stavov prečíta symbol, pre ktorý platí: $\Sigma_{I3} - \#$, tak zmení reťazec na vrchole zásobníka, posunie čítaciu hlavu o jeden symbol vpravo, a ak ide o stav $q_3$, tak prejde do stavu $q_4$, obdobne, zo stavu $q_4$ prejde do $q_5$.

Ak sa $M_3$ nachádza v stave $q_5$ a prečíta symbol patriaci do množiny $\Sigma_{I3} - \#$, tak na základe prečítaného symbolu a dvojice znakov tvoriacich reťazec na vrchole zásobníka určí, či ide o *stop* kodón. ak áno, tak $M_3$ prejde do stavu $s$, vyprázdni zásobník až po počiatočný reťazec $S\$$, a posunie čítaciu hlavu o jeden symbol vpravo. V prípade, že nejde o *stop* kodón, tak $M_3$ prejde do stavu $q_3$, upravý reťazec na vrchole zásobníka, posunie čítaciu hlavu o jeden symbol vpravo a na základe tabuľky 4.3 vypíše príslušnú skratku aminokyseliny patriacej do $\Sigma_{O3}$ na výstupnú pásku. V prípade, že $M_3$ v stave $q_5$ prečíta symbol $\#$ prejde do stavu $f$, vyprázdni zásobník a vráti obsah výstupnej pásky. Avšak opäť v tomto prípade bude posledný spracúvaný proteín nekompletný, pretože zadaná vstupná sekvencia (mRNA) neobsahuje rovnaký počet *start* a *stop* kodónov.

**Algoritmus 8:** Funkcionalita $M_3$

```
Result: Proteín/proteíny vo forme sekvencie aminokyselín
Class Konfigurácia contains
    stav;
    vrcholZásobníka; // reťazec dvoch znakov
    vstupnáSekvencia; // výstup prevodníka M2 alebo manuálne zadaná
        sekvencia mRNA na vstupe
    výstupnáSekvencia;// preteín/proteíny vo forme sekvencie
        aminokyselín
end
Konfigurácia.vstupnáSekvencia += '#'; // vloženie zarážky na koniec
    vstupnej sekvencie
for symbol in Konfigurácia.vstupnáSekvencia do
    if symbol == '#' then
        vyprázdni zásobník;
        prejdi do stavu 'f';
        skráť vstupnú sekvenciu;
        return Konfigurácia.výstupnáSekvencia;
    else
        zmeň stav;
        uprav reťazec na vrchole zásobníka;
        skráť vstupnú sekvenciu;
        uprav výstupnú sekvenciu;
    end
end
```

### 5.5.3 Príklad

Majme vstupnú sekvenciu: $x = GCAAUGUACUUCUAGAGA$ a úvodnú konfiguráciu $M_1$: $(s, x, \alpha, y)$, kde $s$ je počiatočný stav $M_3$, $y$ reprezentuje obsah výstupnej pásky, ktorá je v tomto momente prázdna a $\alpha = S\$$ je počiatočný reťazec na vrchole zásobníka. Ak k $x$ pripojíme zarážku, jej spracovanie prevodníkom $M_3$ bude nasledovné:

$S\$sG \vdash GSq_1$; $GSq_1C \vdash CGq_2$; $CGq_2A \vdash S\$s$; $S\$sA \vdash ASq_1$; $ASq_1U \vdash UAq_2$;
$UAq_2G \vdash GUq_3Met$; $GUq_3U \vdash UGq_4$; $UGq_4A \vdash AUq_5$; $AUq_5C \vdash CAq_3Tyr$;
$CAq_3U \vdash UCq_4$; $UCq_4U \vdash UUq_5$; $UUq_5C \vdash CUq_3Phe$; $CUq_3U \vdash UCq_4$; $UCq_4A \vdash AUq_5$;
$AUq_5G \vdash S\$s$; $S\$sA \vdash ASq_1$; $ASq_1G \vdash GAq_2$; $GAq_2A \vdash S\$s$; $S\$s\# \vdash S\$f$;

