VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

Fakulta informačních technologií
Faculty of Information Technology

DIPLOMOVÁ PRÁCE
MASTER THESIS

Brno, 2019 Bc. Lubomír Ošmera

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SKÁKAJÍCÍ JAZYKOVÉ MODELY
JUMPING LANGUAGE MODELS

DIPLOMOVÁ PRÁCE
MASTER THESIS

AUTOR PRÁCE — Bc. Lubomír Ošmera
AUTHOR

VEDOUCÍ PRÁCE — Prof. RNDr. Alexander Meduna, CSc.
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Akademický rok 2018/2019

Zadání diplomové práce

Ústav informačních systémů (UIFS)

Student: **Ošmera Lubomír, Bc.**

Program: Informační technologie Obor: Informační systémy

Název: **Skákající jazykové modely**
Jumping Language Models

Kategorie: Teoretická informatika

Zadání:

1. Seznamte se detailně s různými variantami skákajících automatů a gramatik dle instrukcí vedoucího.
2. Zaveďte modifikované verze těchto modelů dle instrukcí vedoucího.
3. Studujte vlastnosti modelů zavedených v bodě 2. Zejména porovnejte jejich vyjadřující sílu se silou základních skákajících automatů.
4. Demonstrujte aplikovatelnost modelů navržených v bodě 2 v souvislosti se syntaktickou analýzou či bioinformatikou. Implementujte tuto aplikaci.
5. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Křivka, Z., Kučera, J. and Meduna, A.: Jumping Pure Grammars. *The Computer Journal*. Oxford: Oxford
- University Press, 2018, roč. 2018, č. 1, s. 1-12. ISSN 0010-4620
- Charvát, L. and Meduna, A.: A Reduction of Finitely Expandable Deep Pushdown Automata. Schedae
- Informaticae. Krakov: 2018, roč. 2017, č. 26, s. 61-68. ISSN 0860-0295
- Charvát, L. and Meduna, A.: Internally Expandable Pushdown Automata and Their Computational Completeness. Romanian Journal of Information Science and Technology (ROMJIST). Bukurest:
- Romanian Academy, Publishing House of the Romanian Academy, 2018, roč. 21, č. 3, s. 232-237. ISSN 1453-8245

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Meduna Alexander, prof. RNDr., CSc.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 31. října 2018

Abstrakt

Cílem této diplomové práce je návrh a výzkum nových verzí skákajících automatů a gramatik. Nové verze jsou zaměřeny primárně na aplikace v bioinformatice – DNA computingu. Práce zkoumá jejich vyjadřovací sílu a další vlastnosti navržených modelů a porovnává je s již existujícími modely teoretické informatiky. Následně demonstruje praktické aplikace, konkrétně aplikace pro detekci aminokyselin a proteinů uvnitř DNA sekvence a provádí porovnání s již existujícími nástroji v DNA computingu, jako jsou například Markovy pravděpodobnostní modely.

Abstract

The main goal of this master thesis is introduction and investigation of extended version of jumping automata and grammars. New versions are primarily focused on bioinformatic applications – DNA computing. This thesis examine their power and other properties of new models and makes comparison with existing computer science models. Then thesis demontrates practical applications, specifically amino acid and protein detections inside DNA sequence and makes comparision with existing tools in DNA computing for example Mark´s probabilistic models.

Klíčová slova

skákající, gramatika, tree controlled, rozptýleným kontextem, DNA, RNA, JTC, teoretická informatika, bezkontextové, kontextové, lineární, regulární, jazyk, syntaktická analýza, derivační strom, skákající převodník, prokaryotní

Keywords

jumping, grammar, tree controlled, scattered, DNA, RNA, JTC, computer science, context-free, context-sensitive, linear, regular, language, syntactic analysis, derivation tree, jumping transducer, prokaryotic

Citace

Ošmera Lubomír: Skákající modely, diplomová práce, Brno, FIT VUT v Brně, 2019

Skákající jazykové modely

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Prof. RNDr. Alexandra Meduny, CSc.
Další informace mi poskytl Ing. Zbyněk Křivka, PhD. a Ing. Tomáš Martínek, PhD.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Lubomír Ošmera
Datum 15. 5. 2019

Poděkování

Rád bych poděkoval panu Prof. RNDr. Alexandru Medunovi, CSc. za jeho špičkové vedení a motivaci v průběhu práce. Děkuji také za doporučenou literaturu, která této práci poskytla obohacení o informace z dřívějších výzkumů, teoretický základ a možné směry dalšího výzkumu.

Obsah

Obsah 1
1 Úvod 2
2 Značení a definice 6
2.1 Definice 6
2.1.1 Základní definice sloužící jako stavební prvky 6
2.1.2 Definice složitější potřebné pro aplikování rozšiřování a omezování již existujících prostředků 12
3 Skákající gramatiky – možné rozšíření klasických gramatik 15
3.1 Semiparalelní skákající gramatiky 17
4 Skákající tree controlled gramatiky 19
4.1 Derivační mód 1 (sekvenční mód) 19
4.2 Ekvivalence SCG a TCG gramatik 22
4.3 Převod TCG do Chomského normální formy 23
4.4 Derivační mód 2 (jumping mód verze 1) 28
4.5 Derivační mód 3 (jumping mód verze 2) 30
4.6 Derivační mód 4 (obecný jumping mód) 32
5 Skákající automaty a převodníky 34
5.1 Konečné převodníky 34
5.2 Skákající automaty 35
5.3 Skákající automaty CJFA 37
5.4 Skákající konečné převodníky 39
6 Aplikace v oblasti bioinformatiky 42
6.1 DNA struktura 42
6.2 Detekce proteinů 45
6.3 Možné vstupy 48
6.4 Využití skákajících gramatik v oblasti DNA 52
7 Závěr 54

1 Úvod

Motivace

V teoretické informatice již postupně ubývá vytváření úplně nových formálních modelů. Naopak ale vzniká více rozšíření již existujících modelů gramatik a automatů o nové vlastnosti. Zajímavým rozšířením bezkontextových gramatik mohou být například paralelní gramatiky. Ty dovolují přepisovat více neterminálních symbolů v jednom kroku. Zároveň však ve srovnání s obecnými gramatikami mohou být jednotlivé přepisované neterminály na různých pozicích uvnitř větné formy.

Jak vypadá chování paralelních gramatik a jaké jsou přednosti tohoto modelu ukazuje jeden zástupce této třídy níže, jedná se o rozptýlené gramatiky (Zdroj: TID přednáška 19-10-2017 [1]):

Definice:

Nechť G = (V, T, P, S) je GG. G je tzv. gramatika s rozptýleným kontextem, SCG, právě tehdy, když všechna pravidla z množiny P jsou ve tvaru $(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n)$, kde $A_1, \ldots, A_n \in V - T$, $x_1, \ldots, x_n \in V^*$.

Jednoduše řečeno nepřepisujeme jednotlivé neterminály postupně, ale paralelně v jednom derivačním kroku jich přepíšeme několik najednou.

Výhoda těchto gramatik spočívá v tom, že máme pravidla v jednoduchém tvaru, ve tvaru pravidel bezkontextové gramatiky. Tímto typem gramatik můžeme přijímat i jazyky z nadmnožiny bezkontextových jazyků. Konkrétně je tento model gramatik ekvivalentní k třídě rekurzivně spočetných jazyků.

Příklad ukazuje vygenerování kontextového jazyka $L = \{a^n\ b^n\ c^n : n \geq 0\}$ za pomoci $SCG\ G = (\{A, C, a, b, c\}, \{a, b, c\}, S, \{(S) \rightarrow (AC), (A, C) \rightarrow (aAb, cC), (A, C) \rightarrow (\varepsilon, \varepsilon)\})$.

Jednotlivé derivační kroky generující jazyk L mohou vypadat např. takto:

$S \Rightarrow AC \Rightarrow aAbcC \Rightarrow aaAbbccC \Rightarrow aaaAbbbcccC \Rightarrow aaabbbccc$.

Kromě rozšíření postupně aplikujeme do již existujících formálních prostředků pro generování jazyků i omezení. V oblasti skákajících gramatik jej může představovat omezení směru skoku, délky, počtu skoků. U bezkontextových gramatik se pak může jednat o omezení gramatiky na sekvenci pevně daného zpracování pravidel, směru zpracování pravidel (volba vždy nejlevějšího nebo nejpravějšího nonterminálu).

Přirozeně očekáváme, že aplikovaná omezení omezí i vyjadřovací sílu přímo na podtřídu jazyků v Chomského hierarchii oproti neupravené gramatice. Paradoxně však aplikovanými omezeními dostáváme ve většině případů větší sílu formálního prostředku. [2]

Příklad omezení bezkontextové gramatiky:

Mějme $BKG\ G = (V, T, P, S)$ s množinou P obsahující následující pravidla:

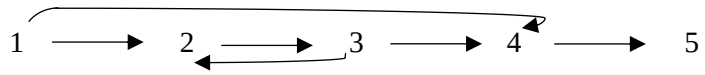
1) $S \rightarrow AB$, 2) $A \rightarrow aA$, 3) $B \rightarrow bBc$, 4) $A \rightarrow a$, 5) $B \rightarrow bc$.

Tato gramatika generuje bezkontextový jazyk $L(G) = \{a^m b^n c^n : m, n \geq 1\}$.

Co se však stane v situaci, kdy omezíme tuto gramatiku tím, že na pevno stanovíme, v jakém sledu budeme aplikovat pravidla z množiny P? V příkladu níže bude pořadí aplikace pravidel určovat orientovaný graf G:

Příklad: Mějme orientovaný graf G = (E, V), kde E představuje množinu hran grafu a V představuje množinu vrcholů grafu G, který ukazuje přesné pořadí aplikování pravidel. Tj. pravidla gramatiky jsou označena čísly 1 – 5 a tato čísla pravidel zároveň tvoří vrcholy v orientovaném grafu G:

Obr. 1 Orientovaný graf G



Pokud budeme provádět derivační kroky podle orientovaného grafu, derivace bude probíhat např. takto: $S \Rightarrow AB\ (1) \Rightarrow aAB\ (2) \Rightarrow aAbBc\ (3) \Rightarrow aaAbBc\ (2) \Rightarrow aaAbbBcc\ (3) \Rightarrow aaabbBcc\ (4) \Rightarrow aaabbbccc\ (5)$.

Generovaný jazyk zmodifikované gramatiky je jazyk kontextový $L = \{a^n b^n c^n : n \geq 1\}$.

Paradoxně jsme omezením původní bezkontextové gramatiky získali větší vyjadřovací sílu. Najednou původní bezkontextová gramatika generuje kontextový jazyk. (přednáška IFJ z 16-12-2018 z [2])

Samozřejmě ale ne vždy omezením formálního prostředku můžeme očekávat jeho větší vyjadřovací sílu. Jako dobrý příklad může posloužit omezování pravé strany pravidel gramatik.

Uvažme bezkontextové gramatiky, které mají pravidla ve tvaru $A \rightarrow u$, kde $A \in V - T$ a $u \in V^*$. Když omezíme pravé strany pravidel bezkontextových gramatik namísto libovolné sekvence terminálů a neterminálů na pevný tvar wK, kde $w \in T^*$ a $K \in V - T$, pak dostáváme lineární gramatiku, která generuje pouze třídu regulárních jazyků.

Modifikace verzí skákajících gramatik

V současné době existují obecné skákající gramatiky a již několik modifikací těchto modelů. U obecných skákajících gramatik je velkou překážkou jejich velká variabilita skoků a z toho vyplývající nedeterminismus při provádění derivačních kroků.

Příkladem může být **Lemma**: Neexistuje žádná CSG ani MONG skákající gramatika, která generuje jazyk $L = \{a\}^*\{b\}^*$ [3]

Překvapivé je, že L je regulární jazyk, avšak právě velká variabilita skoků skákajících gramatik způsobuje, že tento jazyk není žádnou kontextovou skákající gramatikou možné generovat.

Důkaz: Důkaz tohoto tvrzení je proveden kontradikcí. Předpokládáme, že existuje skákající gramatika generující jazyk L.

$S \Rightarrow_j^* a^m b^n$ s m a n delším než pravá strana nejdelšího z pravidel. Vyjádříme derivaci jako $S \Rightarrow_j^* \alpha \Rightarrow_j a^m b^n$.