Celá vstupná sekvencia mRNA obsahovala 6 kodónov. Aby mohol začať preklad proteínu (od stavu $q_3$ ďalej), musel $M_3$ najprv naraziť na *start* kodón $AUG$, ktorý v tomto prípade bol ako druhý v poradí. Nasledovali 2 kodóny patriace do proteínu ($UAC$ a $UUC$) a po nich nasledoval *stop* kodón, ktorý značí koniec aktuálneho proteínu. Posledný kodón $AGA$ už nepatril žiadnemu proteínu. Po spracovaní týchto 6 kodónov narazil $M_3$ na zarážku #, presunul sa do koncového stavu $f$ a vrátil obsah výstupnej pásky. Výsledný proteín má tvar: $MetTyrPhe$.

## 5.6 Moduly

### IBTProjMainModule.py

Tento modul obsahuje len hlavnú funkciu (*main()*) programu, ktorá riadi činnosť celej aplikácie.

Ihneď po spustení aplikácie je vytvorená inštancia triedy *ArgParserOutput* a následne sú jej atribúty naplnené výstupom modulu *argParser* obsahujúcim zoznam zadaných vstupných sekvencií, typov tokenov a chybových kódov pre každú zo sekvencií. (Typ tokenu a chybový kód každej zo zadaných sekvencií je výstupom prevodníka $M_1$ (Token – 5.1).) Následne aplikácia cyklí cez všetky sekvencie v zozname vstupných sekvencií.

Ak má typ tokenu aktuálne spracúvanej sekvencie hodnotu 1 a chybový kód hodnotu 0, ide o sekvenciu DNA. Tá sa ďalej predá prevodníku $M_2$ pre proces transkripcie a jeho výstup ďalej prevodníku $M_3$ pre proces translácie. Výsledky oboch procesov sa následne zobrazia užívateľovi v prehľadnej tabuľke.

V prípade, že typ tokenu aktuálne spracovávanej sekvencie je 2 a jej chybový kód 0, ide o sekvenciu mRNA. Pri tomto type vstupnej sekvencie sa sekvencia predáva priamo prevodníku $M_3$ pre proces translácie. Proces transkripcie sa vynecháva.

Ak typ tokenu obsahuje hodnotu 3, tak sa na základe hodnoty chybového kódu vyvolá výnimka, zobrazí sa príslušná chybová hláška a program sa ukončí.

Funkcia *main()* taktiež postupne ukladá úspešne spracované sekvencie do výstupného súboru *output.txt*.

### argParser.py

Tento modul zabezpečuje kontrolu a spracovanie vstupných parametrov (povinných aj voliteľných). Súčasťou parametrov programu je vstupná sekvencia zadaná priamo (ako parameter programu), ako obsah súboru alebo je zadaný vstupný parameter $-g$ (prípadne jeho alternatívy), ktorý definuje, že vstupná sekvencia má byť náhodne vygenerovaná.

V tomto prípade sa volá hlavná funkcia modulu *generator*, ktorá vracia náhodne vygenerovanú, 150-znakov dlhú sekvenciu, ktorá vždy reprezentuje sekvenciu DNA. Funkcia *main()* modulu *argParser* túto sekvenciu vloží do atribútu *listOfSequences* (zoznam sekvencií) inštancie triedy *ArgParserOutput*, atribút typ tokenu a chybový kód nastavý na hodnoty 1 a 0 a následne vráti túto inštanciu. (V tomto prípade nie je potrebné získať *Token* pomocou prevodníka $M_1$, pretože generovaná sekvencia bude vždy validná.)

Ak je vstupná sekvencia zadaná priamo na vstupe, tak sa všetky symboly vstupnej sekvencie, ktoré boli zadané malými písmenami prevedú na veľké. Ak bol zadaný voliteľný parameter *-complement*, tak sa volá funkcia *createComplementarySequence()*, ktorá podľa princípu komplementarity (4.4.2) vytvorí komplementárnu sekvenciu, ktorú následne predá na vstup prevodníka $M_1$ (modul *scanner*), ktorý vráti *Token* k tejto sekvencii. Ak voliteľný parameter *-complement* zadaný nebol, tak sa na vstup prevodníka $M_1$ vloží samotná zadaná sekvencia. Atribútmi získaného tokenu a vstupnou (komplementárnou) sekvenciou sa naplnia atribúty inštancie triedy *ArgParserOutput*, ktorú modul vracia.

V prípade, že je vstupná sekvencia zadaná ako obsah súboru (v tomto prípade je možné v súbore zadať aj viacero sekvencií, viď. 4.6), tak sa opäť zistí, či na vstupe nebol zadaný voliteľný parameter *-complement*. Ak áno, pre každú zo zadaných sekvencií sa vytvorí sekvencia komplementárna, ktorá sa vloží na vstup prevodníka $M_1$. Ak tento parameter zadaný nebol, tak sa na vstup $M_1$ vloží každá zo zadaných sekvencií. Atribútmi získaného tokenu

(výstup $M_1$) a vstupnou (komplementárnou) sekvenciou sa opäť naplnia atribúty inštancie triedy *ArgParserOutput*, ktorú modul vracia.

Ak je na vstupe zadaný len parameter *-help* (alebo jeho alternatívy), tak je užívateľovi zobrazená nápoveda.

#### generator.py

Modul generátora slúži k vygenerovaniu sekvencie DNA dĺžky 150 symbolov. Týchto 150 symbolov reprezentuje 50 kodónov. Ako je uvedené v časti 4.5.3, preklad samotného proteínu začína nájdením iniciačného (*start*) kodónu a ukončuje narazením na koncový (*stop*) kodón. Čiže generátor musí zabezpečiť, aby sa táto dvojica kodónov vyskytla v tejto sekvencii aspoň raz.

Generátor teda začne po jednom generovať kodóny, pričom si vždy udržuje informáciu o tom, či generuje kodóny patriace proteínu, pretože *stop* sa môže vyskytnúť až za *start* kodónom. Ak aktuálne generuje kodóny mimo proteínu, tak je 25% šanca, že ďalší kodón bude *start* kodón (*ATG*). Po jeho vygenerovaní sa pokračuje v generovaní kodónov (tento raz patriacich do proteínu), pričom je opäť 25% šanca, že ďalším vygenerovaným kodónom bude jeden z trojice *stop* kodónov (*TAA, TAG, TGA*).

Za predpokladu, že výsledná sekvencia bude obsahovať 50 kodónov, tak aj v prípade, ak sa generujú kodóny patriace do proteínu a po vygenerovaní ďalšieho kodónu by už sekvencia bola kompletná, tak sa namiesto náhodnej voľby vygeneruje *stop* kodón, ktorý sekvenciu zakončí. Taktiež, ak sa aktuálne generuje časť mimo proteínu a už ostáva vygenerovať posledný kodón, tak v tomto prípade nie je možné vygenerovať *start* ani *stop* kodón.

#### scanner.py, transcription.py, translation.py

Moduly *scanner*, *transcription* a *translation* implementujú prevodníky $M_1$ (5.3.1), $M_2$ (5.4.1) a $M_3$ (5.5.1).

#### classes.py, extendedStack.py, printer.py

Modul *classes* obsahuje definície všetkých tried (5.2), s ktorými aplikácia pracuje. *ExtendedStack* implementuje rozšírený zásobník pomocou zoznamu, pričom je možné pracovať s reťazcom o dĺžke dvoch symbolov na vrchole tohto zásobníka. Modul *printer* slúži k zobrazeniu vstupnej sekvencie, výsledku transkripcie (mRNA) a produktu translácie (sekvecie aminokyselín). Prípadne k zobrazeniu výstupov definovaných voliteľnými parametrami (tabuľka genetického kódu, výpis translácie po jednotlivých krokoch).

# Kapitola 6

# Testovanie a validácia získaných dát

V tejto kapitole sa nachádza testovanie navrhnutej aplikácie a tiež validácia získaných dát. Samotné testovanie je rozdelené na dve časti. Prvá zobrazuje spustenie a výpis výsledkov pri použití rôznych vstupných parametrov a druhá časť je venovaná porovnaniu výstupu navrhovanej aplikácie s výstupom webovej aplikácie Sequence Manipulation Suite[1], ktorá obsahuje zbierku programov (napísaných v jazyku JavaScript) slúžiacich na generovanie, formátovanie a analýzu krátkych DNA sekvencií. V oboch prípadoch je testovanie prevedené na stroji Merlin[2].