Nechť α vyjádříme jako $a \ldots aAa \ldots ab^n \Rightarrow_j a^m b^n$ a dále budeme aplikovat pravidlo ve tvaru $A \rightarrow a \ldots a$

Čili tím můžeme získat derivaci v tomto tvaru: $a \ldots aAa \ldots ab^n \Rightarrow_j a \ldots ab^n a \ldots a$.

Tím ale zároveň prohlašujeme, že $S \Rightarrow^* a...ab^n a...a$, což je spor. To je důkaz platnosti tvrzení, že neexistuje žádná kontextová či MONG skákající gramatika, která generuje jazyk $L = \{a\}^*\{b\}^*$.

Už první články o skákajících gramatikách však uvádějí velký potenciál využití ve výpočtech DNA. Je nutno vytvořit modifikace a vhodná omezení k těmto modelům a tím můžeme docílit zajímavých aplikací. A nebo naopak již existující gramatiky rozšířit o možnost skoků . Jedním z takových příkladů může být modifikace gramatik s rozptýleným kontextem na model skákajících gramatik s rozptýleným kontextem detailně popsaným v článku [4].

Jaké bude mít modifikovaný model vlastnosti? Jakou třídu jazyků bude tento model generovat a jaký vztah bude mít tento model k již existujícím modelům teoretické informatiky? Podobné otázky budou postupně v průběhu práce zkoumány.

Nové modifikace gramatik

Na základě získaných poznatků je navržen nový model pro skákající gramatiky. (Jumping tree controlled grammars). Tento model vychází z modelů obecných skákajících gramatik, rozptýlených gramatik a tzv. tree controlled grammars a bude mít tyto hlavní přednosti:

1) Pro každou větu generovanou touto gramatikou dokážeme vytvořit derivační strom. Derivační strom je dobře čitelné grafické vyjádření struktury vět daného jazyka, a navíc aplikovatelný například při syntaktické analýze. Tento strom je prostředek, který se využívá u bezkontextových gramatik.
2) V rámci moderních gramatik vzniklo několik rozšíření klasických gramatik, např. *general jumping grammars, jumping scattered context grammars, parallel grammars*. Jejich vyjadřovací síla je různá, od třídy bezkontextových jazyků až po rekurzivně vyčíslitelné jazyky. Nově navržený model v rámci tohoto semestrálního projektu tzv. jumping tree controlled gramatiky (JTC) jsou rozšířením skákajících gramatik. Dodávají omezení taková, abychom byly schopni přijmout konkrétní typ.
3) Tyto gramatiky jsou založeny na bezkontextových gramatikách a zachovávají jejich jednoduchost.
4) JTC gramatiky mají větší vyjadřovací sílu než klasické bezkontextové gramatiky. Dokážou generovat i jazyky z třídy kontextových jazyků a rekurzivně spočetné jazyky.
5) V případě skákajících gramatik dokážeme tímto formálním modelem dodat omezení, které odstraní nedeterminismus skoků. Díky tomu budeme moci tyto moderní rozšíření klasických gramatik velmi dobře využít například v bioinformatice (DNA) či syntaktické analýze. Nebo také v rámci distribuovaných systému při zpracování informací. Jako příklad si představme proces *p* v rámci distribuovaného systému, který jde zpracovávat úlohu *a*. Tato úloha může být na libovolném místě v rámci celého balíčku událostí. Proces ji v daném okamžiku vyhodnotí, smaže a napíše odpověď. Odpověď však nezapíše na původní místo, protože už začínají pracovat s úlohami ostatní procesy. Tj. svůj kousek informace vloží na libovolné místo.

Modifikace automatů

Obdobným způsobem, jakým vytváříme modifikace gramatik, provádíme také modifikace konečných automatů. Konečný automat má vlastnost, že zpracovává vstup kontinuálně, čte vstupní pásku symbol po symbolu. Co se stane, pokud získáme konfiguraci, ve které nemáme žádné pravidlo pro aktuální symbol na vstupní pásce a aktuální stav? Práce automatu končí a řetězec není přijat. Co kdyby však jsme umožnili v takovém případě automatu přeskočit symbol, pro který v tuto chvíli nemá pravidlo. Nebo z nějakého důvodu potřebujeme začít číst pásku z jiného místa než od začátku řetězc - takových příkladů najdeme mnoho v oblasti zpracování DNA řetězce. K tomuto účelu máme k dispozici Turingovy stroje, které dokáží vstupní pásku libovolně prodlužovat a libovolným způsobem se po ní pohybovat. Co když však potřebujeme zachovat jednoduchost konečných automatů se současným přidání dalších schopností? Chceme zachovat tvar konečných automatů s možností pohybu po pásce, který není „omezen“ nutností číst ji kontinuálně. Pak se dostáváme do zajímavé množiny formálních prostředků zvaných skákající automaty. Uvažme následující příklad:

Mějme skákající konečný automat $M = (\{s, q\}, \{a, b\}, R, s, \{s\})$, kde množina $R = \{sa \rightarrow q, qb \rightarrow s\}$. Pokud by se jednalo o klasický konečný automat s nemožností skoků, pak bychom tímto automatem dokázali přijímat jazyk $L(M) = \{ab\}^*$. Pokud povolíme užít zmíněné skoky, přijímáme řetězce se znaky a, b v jakémkoliv pořadí se současným zachováním jejich shodného počtu, tedy $L(M) = \{w = \{a,b\}^* \mid occur(a, w) = occur(b, w)\}$.

Když se vrátíme k příkladu procesu p v distribuovaném systému, kde proces p postupně potřebujeme z různých míst vstupní data a zapisovat jej na výstup nebo zpracování DNA řetězce, tak se dostáváme ke kapitole převodníků, které modely automatů dále rozšiřují o výstupní pásku.

Co když skoky automatu omezíme na skoky určité délky nebo jejich maximální délku? Či zavedeme jiné úpravy k docílení většího determinismu při provádění skoků?

V této práci budou prozkoumány zajímavé modifikace těchto modelů z hlediska vyjadřovací síly. Dále ukázány možné aplikace v oblasti bioinformatiky. Již článek o skákajících automatech z roku 2015 otevřel zajímavé nástiny aplikace v DNA. Nyní tento pohled rozšíříme. Popíšeme problematiku bioinformatiky v širším kontextu, následně jednotlivé části problematiky vložíme do kontextu formálních modelů. Závěr práce bude obsahovat zamyšlení, kterým dalším rozšířením inspirovaným doposud používanými modely v bioinformatice můžeme obohatit formální modely tvořené v této práci tak, aby v praxi dosahovali zajímavých výsledků.

2 Značení a definice

Tato publikace bude rozdělena do 3 hlavních částí: V první části shrneme současné skákající jazykové modely, jejich vlastnosti a limity a zařadíme je do kontextu stávajících modelů teoretické informatiky. Ve druhé části na tyto poznatky navážeme a vytvoříme modifikace modelů, jejichž vlastnosti mají opodstatnění v různých vědních disciplínách, především v oblasti bioinformatiky. V poslední části pak budou skákající modely a jejich využití demonstrovány na specifických úlohách souvisejícími s DNA zpracováním.

Použité značení

Pro lepší orientaci v textu budou zavedena tato pravidla:

- Veškeré zkratky formálních prostředků: řetězce, symboly a součásti automatů nebo gramatik (stavy, neterminály apod.) budou vyznačeny kurzívou. (s výjimkou definic, nadpisů a tabulek)
- Neterminály budou značeny velkým písmenem, kromě některých výjimek při ukázkách aplikace v bioinformatice.
- Řetězce, symboly abecedy a terminály budou značeny malými písmeny.

2.1 Definice

Pro vytváření nových modelů a pro popis jazyků je důležité zavést základní pojmy a definice, na kterých následně budou postaveny složitější definice popisující výše zmíněné nové formální modely. Základní definice jsou převzaty ze zdroje [2], [5], [6] a [7]. Složitější definice související s moderními gramatikami, omezeními a zejména se skákajícími modely jsou přejaty z [3] a [8].

2.1.1 Základní definice sloužící jako stavební prvky

Definice 1 (Abeceda)

Abeceda je konečná, neprázdná množina elementů, které nazýváme symboly.

Definice 2 (Řetězec)

Nechť Σ je abeceda. ε značí řetězec prázdný, tj. takový řetězec, který neobsahuje žádný symbol.
Nechť x je řetězec nad abecedou Σ a platí, že $a \in \Sigma$, pak xa nazýváme řetězcem nad abecedou Σ.

Definice 3 (Délka řetězce)

Nechť x je řetězec nad abecedou Σ, pak délka řetězce x, značena jako $|x|$, je definována takto:

(1) $x = \varepsilon$: $|x| = 0$,

(2) $x = a_1, a_2, \ldots, a_n$: $|x| = n$ pro $n \geq 1$ a $a_i \in \Sigma$ pro všechna $i = 1, \ldots, n$.

Definice 4 (Počet výskytů symbolu v řetězci)

Nechť Σ je abeceda. $w \in \Sigma^*$, $a \in \Sigma$, pak occur (a, w) značí počet výskytů symbolu a uvnitř řetězce w.

Definice 5 (Konkatenace řetězce)

Nechť x a y značí řetězec nad abecedou Σ. Konkatenace x a y, označena jako xy, je řetězec získaný přidáním řetězce y k řetězci x. Dále platí, že pro každé $w \in \Sigma^*$, $w\varepsilon = \varepsilon w = w$.

Definice 6 (Mocnina řetězce)

Nechť x představuje řetězec nad abecedou Σ. Pro $i \geq 0$, i-tá mocnina řetězce x, značeno x^i, je definována:

(1) $x^0 = \varepsilon$,

(2) $x^i = xx^{i-1}$, pro $i \geq 1$.

Definice 7 (Reverzace řetězce)

Mějme x jako řetězec nad abecedou Σ. Reverzace řetězce x, reversal(x), je definována:

(1) Pro $x = \varepsilon$, $reversal(\varepsilon) = \varepsilon$

(2) Pro $x = a_1 \dots a_n$, $reversal(a_1 \dots a_n) = a_n \dots a_1 \mid i = 1,\dots,n$ a $a_n \in \Sigma$.

Definice 8 (Jazyk)

Nechť Σ je abeceda.

Σ^* označuje množinu všech řetězců nad abecedou Σ, které říkáme univerzální jazyk.

Pokud L je podmnožinou Σ^*, pak říkáme, že L je jazykem nad abecedou Σ.

Pokud L je konečná množina, pak říkáme, že jazyk je konečný, jinak je jazyk nekonečný.

Definice 9 (Konkatenace jazyka)

Konkatenace jazyků L_1 a L_2, $L_1 L_2$, je definována:

$L_1 L_2 = \{x_1 x_2 \mid x_1 \in L_1, x_2 \in L_2\}$.

Definice 10 (Mocnina jazyka)

Nechť L je jazyk nad abecedou Σ. Pro $i \geq 0$, i-tá mocnina jazyka L, L^i, je definována:

$L^i = \{\varepsilon\}$ pro $i = 0$, jinak $L^i = LL^{i-1}$.

Definice 11 (Reverzace jazyka)

Reverzace jazyka, reversal(L), je definována jako:

$reversal(L) = \{reversal(x) \mid x \in L\}$.

Definice 12 (Operace nad jazyky)

Mějme jazyky L_1 a L_2, které představují libovolné dva jazyky nad abecedou Σ.

Průnik jazyků, $L_1 \cap L_2$, definujeme jako:

$L_1 \cap L_2 = \{x \mid x \in L_1 \text{ a } x \in L_2\}$.

Sjednocení jazyků, $L_1 \cup L_2$, definujeme jako:

$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ nebo } x \in L_2\}$.

Rozdíl jazyků, $L_1 - L_2$, je definován:

$L_1 - L_2 = \{x \mid x \in L_1 \text{ a } x \notin L_2\}$.

Definice 13 (Doplněk jazyka)

$L' = \{x \in \Sigma^*, x \notin L\}$

Definice 14 (Iterace jazyka)

Nechť L je jazyk nad abecedou Σ. V teorii formálních jazyků rozlišujeme dva typy iterace:

Iterace jazyka, značena L^* je definována předpisem $L^* = \bigcup_{i=0}^{\infty} L^i$.

Pozitivní iterace jazyka, značena L^+ je definována předpisem $L^+ = \bigcup_{i=1}^{\infty} L^i$.

Definice 15 (Substituce)

Mějme abecedy Σ a Γ. Substituce Φ je funkce, pro kterou platí:

(1) $\Phi(\varepsilon) = \{\varepsilon\}$,

(2) $\Phi(xy) = \Phi(x)\Phi(y)$ pro všechny řetězce $x, y \in \Sigma^*$.