## 6.1 Testovanie aplikácie pri použití rôznych vstupných parametrov

**Test spracovania priamo zadanej sekvencie DNA (obr. 6.1)**

V tomto prípade je aplikácia spustená s parametrom $-s$ označujúcim, že vstupná sekvencia pre prevodník $M_1$ je zadaná priamo ako parameter aplikácie. $M_1$ validuje túto sekvenciu, pričom vráti token $(1, 0)$ značiaci validnú DNA sekvenciu. Tú aplikácia vloží na vstup prevodníka $M_2$, ten prevedie proces transkripcie a výstupnú sekvenciu vloží na vstup prevodníka $M_3$. Pri spustení aplikácie bol taktiež zadaný parameter $--table$ označujúci, že užívateľ vyžaduje spolu s výsledkom prekladu zobraziť aj tabuľku genetického kódu.

**Test spracovania priamo zadanej, sekvencie komplementárnej mRNA (obr. 6.2)**

Aplikácia je v tomto prípade opäť spustená s parametrom $-s$ značiacim, že vstupná sekvencia pre prevodník $M_1$ je zadaná priamo ako parameter aplikácie. Avšak bol tiež zadaný parameter $-complement$, ktorý značí, že na vstup prevodníka $M_3$ má byť vložená komplementárna sekvencia k zadanej sekvencii.

Čiže, samotná sekvencia definovaná na vstupe je predaná prevodníku $M_1$, ktorý ju valide(vracia token (2,0), značiaci validnú sekvenciu mRNA). Následne je k tejto sekvencii na základe princípu komplementarity 4.4.2 vytvorená komplementárna sekvencia, ktorá je vložená na vstup prevodníka $M_3$. Výstup tohto prevodníka je zobrazený užívateľovi.

[1] Sequence Manipulation Suite – dostupná na https://www.bioinformatics.org/sms2/index.html
[2] Merlin – https://merlin.fit.vutbr.cz/

```
xpukan01@merlin: ~/IBT$ ./IBTProjMainModule.py -s ACAATGACGGGCCCTTAACAG --table
+-----------------------------------------------+
|           The standard genetic code           |
|-----------------------------------------------|
| TTT - Phe | TCT - Ser | TAT - Tyr | TGT - Cys |
| TTC - Phe | TCC - Ser | TAC - Tyr | TGC - Cys |
| TTA - Leu | TCA - Ser | TAA - STOP| TGA - STOP|
| TTG - Leu | TCG - Ser | TAG - STOP| TGG - Trp |
|-----------------------------------------------|
| CTT - Leu | CCT - Pro | CAT - His | CGT - Arg |
| CTC - Leu | CCC - Pro | CAC - His | CGC - Arg |
| CTA - Leu | CCA - Pro | CAA - Gln | CGA - Arg |
| CTG - Leu | CCG - Pro | CAG - Gln | CGG - Arg |
|-----------------------------------------------|
| ATT - Ile | ACT - Thr | AAT - Asn | AGT - Ser |
| ATC - Ile | ACC - Thr | AAC - Asn | AGC - Ser |
| ATA - Ile | ACA - Thr | AAA - Lys | AGA - ARG |
| ATG - Met | ACG - Thr | AAG - Lys | AGG - ARG |
|-----------------------------------------------|
| GTT - Val | GCT - Ala | GAT - Asp | GGT - Gly |
| GTC - Val | GCC - Ala | GAC - Asp | GGC - Gly |
| GTA - Val | GCA - Ala | GAA - Glu | GGA - Gly |
| GTG - Val | GCG - Ala | GAG - Glu | GGG - Gly |
+-----------------------------------------------+
+-------------------------------------------------------------------------------+
| Input DNA sequence:         ACAATGACGGGCCCTTAACAG                             |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): ACAAUGACGGGCCCUUAACAG                             |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :                 MetThrGlyPro                                      |
+-------------------------------------------------------------------------------+
```