Definice 16 (Morfismus)

Morfismus je speciální případ substituce, kde každý jazyk obsahuje pouze jediný řetězec.

Definice: Nechť Σ a Γ jsou abecedy. Funkce $\phi: \Sigma^* \rightarrow \Gamma^*$ se nazývá morfismus právě tehdy, pokud platí:

(1) $\phi(\varepsilon) = \varepsilon$

(2) $\phi(xy) = \phi(x)\,\phi(y)$ pro všechny řetězce $x, y \in \Sigma^*$.

Definice 17 (Větná forma)

Nechť G = (N, Σ, P, S) je gramatika. Řetězec $\alpha \in (N \cup \Sigma)^*$ nazýváme větnou formou, jestliže platí $S \Rightarrow^* \alpha$, tj. řetězec α je generovatelný z výchozího symbolu S.

Definice 18 (Věta)

Větná forma, která obsahuje pouze terminální symboly, se nazývá věta.

Definice 19 (Obecná gramatika)

Obecná gramatika (zkráceně GG) je čtveřice, G = (V, T, P, S), kde V je množina terminálních a neterminálních symbolů, $T \subseteq V$ je množina terminálních symbolů, pro jednoduchost zavedeme $N = V - T$, které představuje množinu neterminálních symbolů, $S \in N$, $P \subseteq V^* V V^* \times V^*$ a nazýváme jej množinou pravidel. Častěji pro lepší přehlednost místo relačního zápisu pravidel tvaru $(x, y) \in P$ nebo xPy, zapisujeme $x \rightarrow y$.

Definice 20 (Monotónní gramatika)

Monotónní gramatika (zkráceně MONG) je obecná gramatika GG G = (V, T, P, S) taková, kde pro každé pravidlo $x \rightarrow y$ platí, že $|x| \leq |y|$.

Definice 21 (Kontextová gramatika)

Kontextová gramatika (zkráceně CSG) je obecná gramatika GG G = (V, T, P, S) taková, kde pro každé pravidlo $x \rightarrow y$ platí: $x = u_1Au_2$, $y = u_1tu_2$, kde $u_1, u_2 \in V^*$, $A \in N$, $t \in V^+$.

Definice 22 (Bezkontextová gramatika)

Bezkontextová gramatika (zkráceně CFG) je obecná gramatika GG G = (V, T, P, S) taková, kde každé pravidlo je ve tvaru $A \rightarrow y$, $A \in N$, $y \in V^*$.

Definice 23 (Lineární gramatika)

Lineární gramatika (zkráceně LG) je obecná gramatika GG G = (V, T, P, S) taková, kde každé pravidlo je ve tvaru $A \rightarrow y$ nebo $A \rightarrow xBy$, $A, B \in N$, $x, y \in T^*$.

Definice 24 (Pravá lineární gramatika)

Pravá lineární gramatika (zkráceně RLG) je obecná gramatika GG G = (V, T, P, S) taková, kde každé pravidlo je ve tvaru $A \rightarrow y$ nebo $A \rightarrow xB$, $A, B \in N$, $x, y \in T^*$.

Definice 25 (Regulární gramatika)

Regulární gramatika (zkráceně REG) je obecná gramatika GG G = (V, T, P, S) taková, kde každé pravidlo je ve tvaru $A \rightarrow a$ nebo $A \rightarrow aB$, $A, B \in N$, $a \in T$.

Definice 26 (Derivace)

Nechť G = (V, T, P, S) je GG. Nechť $u, v \in V^*$ a pravidlo $p: x \rightarrow y \in P$. Pak uxv přímo derivuje uyv za použití pravidla p v G, zapsáno $uxv \Rightarrow uyv[p]$ nebo zjednodušeně $uxv \Rightarrow uyv$.

Definice 27 (Nejlevější a nejpravější derivace)

Nechť G = (V, T, P, S) je GG.

Nechť $u \in T^*$, $v \in V^*$ a $p: x \rightarrow y \in P$. Pak uxv přímo derivuje uyv za použití nejlevější derivace, zapsáno jako $uxv \ {}_{lm}\Rightarrow uyv$.

Nechť $u \in V^*$, $v \in T^*$ a $p: x \rightarrow y \in P$. Pak uxv přímo derivuje uyv za použití nejpravější derivace, zapsáno jako $uxv \ {}_{rm}\Rightarrow uyv$

Definice 28 (Sekvence derivačních kroků)

Nechť G = (V, T, P, S) je GG. Nechť $u \in V^*$. G provede nula derivačních kroků z u do u, $u \Rightarrow^0 u[\varepsilon]$ nebo zkráceně zapsáno $u \Rightarrow^0 u$.

Nechť $u_0, \ldots, u_n \in V^*$, G provede n derivačních kroků aplikováním posloupnosti pravidel δ, zapsáno jako $u_0 \Rightarrow^n u_n [\delta]$. Pro $n \geq 1$ posloupnost kroků zkráceně zapisujeme jako $u_0 \Rightarrow^+ u_n [\delta]$. Pokud máme $u_0 \Rightarrow^n u_n [\delta]$ pro $n \geq 0$, pak tuto posloupnost derivací zkráceně zapisujeme jako $u_0 \Rightarrow^* u_n [\delta]$.

Definice 29 (Jazyk generovaný gramatikou)
Mějme $G = (V, T, P, S)$ je GG, pak jazyk generovaný gramatikou G je definován jako:
$L(G, r) = \{ x \mid x \in T^*, S \Rightarrow^* x\}$.

Definice 30 (Pravidlový strom)
Pravidlový strom je acyklický graf, který znázorňuje pravidla gramatiky s tím, že levá strana pravidla představuje kořen stromu a pravá strana pravidla syny tohoto stromu.

Definice 31 (Derivační strom)
Derivační strom je takový strom, v němž každý elementární strom (elementární strom představuje úsek otec + jeho synové) je jedním z pravidlových stromů příslušné gramatiky G.

Definice 32 (Úroveň derivačního stromu)
Říkáme, že symbol $x \in V$ je v n-té úrovni derivačního stromu, pokud nejkratší cesta v pravidlovém stromě mezi uzlem startujícího neterminálu S a uzlu x je délky n.

Definice 33 (Nejednoznačná gramatika)
Pokud existuje řetězec $x \in L(G)$ s více jak jedním derivačním stromem, potom G je nejednoznačná. V opačném případě G je jednoznačná.

Definice 34 (Třídy jazyků a Chomského hierarchie)
$\mathscr{L}_0$ nazýváme třídou jazyků rekurzivně spočetných
$\mathscr{L}_1$ nazýváme třídou jazyků kontextových
$\mathscr{L}_2$ nazýváme třídou jazyků bezkontextových
$\mathscr{L}_3$ nazýváme třídou jazyků regulárních
Dle Chomského hierarchie formálních jazyků platí následující tvrzení, o které se často později v textu budeme opírat a vracet se k němu:
$\mathscr{L}_3 \subseteq \mathscr{L}_2 \subseteq \mathscr{L}_1 \subseteq \mathscr{L}_0$
Toto tvrzení vyplývá z faktu, že každý jazyk typu 3 je zároveň jazykem typu 2, každý jazyk typu 2 je zároveň jazykem typu 1, každý jazyk typu 1 je zároveň jazykem typu 0.

Definice 35 V textu bude hojně užíváno rozšířené značení tříd jazyků doplněné také o derivační módy skoků, viz definice. (Třídy jazyků generovanými gramatikami)

Definice 36 (Konečný automat)
Konečný automat, KA zkráceně, je pětice $M = (Q, \Sigma, R, s, F)$:
Q je končená množina stavů

Σ je vstupní abeceda

$R \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow Q$, pravidla v relačním zápisu $(pa, q) \in R$ zapisujeme zjednodušeně jako $pa \rightarrow q$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$

$s \in Q$ je počáteční stav, $F \subseteq Q$ nazýváme množinou koncových stavů.

Definice 37 (Konfigurace a přechod KA)

Mějme KA $M = (Q, \Sigma, \Gamma, R, s, F)$. Konfigurace M je řetězec $Q\,\Sigma^*$.

Mějme konfigurace $\chi = pax$ a $\chi' = qx$, $p, q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in \Sigma^*$ a $p: (pa, q) \in R$. Pak M může provést přechod $\chi \vdash \chi'[p]$, nebo zkráceně zapsáno $\chi \vdash \chi'$.

Sekvence přechodů:

$\chi \vdash^n \chi$ značí, že M provede právě n přechodů. Pro $n \geq 1$ se jedná o tranzitivní uzávěr $\chi \vdash^+ \chi$, pro $n \geq 0$ se jedná o reflexivní uzávěr.

Definice 38 (Přijímaný jazyk KA)

Mějme KA $M = (Q, \Sigma, \Gamma, R, s, F)$. Jazyk přijímaný M, L(M), je definován:

$L(M) = \{w: w \in \Sigma^*, sw \vdash^* f, f \in F\}$.

Definice 39 (Turingův stroj)

Turingův stroj, zkráceně TS, je šestice $M = (Q, \Sigma, \Gamma, R, s, F)$:

Q, Σ, s, F jdou definovány totožně jako v případě konečného automatu. (Konečný automat)

Γ představuje tzv. páskovou abecedu obsahující navíc symbol „blank“, zapisovaný jako Δ, $\Delta \in \Gamma$, $\Delta \notin \Sigma$, $\Sigma \subseteq \Gamma$.

$R \subseteq (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{S, R, L\}$, pravidla v relačním zápisu $(pa, qbt) \in R$ zapisujeme zjednodušeně jako $pa \rightarrow qbt$, $p, q \in Q$, $a, b \in \Gamma$, $t \in \{S, R, L\}$.

Definice 40 (Konfigurace a přechod TS)

Mějme TS $M = (Q, \Sigma, \Gamma, R, s, F)$. Konfigurace M je řetězec $\Gamma^* Q\,\Gamma^*$.

Nechť χ a χ' jsou dvě konfigurace TS M. Pak M může provést přechod $\chi \vdash \chi$, konkrétně jeden ze tří typů přechodů:

1) Stacionární: $\chi \vdash_S \chi'$ pro $\chi = xpay$, $\chi' = xqby$, $q, p \in Q$, $a, b, x \in \Gamma^*$, $y \in \Gamma^* (\Gamma - \{\Delta\}) \cup \{\Delta\}$, $(pa, qbS) \in R$.
2) Pravý přechod: $\chi \vdash_R \chi'$, $\chi = xpay$, $(pa, qbR) \in R$, $\chi' = xbqy$, $y \neq \varepsilon$, popř. $\chi' = xbq\Delta$ pro $y = \varepsilon$.
3) Levý přechod: $\chi \vdash_L \chi'$, $\chi = xcpay$, $(pa, qbL) \in R$, $\chi' = xqcby$, $y \neq \varepsilon$ nebo $b \neq \Delta$, popř. $\chi = xcpa$ $\chi' = xqc$.

Stejně jako v případě konečných automatů máme k dispozici rozšíření ⊢ na $\vdash^n$, transitivní uzávěr $\vdash^+$ a reflexivní uzávěr $\vdash^*$.

Definice 41 (Jazyk přijímaný TS)

Mějme TS M = (Q, Σ, Γ, R, s, F). Jazyk přijímaný M, L(M), je definován:

$L(M) = \{w: w \in \Sigma^*, sw \vdash^* xfy\} \cup \{\varepsilon: s\Delta \vdash^* xfy\}, x, y \in \Gamma^*, f \in F\}$.

2.1.2 Definice složitější potřebné pro aplikování rozšiřování a omezování již existujících prostředků

Definice 42 (Skákající gramatika)

Skákající gramatika (zkr. JGG – Jumping general grammar) je obecná gramatika taková, která může při generování jazyka provádět derivační kroky čtyř módů: sequential step, left jump, right jump, jump (Derivační módy).

Definice 43 (Derivační módy)

$u \Rightarrow_s v$ v G nazýváme **sequential step** tehdy a jen tehdy, když platí $x \rightarrow y \in P$, $w, t, z \in V^*$ takové, že $u = wxz$ a $v = wyz$

$u \Rightarrow_{lj} v$ v G nazýváme **left jump** tehdy a jen tehdy, když platí $x \rightarrow y \in P$, $w, t, z \in V^*$ takové, že $u = wtxz$ a $v = wytz$

$u \Rightarrow_{rj} v$ v G nazýváme **right jump** tehdy a jen tehdy, když platí $x \rightarrow y \in P$, $w, t, z \in V^*$ takové, že $u = wxtz$ a $v = wtyz$

$u \Rightarrow_j v$ v G nazýváme **jump** tehdy a jen tehdy, když může být proveden $u \Rightarrow_{lj} v$ nebo $u \Rightarrow_{rj} v$

Definice 44 (Tree controlled grammars)

Skákající stromem kontrolované gramatiky (Tree Controlled grammars, TCG zkráceně) je uspořádaná dvojice (G, R), kde G = (V, T, P, S) je definována totožně jako CFG (Bezkontextová gramatika), $R \subseteq (N \cup T)^*$ je regulární množina.

Definice 45 (Gramatiky s rozptýleným kontextem)

Nechť G = (V, T, P, S) je GG. G je tzv. gramatika s rozptýleným kontextem, SCG, právě tehdy, když všechna pravidla z množiny P jsou ve tvaru

$(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n)$, kde $A_1, \ldots, A_n \in N$, $x_1, \ldots, x_n \in V^*$

Definice 46 (Třídy jazyků generovanými gramatikami)

Nechť $L(G, \Rightarrow_s) = \{x \in T^* \mid S \Rightarrow_s^* x\}$ je jazyk generovaný gramatikou G užitím $\Rightarrow_s$. Pro libovolné $X \subseteq \Gamma_{GG}$, kde Γ_{GG} značí množinu obecných gramatik mějme $\mathscr{L}(X, \Rightarrow_s) = \{L(G, \Rightarrow_s) \mid G \in X\}$.

Nechť $L(G, \Rightarrow_j) = \{x \in T^* \mid S \Rightarrow_j^* x\}$ je jazyk generovaný gramatikou G užitím $\Rightarrow_j$. $\mathscr{L}(X, \Rightarrow_j) = \{L(G, \Rightarrow_j) \mid G \in X\}$.

Mějme $\Gamma_{GG}, \Gamma_{MONG}, \Gamma_{CSG}, \Gamma_{CFG}, \Gamma_{LG}, \Gamma_{REG}$, značící v pořadí množinu všech gramatik GG, MONG, CSG, CFG, LG, REG.

Pak platí, že:

$\mathscr{L}(\Gamma_{GG}, {}_s\Rightarrow)$ nazýváme třídou jazyků rekurzivně spočetných, zkráceně **RE**.

$\mathscr{L}(\Gamma_{MONG}, {}_s\Rightarrow)$ a $\mathscr{L}(\Gamma_{CSG}, {}_s\Rightarrow)$ nazýváme třídou jazyků kontextových, zkráceně **CS**.

$\mathscr{L}(\Gamma_{CFG}, {}_s\Rightarrow)$ nazýváme třídou jazyků bezkontextových, zkráceně **CF**.

$\mathscr{L}(\Gamma_{LG}, {}_s\Rightarrow)$ nazýváme třídou jazyků lineárních, zkráceně **LIN**.

$\mathscr{L}(\Gamma_{REG}, {}_s\Rightarrow)$ nazýváme třídou jazyků rekurzivně spočetných, zkráceně **RG**.

$\mathscr{L}(\Gamma_{SCG}, {}_s\Rightarrow)$ nazýváme třídou jazyků generovaných SCG gramatikami, zkráceně **SC**.

$\mathscr{L}(\Gamma_{TCG}, {}_s\Rightarrow)$ nazýváme třídou jazyků generovaných TCG gramatikami, zkráceně **TC**.

$\mathscr{L}(\Gamma_{JTC}, {}_n\Rightarrow)$ nazýváme třídou jazyků generovaných JTC gramatikami v módu n, zkráceně **JC**.

$\mathscr{L}(\Gamma_{GJFA})$ nazýváme třídou jazyků přijímaných GJFA automaty, zkráceně **GJF**.

$\mathscr{L}(\Gamma_{JFA})$ nazýváme třídou jazyků přijímaných JFA automaty, zkráceně **JF**.

Podle Chomského platí následující:

RE ⊆ CS ⊆ CF ⊆ LIN ⊆ RG a rovněž **RE ⊂ CS ⊂ CF ⊂ LIN ⊂ RG.**

Definice 47 (Obecné skákající automaty)

Obecný skákající automat, zkráceně GJFA, je pětice, M = (Q, Σ, R, s, F), kde symboly Q, Σ, s, F jsou definovány shodně jako v případě konečných automatů. (Konečný automat)

$R \subseteq Q \times \Sigma^* \rightarrow Q$, pravidla v relačním zápisu $(py, q) \in R$ zapisujeme zjednodušeně jako $py \rightarrow q$, $p, q \in Q$, $y \in \Sigma^*$.

Definice 48 (Konfigurace a přechod u GJFA)

Mějme GJFA M = (Q, Σ, R, s, F), konfigurace M je řetězec $\Sigma^*Q\,\Sigma^*$.

Nechť x. z, x´, z´ ∈ Σ a zároveň platí xz = x´z´, $py \rightarrow q \in R$, pak M provede skok z xpyz do x´qz´, zapsáno jako: xpyz ${}_j|\!-$ x´qz´. Stejně jako v případě konečných automatů máme k dispozici rozšíření ${}_j|\!-$ na ${}_j|\!-^n$, transitivní uzávěr ${}_j|\!-^+$ a reflexivní uzávěr ${}_j|\!-^*$.

Definice 49 (Jazyk přijímaný GJFA)

Mějme GJFA M = (Q, Σ, R, s, F), jazyk přijímaný GJFA M, označený jako L(M), je definován:

$L(M) = \{uv \mid u, v \in \Sigma^*, usv\ {}_j|\!-^*\ f, f \in F\}$.

Definice 50 (Speciální typy skákajících konečných automatů)

Nechť M = (Q, Σ, R, s, F) je GJFA. M je skákající automat stupně n, kde $n \geq 0$, pokud pro každé pravidlo $py \rightarrow q \in R$ platí, že $|y| \leq n$. M je skákajícím automatem stupně 1, zkráceně JFA, pokud $|y| \leq 1$.

Definice 51 (Konečný převodník)

Konečný převodník je pětice M = (Q, Σ, R, s, F)

Q – konečná množina stavů

Σ – abeceda, u které platí že $Q \cap \Sigma$ = blank, $\Sigma = \Sigma_I \cup \Sigma_O$, Σ_I představuje abecedu vstupních symbolů a Σ_O je abeceda výstupních symbolů

$R \subseteq Q\ (\Sigma_I \cup \{\varepsilon\}) \times Q\ \Sigma_O^*$, tvar pravidla $(pa, qz) \in R$ zapisujeme častěji ve tvaru $pa \rightarrow qz$.

$s \in Q$ je startující stav a $F \subseteq Q$ je množina konečných stavů

Definice 52 (Vstupní konečný automat u převodníku)

Nechť $M = (Q, \Sigma, R, s, F)$ je konečný převodník. Vstupní konečný automat M_I náležící k M definujeme jako:

$M_I = (Q, \Sigma_I, R_I, s, F)$, kde Σ_I je vstupní abeceda převodníku M, $R_I = \{pa \rightarrow q: pa \rightarrow qx \in R, x \in \Sigma_O^*\}$

Definice 53 (Přechod a konfigurace u převodníku)

Nechť $M = (Q, \Sigma, R, s, F)$ je konečný převodník a $M_I = (Q, \Sigma_I, R_I, s, F)$ vstupní automat M.

Konfigurace převodníku M, χ, je řetězec $\chi = \chi_I \mid y$, kde χ_I je vstupní konfigurace, tedy konfigurace M_I, $\mid$ je speciální symbol oddělující vstupní pásku od výstupní, $\mid \notin \Sigma$ a $y \in \Sigma_o^*$.

Nechť $\chi = \chi_I \mid y$ a $\chi' = \chi_I' \mid yz$ jsou dvě konfigurace převodníku M, a nechť $r: qa \rightarrow pz \in R$, pak M udělá přechod z χ do χ', zapsaný jako $\chi \vdash \chi'[qa \rightarrow pz]$, zkr. $\chi \vdash \chi'$, pokud M_I může provést přechod $\chi \vdash \chi'[qa \rightarrow p]$.

Transitivní a reflexivní uzávěr u přechodů převodníku je definován analogicky jako v případě konečného automatu.

Definice 54 (Translace u převodníku)

Nechť $M = (Q, \Sigma, R, s, F)$ je konečný převodník, $x \in \Sigma_I^*$ a $y \in \Sigma_o^*$. M překládá x na y tehdy a jen tehdy když $sx \mid \vdash^* f\mid y$, kde $f \in F$. Translace definovaná převodníkem M, T(M), je definována:

$T(M) = \{(x, y): x \in \Sigma_I^*, y \in \Sigma_o^*, M \text{ překládá } x \text{ na } y\}$

3 Skákající gramatiky – možné rozšíření klasických gramatik

Všechny typy doposud užívaných gramatik (obecné, kontextové, bezkontextové, lineární, regulární) mají jednu společnou vlastnost. V každém derivačním kroku dochází k aplikaci pravidla stejným způsobem. Tj. levá strana aplikovaného pravidla se nahradí obsahem pravé strany pravidla uvnitř větné formy přesně na stejné pozici. U skákajících gramatik je toto chování odlišné. Definuje jej algoritmus I.

Algoritmus I. Derivace skákající gramatiky

Nechť t = uvw a máme pravidlo gramatiky $x \rightarrow y$, pak G provede derivační krok způsobem [3]:

1. najdi výskyt symbolu x uvnitř řetězce t
2. vymaž x z řetězce t
3. vlož symbol y na libovolnou pozici uvnitř řetězce t

Příklad 3.1: Mějme pravidlo $A \rightarrow y$ a větnou formu $uvwAxz$. Nejdříve připomeňme, jak by vypadala aplikace pravidla v případě obecných gramatik. Po aplikaci pravidla se neterminál *A* přepíše na *y*. Řetězec *y* bude přesně na stejné pozici, jako byl přepisovaný neterminál *A* a zbytek větné formy zůstane zachovaný ve stejné podobě, jako před aplikací pravidla, tedy $uvwyxz$.

Jak by vypadala aplikace pravidla v případě skákajícího typu gramatiky? Budeme postupovat podle algoritmu I. Opět by proběhlo přepsání *A* na *y*, ovšem tentokrát by *y* nebylo nutně na stejné pozici uvnitř větné formy, jako byla pozice přepisovaného *A*. Jeho přesná pozice by nyní závisela na typu derivačního kroku, který by se provedl při aplikaci pravidla (Definice 43). Konkrétně například při provedení right jump by pak větná forma vypadala následovně: $uvwxyz$, tj. pozice *y* je odlišná od pozice *A* uvnitř větné formy, *y* se vložilo vpravo od pozice přepisovaného *A*.

Příklad 3.2: Mějme gramatiku $G_a = (\{A, S\}, \{a, b\}, \{S \rightarrow aA, A \rightarrow bA, A \rightarrow \varepsilon\}, S)$. Dle chomského hierarchie se jedná o regulární gramatiku.

Jak budou vypadat derivační kroky této regulární gramatiky a jaký jazyk tato gramatika generuje?

$S \Rightarrow aA \Rightarrow abA \Rightarrow abbA \Rightarrow abbbA \ldots \Rightarrow ab^n$.

$L(G_a) = \{ab^n : n \geq 0\}$.

Avšak pokud by se jednalo o gramatiku skákající, která by v průběhu generování jazyka užila libovolného z jump módů, pak by derivační kroky mohly vypadat např. takto:

$S \Rightarrow aA \,{}_j\!\Rightarrow abA \,{}_j\!\Rightarrow bAab \,{}_j\!\Rightarrow bbAab \,{}_j\!\Rightarrow bbab$.

Úprava gramatiky *Ga, Ga´,* která může provádět při generování jazyka skoky, generuje odlišný jazyk, a sice

$L(G_a', {}_j\!\Rightarrow) = \{b^* a\, b^*\}$.

Jak vidíme v uvedeném příkladu, skoky dodají větší variabilitu pozic jednotlivých znaků. Není tudíž jako v „neskákající“ gramatice zaručeno pořadí znaku *a* před *b*.