Obr. 6.1: Test spracovania priamo zadanej sekvencie a zobrazenie tabuľky genetického kódu

```
xpukan01@merlin: ~/IBT$ ./IBTProjMainModule.py -s UACGCAAUGUUCAGAAGGAUU -complement
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): AUGCGUUACAAGUCUUCCUAA                             |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :                 MetArgTyrLysSerSer                                |
+-------------------------------------------------------------------------------+
```

Obr. 6.2: Test spracovania priamo zadanej, sekvencie komplementárnej mRNA

**Test spracovania generovanej sekvencie (obr. 6.3)**

Aplikácia je spustená len so vstupným parametrom $--g$, označujúcim, že vstupná sekvencia pre prevodník $M_2$ má byť vygenerovaná. (Avšak nie úplne náhodným spôsobom, ako je aj uvedené v 5.6) Pre túto sekvenciu vždy platí, že je typu DNA, čiže obsahuje len symboly z množiny $\{A, T, C, G\}$. Výsledkom prekladu (výstupom $M_3$) sú štyri proteíny vo formáte sekvencie aminokyselín.

```
xpukan01@merlin: ~/IBT$ ./IBTProjMainModule.py --g
+-------------------------------------------------------------------------------+
| Input DNA sequence:         ATGGCCTAAATGAGATAACTTATCATGCTAGGCAGCTAAATGCTGGCGGC |
|                             ACTCTACGATGTGTAGCATTTACACTCAATTCAATGTAGTAATATTCCCA |
|                             TTATTTACTTAACCAATGTACACTTGAAACACCCAATCACACACTCCATT |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): AUGGCCUAAAUGAGAUAACUUAUCAUGCUAGGCAGCUAAAUGCUGGCGGC |
|                             ACUCUACGAUGUGUAGCAUUUACACUCAAUUCAAUGUAGUAAUAUUCCCA |
|                             UUAUUUACUUAACCAAUGUACACUUGAAACACCCAAUCACACACUCCAUU |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :                 MetAla                                            |
| protein 2 :                 MetArg                                            |
| protein 3 :                 MetLeuGlySer                                      |
| protein 4 :                 MetLeuAlaAlaLeuTyrAspVal                          |
+-------------------------------------------------------------------------------+
```

Obr. 6.3: Test spracovania vygenerovanej sekvencie

**Test spracovania priamo zadanej sekvencie pomocou krokovania (obr. 6.4, 6.5)**

Aplikácia je spustená s parametrami $-s$ – značiaci priamo zadanú sekvenciu, $-steps$ – pre výpis jednotlivých výpočtových krokov a $-manual$ – pre manuálny výpis jednotlivých krokov. Výpočtové kroky $1 - 3$ sú zobrazené na obr. 6.4 a predposledný výpočtový krok 9 spolu s výstupom aplikácie na obr. 6.5.

```
xpukan01@merlin: ~/IBT$ ./IBTProjMainModule.py -s ATGAACTAG -steps -manual

+-------------------------------------------------------------------------------+
                            Translation step  1
+-------------------------------------------------------------------------------+
  Input:                    UGAACUAG#
  Output:
  State:                    q1
  Top of stack:             AS

+-------------------------------------------------------------------------------+
                            Translation step  2
+-------------------------------------------------------------------------------+
  Input:                    GAACUAG#
  Output:
  State:                    q2
  Top of stack:             UA

+-------------------------------------------------------------------------------+
                            Translation step  3
+-------------------------------------------------------------------------------+
  Input:                    AACUAG#
  Output:                   Met
  State:                    q3
  Top of stack:             GU

+-------------------------------------------------------------------------------+
```

Obr. 6.4: Test spracovania priamo zadanej sekvencie pomocou krokovania (kroky 1-9)

```
+-------------------------------------------------------------------------------+
                            Translation step  9
+-------------------------------------------------------------------------------+
  Input:                    #
  Output:                   MetAsn

  State:                    s
  Top of stack:             S$
+-------------------------------------------------------------------------------+
| Input DNA sequence:       ATGAACTAG                                           |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): AUGAACUAG                                         |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :               MetAsn                                              |
+-------------------------------------------------------------------------------+
```