Příklad 3.3: Poslední příklad ukáže zajímavý úkaz, že skákající regulární gramatika dokáže generovat jazyk ze třídy kontextových jazyků:

Mějme skákající gramatiku $G_b = (\{A, B, C, a, b, c\}, \{a, b, c\}, \{A \rightarrow aB, B \rightarrow bC, C \rightarrow cA\}, A)$.

Generovaný jazyk $L\ (G_b, {}_j\Rightarrow) = \{w \in \Sigma^* \mid$ *počet výskytů znaků a, b, c v řetězci w je shodný*$\}$, který je dobře známým jazykem kontextovým, avšak oproti tomu jazyk $L(G_b, {}_s\Rightarrow) = \{abc\}\{abc\}^*$ je regulární.

Variabilita skoků dává gramatikám větší sílu při generování jazyků, avšak zároveň je určitým způsobem omezuje při generování některých jazyků. Vzpomeňme si na příklad z úvodní kapitoly, kde je lemma, že žádná skákající gramatika není schopna vygenerovat jazyk $L = \{a\}^*\{b\}^*$. A tento jazyk je přitom jazykem regulárním.

Některé vlastnosti skákajících gramatik

Obecné skákající gramatiky mají stejnou vyjadřovací sílu jako klasické obecné gramatiky.

$\mathscr{L}(\Gamma_{GG}, {}_s\Rightarrow) = \mathscr{L}(\Gamma_{GG}, {}_j\Rightarrow)$: toto tvrzení je dokázáno skrze důkaz, že $\mathscr{L}_0 \subseteq \mathscr{L}(\Gamma_{GG}, {}_j\Rightarrow)$.

GJF = $\mathscr{L}(X_{RLG}, {}_j\Rightarrow)$

JF = $\mathscr{L}(X_{REG}, {}_j\Rightarrow)$

Tyto třídy jazyků jsou navzájem neporovnatelné, avšak nejsou nespojité:

$\mathscr{L}_3$ a $\mathscr{L}(\Gamma_{MONG}, {}_j\Rightarrow)$, $\mathscr{L}_2$ a $\mathscr{L}(\Gamma_{MONG}, {}_j\Rightarrow)$, $\mathscr{L}_3$ a $\mathscr{L}(\Gamma_{REG}, {}_j\Rightarrow)$, $\mathscr{L}_2$ a $\mathscr{L}(\Gamma_{REG}, {}_j\Rightarrow)$.

Mezi otevřený problém z oblasti skákajících gramatik patří například otázka, jestli $\mathscr{L}(\Gamma_{CFG}, {}_j\Rightarrow) \subseteq \mathscr{L}(\Gamma_{CSG}, {}_j\Rightarrow)$.

Praktické využití skákajících gramatik

Tento typ gramatik je dobře využitelný ve výpočtech DNA. DNA je řetězec tvořený abecedou {G, A, T, C}. Např. *GGGGAGCTTTTGCCC*.

Skákající gramatiky konkrétně mohou pomoci vyřešit problém výskytu stejného počtu nukleotidu C a G a současně stejného počtu nukleotidu A a nukleotidu T uvnitř DNA řetězce. [3]

Pro řešení problému uvažme skákající verzi lineární gramatiky $G = (\{E, F, H, I, c, g, a, t\}, \{c, g, a, t\}, \{E \rightarrow gF, F \rightarrow cE, E \rightarrow H, H \rightarrow aI, I \rightarrow tH, H \rightarrow \varepsilon\}, E)$.

G je gramatikou generující pouze řetězce DNA splňující problém stejného výskytu znaků.

Pomocí standardních gramatik by se tento problém řešil obtížněji. Problematiku DNA computingu více do hloubky popisuje kapitola 6, včetně užití různých variant zejména skákajících formálních modelů v této oblasti.

Modifikace BKG gramatik

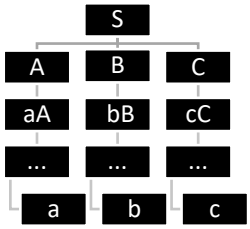
Jako jeden z příkladů můžeme uvést rozptýlené gramatiky – SCG (Gramatiky s rozptýleným kontextem). Tyto gramatiky při derivacích dovolují přepsat více neterminálů v jednom kroku, přesně podle pořadí v závorce, ale zároveň na rozdíl od běžných obecných gramatik, nemusí tyto neterminály ležet vedle sebe. Z toho vyplývá i název gramatiky rozptýlené nebo také gramatiky s rozptýleným

kontextem. (Zdroj: TID přednáška 19-10-2017) Tyto gramatiky najdou uplatnění při paralelním zpracování, případně v oblasti generování vět přirozeného jazyka podle správného slovosledu. Konkrétně např. tvorba otázek v angličtině a k nim příslušných odpovědí, tvorba záporných vět z kladných. [9]. Abychom získali konkrétní představu o principu tohoto typu gramatik, uveďme si jednoduchý příklad generování kontextového jazyka prostřednictvím SCG gramatiky.

Příklad: Mějme kontextový jazyk $L_{sc} = \{a^n b^n c^n : n \geq 1\}$ a příklad gramatiky s rozptýleným kontextem, která tento jazyk generuje $G_{LA} = (\{A, B, C, S, a, b, c\}, \{a, b, c\}, \{1: (S) \rightarrow (ABC), 2: (A, B, C) \rightarrow (aA, bB, cC), 3: (A, B, C) \rightarrow (a, b, c)\}, S)$.

Při derivačních krocích se nejprve uplatní pravidlo 1, poté n – krát pravidlo 2 a nakonec pravidlo 3, které všechny zbývající neterminály přepíše v jednom kroku na terminály a vznikne výsledná věta.

$S \Rightarrow ABC \Rightarrow aAbBcC \Rightarrow \ldots \Rightarrow aa\ldots abb\ldots bcc\ldots c$, *occur (a, w) = occur (b, w) occur (c, w).*



Obr 3-1 Derivační strom k příkladu rozptýlené gramatiky

Jak můžeme z příkladu vypozorovat, lze aplikaci jednotlivých pravidel znázornit pomocí derivačního stromu, samotná pravidla jsou ve velmi jednoduchém a čitelném tvaru, ve tvaru pravidel shodných s pravidly bezkontextové gramatiky. Ekvivalentní model generující L_{sc} je například obecná gramatika $G_{LB} = (\{A, B, C, S, a, b, c\}, \{a, b, c\}, \{S \rightarrow aSBC \mid abC, CB \rightarrow BC, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\})$. Zde už ovšem nezískáváme derivační strom využitelný v syntaktické analýze, tak jako v případě SCG gramatik.

Gramatika s rozptýleným kontextem tedy dodává k bezkontextové gramatice navíc paralelismus v podobě zpracování více pevně daných pravidel v jednom kroku se současným zachováním jednoduchosti tvaru pravidel. Jaká je vyjadřovací síla SCG? Je vyšší než síla BKG? Vyjadřovací síla tohoto modelu je **ekvivalentní s třídou rekurzivně spočetných jazyků**.

3.1 Semiparalelní skákající gramatiky

Do této třídy gramatik spadá verze gramatik SCG rozšířena o speciálně navržené módy skoků (jumping modes). [4]. Tyto gramatiky dále v textu budeme nazývat skákající gramatiky s rozptýleným kontextem (JSCG). V úvodní kapitole Modifikace verzí skákajících gramatik byla popisována „slabina“ obecné verze skákajících gramatik = nedetermismus skoků. Následně byl jako příklad ve stejné kapitole rozebrán s tím související důkaz věty, že neexistuje žádná MONG ani CSG skákající gramatika, která generuje regulární jazyk $L = \{a\}^*\{b\}^*$. Aby byl omezen popisovaný nedeterminismus u JSCG gramatik, jsou pro tento model zavedená striktně omezující pravidla:

Nechť máme pravidlo SCG gramatiky p: $(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n)$, ve všech použitelných módech skoku platí:

1. Vložení symbolu x_i musí být provedeno na libovolné místo mezi x_{i-1} a x_{i+1} v rámci větné formy, jinými slovy zůstává ve větné formě zachováno pořadí $x_1, \ldots, x_n$.
2. x_1 může být vloženo kamkoliv za výchozí pozici A_1 a zároveň platí, že x_n může být vloženo na libovolné místo před výchozí pozici A_n.
3. x_1 může být vloženo kamkoliv před výchozí pozici A_1 a zároveň platí, že x_n může být vloženo na libovolné místo za výchozí pozici A_n.

JSCG gramatiky nepoužívají přímo standardní skoky z (Definice 43), používané skoky jsou jejími modifikacemi a striktně se drží omezujících pravidel definovaných výše.

Ke klasickému sekvenčnímu módu, kdy je x_n umístěno striktně na pozici přepisovaného A_n přidává ještě módy dodržující některé z těchto upřesnění:

1. Použití libovolného módu skoku z def. 42. (zde je možný nedeterminismus skoků odstraněn použitím omezení (1), (2) a (3) a tím jednotlivé skoky musí být provedeny simultálně: $u_0A_1u_1A_2u_2 \ldots A_nu_n \Rightarrow v_0x_1v_1x_2v_2 \ldots x_nv_n$, kde platí $u_0u_1 \ldots u_n = v_0v_1 \ldots v_n$, $u_0z_1 = v_0$ a $z_2u_n = v_n$, $z_1, z_2 \in V^*$
2. Skoky mohou být libovolně dlouhé, avšak každý skok je proveden přesně na místo přepisovaného sousedního neterminálu:
   $u_0A_1u1A_2u_2 \ldots u_n \Rightarrow u_0u_1x_1u_2x_2 \ldots x_nu_{n-1}u_n$, kde platí $u_0u_1 \ldots u_n = v_0v_1 \ldots v_n$, $u_0z_1 = v_0$ a $z_2u_n = v_n$, $z_1, z_2 \in V^*$
3. Skoky nejsou provedeny simultálně, každý skok může být proveden kamkoliv, skok je pouze ohraničen sousedními dvěma neterminály. Nově vzniklé terminály $x_1 \ldots x_n$ se mohou křížit. Dodrženy jsou tedy pouze omezující podmínky (2) a (3).
   $u_0A_1u_1A_2u_2 \ldots A_nu_n \Rightarrow v_0x_1v_1x_2v_2 \ldots x_nv_n$, kde platí $u_0u_1 \ldots u_n = v_0v_1 \ldots v_n$, $u_0z_1 = v_0$ a $z_2u_n = v_n$, $z_1, z_2 \in V^*$

Podrobně jsou všechny módy včetně důkazů popsány v [4].

Síla skákajících rozptýlených gramatik:

JSCG gramatiky generující jazyky ze třídy rekurzivně spočetných jazyků.

4 Skákající Tree controlled gramatiky

Tyto gramatiky vycházejí z tzv. Tree controlled grammars [10], které rozšiřují o tzv. derivační módy neboli módy skoků. Derivační módy zavedené u TCG gramatik vycházejí ze článku o skákajících gramatikách s rozptýleným kontextem. [4]

Skákající tree controlled gramatiky (Jumping tree controlled grammars, JTC zkráceně) jsou uspořádané dvojice (G, R):

G = (V, T, P, S) je bezkontextovou gramatikou a R ⊆ (N ∪ T)* je regulární množina. Pro každý řetězec w ∈ L (G, R) musí platit, že všechny úrovně derivačního stromu znázorňující posloupnost derivačních kroků S ⇒* w vyjma poslední úrovně stromu musí být shodné s jedním z výrazů v R.

4.1 Derivační mód 1 (sekvenční mód)

V prvním derivačním módu není umožněno provádět žádné skoky. Gramatika G aplikuje n derivačních kroků. Při jednotlivých krocích vždy přepíše neterminální symbol za větnou formu na stejné místo, na jakém stál přepisovaný nonterminál.

$_1\Rightarrow$ reprezentuje derivační krok módu 1.

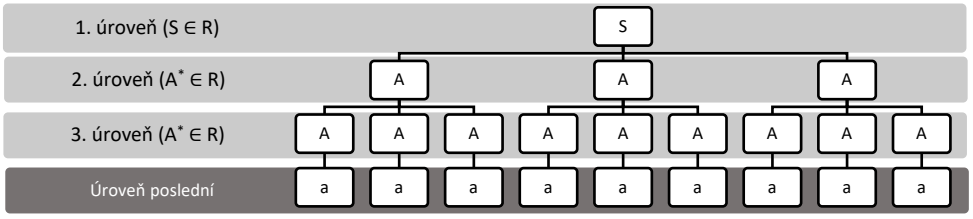
Nechť G = (V, T, P, S) je JTC a nechť w, z ∈ V*, A ∈ N, x ∈ V* a máme pravidlo A → x ∈ P,

pak derivační krok módu 1: **u** $_1\Rightarrow$ **v** v G, se provede právě tehdy a jen tehdy když u = wAz a v = wxz.

Příklad 4.1.1: *Demonstrujme příklad generování kontextového jazyka $L_1 = \{a\text{^}3^n \mid n \geq 1\}$. Takový jazyk vygeneruje JTC J = (G, R), G = ({S, A, a}, {a}, {S→AAA, A →AAA, A→a}),*

R ⊆ {S, A}* v sekvenčním módu, pro příklad uveďme generování řetězce a^{27}*:*

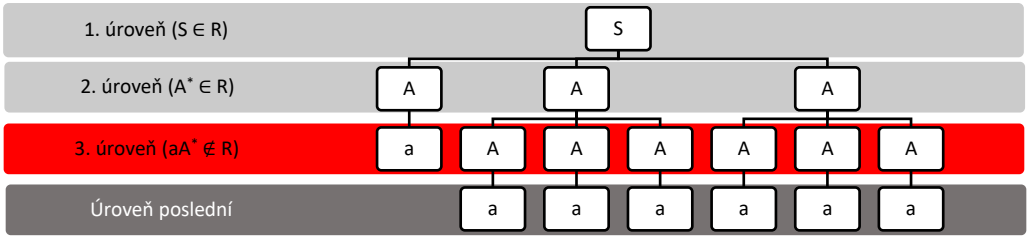
S $_1\Rightarrow$ *AAA* $_1\Rightarrow^3$ *AAAAAAAAA* $_1\Rightarrow^9$ *aaaaaaaaa*



Je evidentní, že regulární množina R nám částečně hlídá samotný postup aplikace pravidel, zejména pak neumožňuje aplikovat pravidla, která by v konečném důsledku způsobila vygenerování $w \notin L_1$, jako např. *aaaaaaa, |a| = 7.* Tento řetězec bychom při absenci kontrolní množiny R vygenerovali následující posloupností pravidel:

S $_1\Rightarrow$ *AAA* $_1\Rightarrow$ *aAA* $_1\Rightarrow$ *aAAAA* $_1\Rightarrow$ *aAAAAAA* $_1\Rightarrow^6$ *aaaaaaa.*

aaaaaaa $\notin L_1$, to musí nutně znamenat, že některá úroveň derivačního stromu $\notin R$*:*



Příklad 4.1.2: Mějme gramatiku *JTC H = (G, R), G = ({S, A, B, C, a, b, c}, {a, b, c}, {A → aA, B → bB, C → cC, S → ABC, A → a, B → b, C → c}).*

R ⊆ {S, ABC, aAbBcC, aAbBc, aAbB, cC, abcC}

Tj. jedná se o jazyk *L (H) = {$a^n b^n c^m$ | n, m ≥ 1}*

Každá úroveň stromu musí být jako v předchozím příkladu v souladu s jedním regulárním výrazem uvnitř kontrolní množiny R.

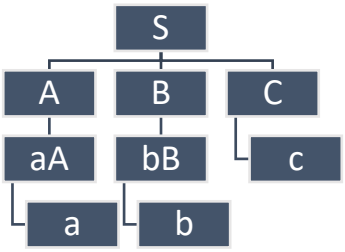
Kroky derivace gramatiky H generující kontextový jazyk *L (H)* mohou být např. tyto:

1) *S ₁⇒ ABC ₁⇒ aABC ₁⇒ aAbBC ₁⇒ aAbBc ₁⇒ aabBc ₁⇒ aabbc*

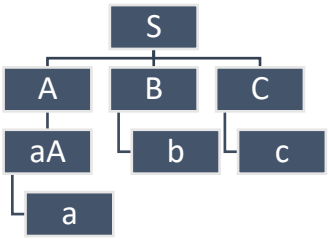
Zkusme provést derivační kroky odlišnou aplikací pravidel:

2) S ₁⇒ ABC ₁⇒ aAbc ₁⇒ aabc

Při druhém způsobu derivování jsme vygenerovali jiný jazyk, regulární jazyk *L (G, R) = {$a^i b^n c^m$ | i, n, m ≥ 1},* avšak na Obr 4-2 vidíme, že po aplikaci pravidel vznikne derivační strom, kde předposlední úroveň derivačního stromu nevyhovuje ani jednomu regulárnímu výrazu z množiny R, čili věta z druhého případu není věta generovaná gramatikou G.



Obr 4-1 První případ derivačních kroků -
1. úroven se shoduje s regulárním výrazem ABC,
2. úroveň aAbBc, 3. úroveň je poslední úrovní.



Obr 4-2 Druhý případ derivačních kroků - regulární výraz z předposlední úrovně derivačního stromu nevyhovuje žádnému z výrazů kontrolní množiny R.

Do jaké třídy Chomského hierarchie patří tento typ gramatik? Naše očekávání je, že tento model je silnější než BKG gramatiky. Předpokládáme, že vyjadřovací síla JTC v sekvenčním módu je rovna jazykům rekurzivně spočetným.

Teorém 1: JC (₁⇒) = RE

Lemma 1.1:

Třída bezkontextových jazyků je uzavřena vůči morfismu. [4]

Důkaz:

Je zřejmé, že libovolný jazyk generovaný JTC G gramatikou může být přijat Turingovým strojem M. Tudíž M akceptuje třídu jazyků $\alpha(G, 1\Rightarrow)$, z toho vyplývá, že platí JC $(1\Rightarrow) \subseteq$ RE.
Abychom ale dokázali, že je třída jazyků generovaných JTC gramatikami je ekvivalentní s jazyky rekurzivně vyčíslitelnými, musíme dokázat ještě tvrzení RE $\subseteq$ JC $(1\Rightarrow)$:

Lemma 1.2:

Nechť L $\in$ RE. Mějme dvě abecedy Σ a Γ, homomorfismus $h: T^* \rightarrow \Sigma^*$ a dva bezkontextové jazyky L_1 a L_2 takové, že platí: $L = h(L_1 \cap L_2)$
Nechť $L \in RE$. Vyjádříme tento rekurzivně vyčíslitelný jazyk podle Lemma 1.2. Budeme vytvářet gramatiku $G = (\{V_1 \cup V_2\}, T, \{P_1 \cup P_2\}, \{S_1 \cup S_2\})$. Dále vytvoříme nové terminální symboly $T = \{a_1, a_2, a_3, \ldots, a_n\}$ a $0, 1, S \notin V_1 \cup V_2$ bude představovat nový startovací symbol. Dále musí platit, že $V_1 \cap V_2 = \emptyset$

Definujeme nový homomorfismus

- $c: a_i \rightarrow 10^i1$
- $f: a_i \rightarrow h(a_i)c(a_i)$
- $C_1: V_1 \cup T \rightarrow V_1 \cup \Sigma \cup \{0, 1\}^*$

  $A \rightarrow A, A \in V_1 - T,$

  $a \rightarrow f(a), a \in T$
- $C_2: V_2 \cup T \rightarrow V_2 \cup \Sigma \cup \{0, 1\}^*$

  $A \rightarrow A, A \in V_2 - T, a \rightarrow c(a), a \in T$
- $t: \Sigma \cup \{0, 1\} \rightarrow \Sigma,$

  $a \rightarrow a, a \in \Sigma, A \rightarrow \varepsilon, A \notin \Sigma$
- $t': \Sigma \cup \{0, 1\} \rightarrow \{0, 1\},$

  $a \rightarrow a, a \in \{0, 1\}, A \rightarrow \varepsilon, A \notin \{0, 1\}$

Nechť $G = (V, \Sigma, P, S)$ je *JTC*, $V = V_1 \cup V_2 \cup \{S, 0, 1, \$\}$ a množina P obsahuje pravidla:

1) $S \rightarrow \$S_1 1111 S_2$
2) Pro každé pravidlo $A \rightarrow w \in P_1$ platí $A \rightarrow C_1(w)$
3) Pro každé pravidlo $A \rightarrow w \in P_2$ platí $A \rightarrow C_2(w)$

Claim 1: α (G, 1$\Rightarrow$) = L.

Nejdříve aplikujeme pravidlo 1: $S \rightarrow S_1 1111 S_2$. Následně probíhá simulace bezkontextových gramatik G_1 a G_2 pomocí pravidla 2. Tento postup nám vygeneruje větnou formu $w_1 1111 w_2$.
Předpokládejme nyní, že w_1 a $w_2 \in T^*$.
Pokud platí, že $t'(w_1) = w_2$, pak platí $t(w_1) = h(v)$, kde $v \in L_1 \cap L_2$ a $h (L_1 \cap L_2)$

Lemma 1.3:

Nechť Σ je abeceda. Pak existuje bezkontextová gramatika $G = (N, \Sigma, P, S)$, pak pro každý jazyk typu 0 L, $L \subseteq \Sigma^*$ tady existuje regulární množina R, $R \subseteq (\Sigma \cup N)^*$ taková, že $L = L(G, R)$.

Dokázáno v [10] na straně 132.

Dle Lemmy 1.3 vytvoříme regulární množinu R_1 pro gramatiku G_1, R_2 pro gramatiku G_2 respektive. Provedeme konkatenaci těchto vytvořených regulárních množin abychom získali regulární množinu R pro gramatiku G.

Nechť $w \in R$, kde $w \in T^*$, konkrétně se jedná o řetězec tvořený znaky *0* a *1*. Vytvoříme konečný automat M, který přijímá w_1 uvnitř věty $w_1 1111 w_2$. Následně pokud jsme stejným automatem schopni přijmout i větu w_2, pak můžeme prohlásit, že gramatika G generuje $h(v) \in L$.

Claim 1 znamená tvrzení, že RE ⊆ JC (1⇒). Teorém 1: JC (1⇒) = RE
platí.

4.2 Ekvivalence SCG a TCG gramatik

V 4.1 bylo ukázáno, že JTC gramatiky v módu 1 mají stejnou vyjadřovací sílu jako gramatiky s rozptýleným kontextem (SCG). Čili generují jazyky patřící do třídy rekurzivně spočetných jazyků (RE). Velkou pomůcku usnadňující dokázání některých modifikací Tree controlled gramatik by představoval algoritmus pro převod z TCG gramatiky na ekvivalentní SCG gramatiku.

K tomu, abychom mohli TCG gramatiky úspěšně využít v syntaktické analýze, je velmi užitečné také ověření, zda můžeme převést pravidla gramatiky TCG do některého normalizovaného tvaru. Jedním z nejvíce používaných je CNF (Chomského normální forma). Převedením gramatiky do Chomského normální formy získáme některé výhodné vlastnosti pro aplikace v syntaktické analýze. V oblasti syntaktické analýzy jsou využívány i aplikace, které striktně vyžadují pravidla v Chomského normální formě. Pojďme oddemonstrovat zjednodušený příklad obecné syntaktické analýzy založené na CNF, kde nám velmi pomůže mít gramatiku s pravidly v Chomského normální formě. Kompletní formální algoritmus popisující implementaci syntaktické analýzy založené na CNF je definovaný v [2], (přednáška 11, slide 6) nebo podrobněji včetně důkazu v [5].

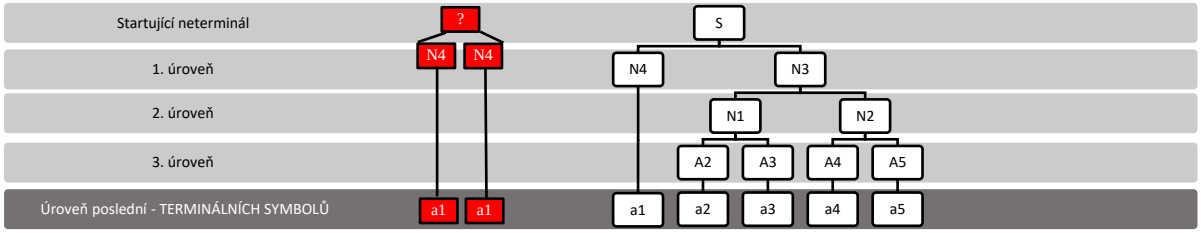
Mějme řetězec terminálů $a_1 a_2 a_3 \ldots a_n$ a potřebujeme zjistit, zda patří do jazyka generovaného gramatikou v CNF.