Obr. 6.5: Test spracovania priamo zadanej sekvencie pomocou krokovania (krok 9 a výstup)

## 6.2 Validácia získaných výsledkov

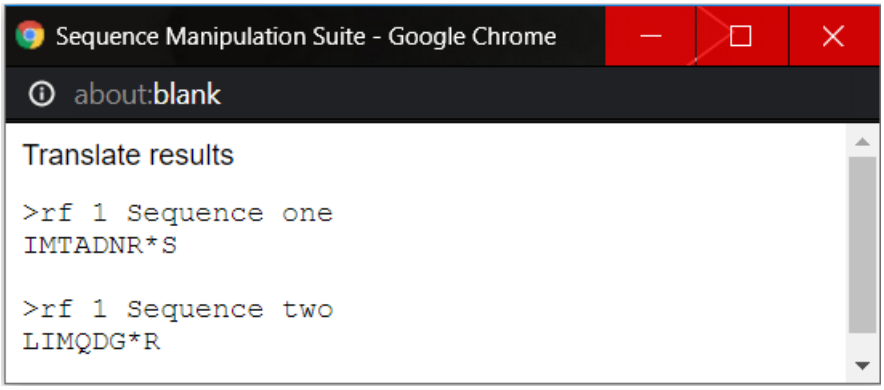
Ako už bolo v úvode tejto časti spomenuté, získané výsledky sa pokúsime validovať porovnaním s výstupom webovej aplikácie Sequence Manipulation Suite[3].

Majme teda výstup našej aplikácie pri zadaní dvoch vstupných sekvencií vo formáte FASTA (4.6) ako súčasť súboru *testFile.fasta* uloženého v adresári *inputFiles*, pričom prvá sekvencia je typu DNA a druhá typu mRNA.

```
xpukan01@merlin: ~/IBT$ ./IBTProjMainModule.py --file inputFiles/testFile.fasta
+-------------------------------------------------------------------------------+
| Input DNA sequence:         ATAATGACCGCCGATAATCGCTGAAGT                       |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): AUAAUGACCGCCGAUAAUCGCUGAAGU                       |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :                 MetThrAlaAspAsnArg                                |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Transcription result(mRNA): CUAAUCAUGCAAGAUGGUUAACGU                         |
+-------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------+
| Translation result:                                                           |
|                                                                               |
| protein 1 :                 MetGlnAspGly                                      |
+-------------------------------------------------------------------------------+
```

Obr. 6.6: Test spracovania dvoch sekvencií zadaných ako obsah súboru

A taktiež majme výstup už vyššie spomenutej webovej aplikácie, pričom vstupné sekvencie zadávame v sekcii *Sequence analysis* a možnosti *Translate*, slúžiacej pre preklad sekvencií DNA (prípadne mRNA) na proteíny.




Obr. 6.7: Test spracovania dvoch sekvencií webovou aplikáciou

[3]Sequence Manipulation Suite – dostupná na https://www.bioinformatics.org/sms2/index.html

Ako je možné vidieť, výstupy sa na prvý pohľad líšia. Je to spôsobené tým, že aplikácia Sequence Manipulation Suite zobrazuje preložené sekvencie spôsobom definovaným tabuľkou 4.2. Taktiež je nutné dodať, že Sequence Manipulation Suite prekladá aj kodóny, ktoré priamo nepatria žiadnemu proteínu. Oproti tomu naša aplikácia takéto kodóny ignoruje.

Zoberme teda prvý výstup aplikácie Sequence Manipulation Suite: IMTADNR*S. Ak pomocou tabuliek 4.2 a 4.3 sekvenciu prepíšeme na nami používané značenie, získame: *IleMetThrAlaAspAsnArg∗Ser*, pričom po odstránení *Ile, Ser*, aminokyselín nepatriacich žiadnemu proteínu a tiež $*$, reprezentujúcej *stop* kodón, získame zhodný proteín ako pomocou našej aplikácie.