Z definice CNF plyne, že každé pravidlo musí být ve tvaru $A \rightarrow BC$, $A \rightarrow a$, čili jediná možnost, jak získat terminály $a_1 a_2 a_3 \ldots a_n$, je pomocí jednoduchých přepisovacích pravidel ve tvaru $A_k \rightarrow a_k$, $k \in 1, \ldots, n$. Pro neterminály $A_1 \ldots A_n$ budeme následně hledat pravidla, které mají na pravé straně 2 z těchto neterminálních symbolů. Postupně budeme postupovat do vyšších úrovní derivačního stromu až se postupně dostaneme na vrchol, do startujícího stavu. V případě, že se dostaneme až ke startujícímu stavu, jinými slovy dostaneme kompletní derivační strom derivující řetězec, odpověď následně zní

ANO, řetězec patří do jazyka generovaného gramatikou. V opačném případě, pokud se nedostaneme do startujícího stavu postupným hledáním pravidel, neboli se nám nepodaří získat kompletní derivační strom, řetězec nepatří do jazyka.

Př. Mějme *BKG G = (V, T, P, S) s množinou P = {S* $\rightarrow N_4N_3$, $N_3 \rightarrow N_1N_2$, $N_1 \rightarrow A_2A_3$, $N_2 \rightarrow A_4A_5$, $N_4 \rightarrow a_1$*}*

Pro řetězec $a_1a_2a_3a_4a_5$ je odpověď ANO, ale například pro $a_1a_1a_1a_2a_3a_4a_5$ je odpověď ne, jelikož pro tento řetězec nemůžeme vytvořit derivační strom.



Dále gramatika s pravidly v Chomského normální formě usnadňují dokazování. V důkazech pak nemusíme zkoumat všechny obecné možnosti, ale namísto toho provádíme důkaz pro konkrétní omezenou množinu. Celé tvrzení důkazu pak můžeme „obohatit“ větou „bez újmy na obecnosti platí“ a přímo překročit k dokazování na gramatice v normalizovaném tvaru. V neposlední řadě můžeme blíže specifikovat a konkretizovat některé vlastnosti derivačního stromu - např. vlastnost, že každý otec v CNF může mít nejvýše 2 syny.

4.3 Převod TCG do Chomského normální formy

Idea:

Převedeme pravidla TCG gramatiky do Chomského normálního tvaru. Hlavní nosnou myšlenkou pro převod je, že pravidla TCG gramatiky mají stejný tvar jako pravidla BKG gramatiky neboli gramatiky typu 2.

Teorém 4.1: Pro každou ε-free gramatiku typu 2 (gramatiku negenerující prázdný řetězec) G = (V, T, P, S) existuje ekvivalentní gramatika typu 2 H = (M, T, R, S) v Chomského normální formě. [5]

Dle teorému 4.1 jsme schopni pravidla každé TCG gramatiky typu 2 převést do CNF. Provedeme to pomocí algoritmu II.

Algoritmus II. (převod TCG gramatiky na Chomského normální formu)

Vstup: ε-free TCG G_1 = (G, R), G = (V, T, P, S), R ⊆ V^*

Výstup: Ekvivalentní G´ = (V´, T, P´´´, S) v Chomského normální formě

Metoda [11]:

1) Zbavíme se ε-pravidel:
    a) Odebereme všechna pravidla ve tvaru A → ε z P. (I gramatika negenerující jazyk obsahující prázdný řetězec může mít pravidla v tomto tvaru.)

b) Sestrojíme množinu $M = \{B \mid B \in N, B \Rightarrow^* \varepsilon\}$.

c) Přidáme do P´ všechna pravidla ve tvaru $A \rightarrow \beta$, kde $\beta \neq \varepsilon$; $A \rightarrow \alpha \in P$ a β je získána z řetězce α postupným odebíráním 0, …, n výskytů $B \in M$.

Příklad: *Nechť máme pravidla gramatiky $S \rightarrow A, A \rightarrow AabA, A \rightarrow \varepsilon$, pak dle kroku a) odebereme pravidlo $A \rightarrow \varepsilon$, dále A přidáme do množiny M podle b). Nakonec přidáme do P´ $A \rightarrow AabA \mid abA \mid Aab \mid ab$.*

2) Zbavíme se jednoduchých pravidel, tj. pravidel ve tvaru $A \rightarrow B$, kde $A, B \in N$:

a) Pro každý neterminál A sestrojíme množinu $N_A = \{B \mid A \Rightarrow^* B\}$.

b) Do množiny P´´ vložme všechna pravidla $B \rightarrow \alpha$ z P´, která nemají tvar jednoduchých pravidel.

c) Pro všechna A, pro které platí, že $B \in N_A$, přidej do množiny P´´ pravidla $A \rightarrow \alpha$.

Příklad: *Nechť máme pravidla gramatiky $A \rightarrow AX, A \rightarrow X, A \rightarrow c, X \rightarrow ab, X \rightarrow ba$, pak dle kroku a) získáme množiny $N_A = \{A, X\}, N_X = \{X\}$. Přidaná pravidla množiny P´ dle kroku b) budou $A \rightarrow AX, A \rightarrow c, X \rightarrow ab, X \rightarrow ba$. Krok c) nám přidá pravidla $A \rightarrow ab, A \rightarrow ba$.*

3) Odstraníme z pravidel cykly ve tvaru $A \Rightarrow^+ A$ pro nějaké $A \in N$.

4) Upravíme pravidla na pravidla tvaru CNF:

a) Do P´´´ přidáme všechna pravidla z množiny P´´, která mají tvar $A \rightarrow BC$ nebo $A \rightarrow a$

b) Pro všechna ostatní pravidla tvaru $A \rightarrow \alpha_1\alpha_{2\ldots}\alpha_n$, kde $n \geq 2$ a pro některé α_k v pravidle představuje terminální symbol: Nahraď každý terminál a novým neterminálem A_a, přidej upravené pravidlo do P´´´ a přidej související pravidlo $A_a \rightarrow a$.

c) Zaveď nové neterminály $C_{1, \ldots,} C_{n-2}$. Nahraď všechna pravidla tvaru $A \rightarrow B_1B_{2\ldots}B_n$, $n \geq 3$ z množiny P´´´ pravidly $A \rightarrow B_1C_1$, $C_1 \rightarrow B_2C_2$, …, $C_{n-3} \rightarrow B_{n-2}C_{n-2}$, $C_{n-2} \rightarrow B_{n-1}B_n$.

d) Všechny nově zavedené neterminály a obsah množiny V přidej do V´.

Příklad: *Nechť máme pravidlo gramatiky $K \rightarrow AXYZ$: pak toto pravidlo bude dle kroku c) nahrazeno třemi pravidly: $K \rightarrow AC_1, C_1 \rightarrow XC_2, C_2 \rightarrow YZ$.*

5) Druhá část TCG gramatiky je R – tj. množina regulárních výrazů popisující jednotlivé úrovně derivačního stromu. Čili cílem bude upravit stávající množinu regulárních výrazů R na množinu R‘ vyhovující nově vygenerovaným pravidlům. Provedeme podle algoritmu III. níže. V případě R´ musíme uvažovat i poslední úroveň derivačního stromu.

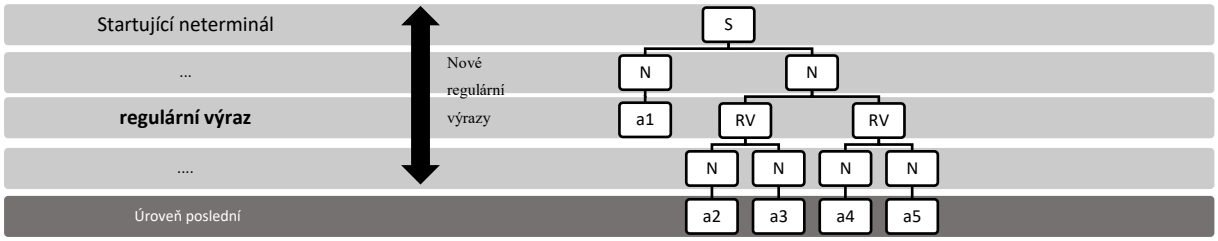
Teorém 4.2: Pro každou množinu R ε-free gramatiky TCG $G = (V, T, P, S)$ existuje množina R´, která respektuje změnu pravidel do Chomského normální formy a zároveň se nijak nezmění vyjadřovací síla původní TCG gramatiky.

Algoritmus III. (tvorba množiny R´ z původní množiny R v TCG gramatice):

Idea:

Pro každý regulární výraz vyhledáme pravidla z P´´´, která regulární výraz vygenerují. (každý regulární výraz představuje některou úroveň derivačního stromu mimo poslední úroveň). Využijeme metodu obecné syntaktické analýzy v CNF popisovanou výše.

Princip: Ke každému r ∈ R v TCG G_1 doplníme derivační strom. Přesněji řečeno dotvoříme derivační strom od startujícího neterminálu, přes regulární výraz r až po terminální řetězec patřící do jazyka prostřednictvím pravidel z množiny P´´´ gramatiky TCG G_2.



Vstup: TCG $G_1 = (G, R)$, G´

Výstup: TCG $G_2 = (G´, R´)$ taková, že $L(G_1) = L(G_2)$

Metoda:

Pro každé r ∈ R:

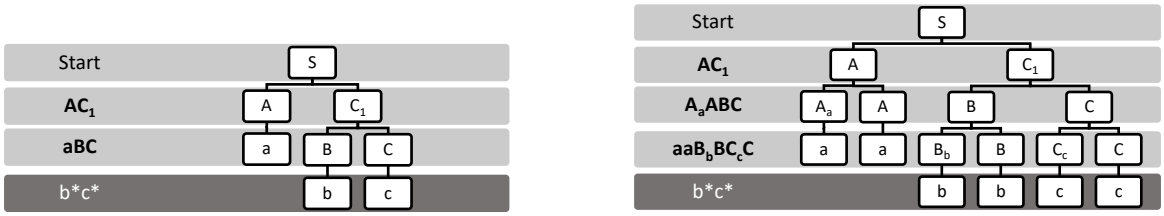
1) Najdeme posloupnost δ_1 pravidel z množiny P´´´, aby platilo $S\ [\delta_1] \Rightarrow r$.
2) Vytvoříme derivační strom znázorňující posloupnost derivačních kroků $S\ [\delta_1] \Rightarrow^* r$.
3) Derivační strom doplníme dvěma způsoby:
    a. Přepíšeme zbývající neterminály pomocí pravidel z P´´´ způsobem, abychom vygenerovali $w \in T^*$, $w \in L(G_2)$ a zároveň, aby strom měl minimální výšku.
    b. Přepíšeme zbývající neterminály pomocí pravidel z P´´´ způsobem, abychom vygenerovali $w \in T^*$, $w \in L(G_2)$ a zároveň, aby měl tento strom maximální možnou výšku bez opakování úrovní, tj. každá úroveň tohoto stromu obsahuje odlišný regulární výraz.
4) Do množiny R´ přidáme regulární výraz r´ pro každou úroveň takto vzniklého derivačního stromu vyjma poslední úrovně.

Příklad: Vstup: *TCG* $G_1 = (G, R)$, $G = (\{S, A, B, C\}, \{a, b, c\}, \{S \rightarrow ABC, A \rightarrow aA, B \rightarrow bB, C \rightarrow cC, A \rightarrow a, B \rightarrow b, C \rightarrow c\}, S)$, $R = \{S, ABC, aAbBcC\}$.

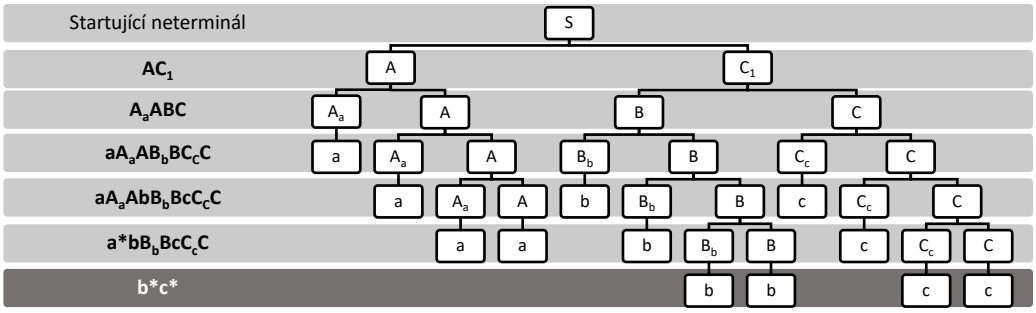
Budeme postupovat podle Algoritmu II. Tato gramatika neobsahuje žádná ε-pravidla, krok 1) přeskočíme. Přeskočíme i krok 2) a 3), neboť gramatika na vstupu neobsahuje ani jednoduchá pravidla, ani cyklická pravidla. Dle 4a) přidáme pravidla $A \rightarrow a, B \rightarrow b, C \rightarrow c$ do $P´´´$. Dle 4b) přidáme pravidla

$A \rightarrow A_aA$, $B \rightarrow B_bB$, $C \rightarrow C_cC$, $A_a \rightarrow a$, $B_b \rightarrow b$, $C_c \rightarrow c$. Dle 4c) Zavedeme neterminál C_1 a do P ´´´ přidáme pravidla $S \rightarrow AC_1$, $C_1 \rightarrow BC$. $V' = \{S, A, B, C, A_a, B_b, C_c, C_1\}$.