Rovnako ak zoberieme druhú získanú sekvenciu: *LIMQDG∗R* a prevedení rovnakých operácií ako s prvou sekvenciou získame: *MetGlnAspGly*. Pri oboch sekvenciách sme teda získali validné hodnoty.

# Kapitola 7

# Záver

V úvode tejto práce som si stanovil cieľ navrhnúť a implementovať aplikáciu, ktorá by pomocou prevodníkov bola schopná spracovávať sekvencie DNA či mRNA. Bolo potrebné dôkladné štúdium vlastností prevodníkov, ale aj molekulárnej biológie – vedy, ktorá sa obsahom pomerne dosť odlišuje od typického učiva študenta informatiky. Avšak nakoniec musím konštatovať, že to bola príjemná a obohacujúca skúsennosť využívať prvky týchto, zdanlivo odlišných vied na vytvorenie niečoho prospešného pre oba svety. Vytvorená aplikácia spĺňa všetky požiadavky definované v úvode tejto práce a je pripravená na použitie. V súčasnom stave je vhodná najmä pre výukové účely, a to buď pre demonštráciu práce konečných alebo zásobníkových prevodníkov alebo pre demonštráciu procesu *proteosyntézy* na stredných alebo základných školách.

Čo sa týka vyhliadok do budúcnosti, rád by som rozšíril funkcionalitu aplikácie o spracovanie aj sekvencií patriacich Eukariotickým bunkám. To znamená schopnosť rozoznať kódujúce segmenty – *exóny* a nekódujúce segmenty – *intróny*, tie následne odstrániť a zvyšnú časť vstupnej sekvencie spracovať. Ďalej by som rád napojil vytvorenú aplikáciu na niektorú z databáz obsahujúcich informácie o jednotlivých bielkovinách a následne vytvoril mechanizmus (pravdepodobne založený na nejakom type prevodníka), ktorý by bol schopný k získanej sekvencii aminokyselín priradiť aj presný názov proteínu.

# Literatúra


[1] Alfred V. Aho, J. D. U. *The theory of parsing, translation and compiling*. 1. vyd. Prentice Hall; Later Printing edition (June 1, 1972), 1972. ISBN 0-13-914556-7.

[2] Balcar, B. *Teorie množin*. 1. vyd. Praha : Academia, 2000. ISBN 80-200-0470-X.

[3] Břížďala, R. J. *E-CHEMBOOK.EU* [online]. 2020. 2020 [cit. 2020-10-03]. Dostupné z: http://e-chembook.eu/nukleove-kyseliny.

[4] Crespi Reghizzi S., M. A. *Formal Languages and Compilation*. 1. vyd. Springer International Publishing AG, 2019. ISBN 9783030048785.

[5] Encyclopedia, W. T. F. *Translace (biologie)* [online]. 2010. 05-04-2020 [cit. 2020-12-05]. Dostupné z: https://cs.wikipedia.org/wiki/Translace_(biologie).

[6] Encyclopedia, W. T. F. *FASTA format* [online]. 2018 [cit. 2020-12-05]. Dostupné z: https://en.wikipedia.org/wiki/FASTA_format.

[7] kol., S. R. . *Terminologie molekulární biologie*. 1. vyd. © prof. RNDr. Stanislav Rosypal, DrSc., 2001. ISBN 80-902562-3-6.

[8] Lesk, A. M. *Bioinformatics*. 3. vyd. OXFORD UNIVERSITY PRESS, 2008. ISBN 978-0-19-920804-3.

[9] Levelt, W. J. M. *An introduction to the theory of formal languages and automata*. 1. vyd. Amsterdam ; Philadelphia : Benjamins, 2008. ISBN 978-90-272-3250-2.

[10] Meduna, A. *Automata and Languages*. 2. vyd. Springer London, 2000. ISBN 9781447105015.

[11] RNDRr. Eduard Kočárek, P. *Genetika*. 2. vyd. NAKLADATELSTVÍ SCIENTA, spol. s.r.o., 2008. ISBN 978-80-86960-36-4.

# Príloha A

# Zdrojové kódy, manuál a dokumentácia

Dostupné na: https://nextcloud.fit.vutbr.cz/s/Fx6GjN4RLsdwbrg.