Tvorba množiny R´:



Obr 4-1 Strom dle algoritmu III. s minimální výškou k reg. výrazu ABC vlevo, k reg. výrazu aAbBcC vpravo



Obr 4-2 Strom dle algoritmu III. s maximální výškou k výrazu aAbBcC a výrazu ABC

Výstup: *TCG* $G_2 = (G', R')$, $G' = (\{S, A, B, C, C_1, A_a, B_b, C_c\}, \{a, b, c\}, \{S \rightarrow AC_1, C_1 \rightarrow BC, A \rightarrow a, B \rightarrow b, C \rightarrow c, A \rightarrow A_aA, B \rightarrow B_bB, C \rightarrow C_cC, A_a \rightarrow a, B_b \rightarrow b, C_c \rightarrow c\}, S)$, $R' = \{S, AC_1, aBC, A_aABC, aaB_bBC_cC, aA_aAB_bBC_cC, aA_aAbB_bBcC_cC, a^*bB_bBcC_cC\}$.

Důkaz:

Můžeme předpokládat, že δ_1 opravdu existuje. To plyne z korektnosti algoritmu II. (převod gramatiky na CNF). Dále hledání δ_1 v kroku 1 nemůže být nekonečně dlouhé. To by znamenalo, že by některé z $r \in R$ nepatřilo do ani jedné úrovně derivačního stromu původní gramatiky G_1, což můžeme vyloučit. Nekonečnost se dá vyloučit i tím, že máme konečný počet terminálů, neterminálů, pravidel gramatiky. Je evidentní, že $L(G_1) = L(G_2)$, jelikož R´ odvozujeme z R. Pro všechny regulární výrazy $r' \in R'$ a $r' \notin R$ platí jedna z těchto dvou možností:

1) $S \Rightarrow^* r' \Rightarrow^* r$
2) $r' \Rightarrow^* w$

Důkaz teorému 4.2:

Idea:

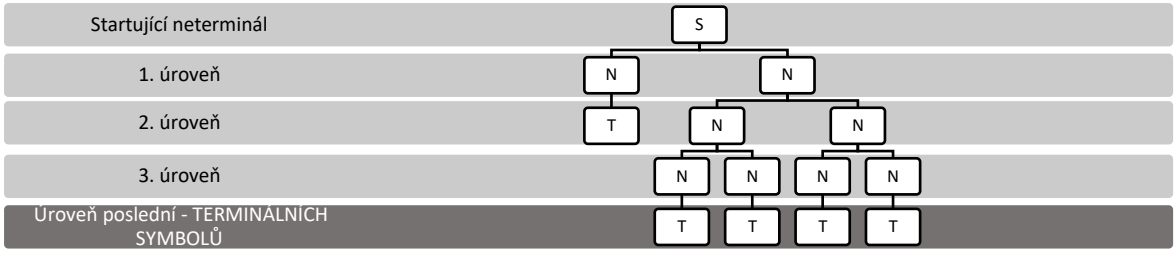
Budeme provádět důkaz sporem.

Předpokládejme, že existuje G_1 taková, že u některého derivačního stromu sestaveného pomocí algoritmu II. z P´´´ je minimálně 1 úroveň stromu, kterou není možno popsat žádným RV.

Pro každé $r \in R$ mohou nastat 2 případy:

1) S $[\delta_1] \Rightarrow^+$ r – jelikož máme konečnou množinu P, T, V, pak musí být na každé úrovni vzniklého derivačního stromu konečný řetězec. Což je spor s naším předpokladem, neboť každý jazyk obsahující právě jeden konečný řetězec lze popsat regulárním výrazem.
2) r $\Rightarrow^*$ w, z = card(A). Jedná se opět o stejný případ jako v bodu 1. Opět dojdeme ke sporu.

Derivační strom bude mít podobu jako na obrázku níže. V poslední úrovni stromu již mohou být pouze terminální symboly, ve zbylých úrovních je buď synem neterminálu předchozí úrovně buď jeden terminál nebo dva synové = dva neterminály.



Algoritmus IV. (Převod TCG na SCG gramatiku)

Idea:

Vstup: TCG $G_1 = (V_1, T_1, P_1, S_1)$, $R \subseteq V^*$ v Chomského normální formě, bez pravidel tvaru $S \rightarrow x$, $x \in T$

Výstup SCG $G_2 = (V_2, T_2, P_2, S_2)$ taková, že $L(G_1) = L(G_2)$

Metoda:

$V_2 := V_1$; $T_2 := T_1$; $S_2 := S_1$;

Konstrukce množiny P_2:

For each p: $A \rightarrow x$; WHERE $x \in T$

přidej p do P_2.

For each $r_i \in R$, $n > i \geq 1$, n = card(R) proveď:

While ($i < n$):

Pokud r_i $[\delta]$ $_{lm}\Rightarrow^n$ r_{i+1}, n = card(A), kde $A \in N$:

vlož δ jako n-tici $(A_1, \ldots A_n)$, jenž představuje levou stranu jednotlivých pravidel a poté jako n-tici i pravou stranu pravidel $(\alpha_1, \ldots, \alpha_1)$

break

Nebo pokud r_{i+1} $[\delta]$ $_{lm}\Rightarrow^n$ r_i, n = card(A), kde $A \in N$

vlož δ jako n-tici $(A_1, \ldots A_n)$ jenž představuje levou stranu jednotlivých pravidel a poté jako n-tici i pravou stranu pravidel $(\alpha_1, \ldots, \alpha_n)$

break

Nebo pokud r_i $[\delta]$ $_{lm}\Rightarrow^n$ w, $w \in T^*$, n = card(A), kde $A \in N$ a pro všechna A máme pravidlo $A \rightarrow a$

vlož δ jako n-tici $(A_1, \ldots A_n)$ jenž představuje levou stranu jednotlivých pravidel a poté jako n-tici i pravou stranu pravidel $(a_1, \ldots, a_n)$

break

Else i ++

Příklad:

Vstup: *TCG* $G_1 = (G, R)$, $G = (\{S, A, B, C, C_1, A_a, B_b, C_c\}, \{a, b, c\}, \{S \rightarrow AC_1, C_1 \rightarrow BC, A \rightarrow a, B \rightarrow b, C \rightarrow c, A \rightarrow A_aA, B \rightarrow B_bB, C \rightarrow C_cC, A_a \rightarrow a, B_b \rightarrow b, C_c \rightarrow c\}, S)$, $R = \{S, AC_1, aBC, A_aABC, aaB_bBC_cC, aA_aAB_bBC_cC, aA_aAbB_bBcC_cC, a^*bB_bBcC_cC\}$

Výstup: Ekvivalentní *SCG* $G_2 = (\{S, A, B, C, C_1, A_a, B_b, C_c\}, \{a, b, c\}, \{(S) \rightarrow (AC_1), (A, C_1) \rightarrow (a, BC), (B, C) \rightarrow (b, c), (A, C_1) \rightarrow (A_aA, BC), (A_a, A, B, C) \rightarrow (a, a, B_bB, C_cC), (B_b, B, C_c, C) \rightarrow (b, b, c, c), (A_a, A, B_b, B, C_c, C) \rightarrow (a, A_aA, b, B_bB, c, C_cC), (A_a, A, B, C) \rightarrow (a, A_aA, B_bB, C_cC), (A_a, A, B_b, B, C_c, C) \rightarrow (a, a, b, B_bB, c, C_cC), (A_a, A, B, C) \rightarrow (a, A_aA, B_bB, C_cC)\ \}, S)$.

4.4 Derivační mód 2 (jumping mód verze 1)

V prvním derivačním módu není umožněno provádět žádné skoky. Gramatika aplikuje postupně n derivačních kroků. Při každém kroku přepíše neterminální symbol za větnou formu vždy na stejné místo, na jakém stál přepisovaný neterminál. Tj. $vAwBz \ {}_1\Rightarrow^+ vx_awx_bz$ pro pravidla $A \rightarrow x_a$, $B \rightarrow x_b \in P$.

Nechť ${}_2\Rightarrow$ reprezentuje derivační krok módu 2.

Mějme H = (G, R), kde G = (V, T, P, S) a $R \subseteq V^*$ jako JTC gramatiku a nechť $u = u_0A_1u_1\ldots A_nu_n \in V^*$, $A_k \in N$, pro $k \geq 0$ a $n \geq k$, máme k dispozici pravidla gramatiky pro každé A_k, $A_k \rightarrow x_k \in P$, $x_k \in V^*$, $w_0A_1w_1A_2w_2 \ldots A_nw_n \in R$, $w_k \in V^*$, tj. řetězec se nachází na některé z úrovní derivačního stromu. Dále platí, že derivace módu 2:

$u_0A_1u_1A_2u_2 \ldots A_nu_n \ {}_2\Rightarrow^n v_0x_1v_1x_2v_2 \ldots x_nv_n$, kde platí:

$u_0u_1 \ldots u_n = v_0v_1 \ldots v_n$, $u_0z_1 = v_0$, $z_2u_n = v_n$, $z_1, z_2 \in V^*$.

Derivační krok módu 2 se od derivačního kroku 1 liší tím, že pokud aplikujeme pravidlo $A_k \rightarrow x_k \in P$, pak platí, že x_k nemusíme ve větné formě umístit přímo na pozici přepisovaného neterminálu A_k.

Derivační krok módu 2, ${}_2\Rightarrow$, je jump (Derivační módy), u kterého navíc platí **omezení** (1) a (2) a (3).

(1) Nechť $A_1, A_2, \ldots, A_n$ jsou neterminály na levé straně jednotlivých pravidel gramatiky JTC a $x_1, x_2, \ldots, x_n$ jsou výrazy na příslušné pravé straně pravidel gramatiky. Pak platí, že x_n je vždy vloženo striktně mezi x_{n-1} a x_{n+1} uvnitř produkované větné formy. Jinak řečeno, gramatika G nikdy nevloží x_i na pozici před x_{i-1} nebo za pozici x_{i+1}.

(2) x_1 je vloženo kamkoliv za původní pozici A_1, a zároveň x_n je vloženo kamkoliv před pozici A_n.

(3) x_1 je vloženo kamkoliv před původní pozici A_1, a zároveň x_n je vloženo kamkoliv za pozici A_n.

Pro upřesnění kroků derivačního módu 2 jsou tyto popsatelné jednoduchým algoritmem:

Algoritmus:

1) $A_1, A_2, \ldots, A_n$ jsou smazány.
2) $x_1, x_2, \ldots, x_n$ jsou vloženy mezi u_0 a u_n s dodržením omezení (1), (2) a (3).

Příklad:

Vezměme v úvahu následující gramatiku:

Mějme *JTC H = (G, R), G = ({S, A, B, C, a, b, c}, {a, b, c}, {(1) A → aA, (2) B → bB, (3) C → cC, (4) S → ABC, (5) A → a, (6) B → b, (7) C → c}, S)*

R ⊆ {S, ABC, aAbBcC}.

Pokud použijeme derivační kroky módu 2, $_2\Rightarrow$, jedná se o jazyk $L_H\,(G, R) = \{a^n b^n c^n \mid n \geq 1\}$?

Odpověď zní ne. Tato gramatika generuje zcela jiný jazyk, konkrétně jazyk:

$L_H\,(G, R) = \{\{a\}w\{c\} \mid occur\,(b, w) = occur\,(c, w) + 1, occur\,(b, w) = occur\,(a, w) + 1, n \geq 1\}$.

Nejdříve se aplikuje startující pravidlo *(4)*, po kterém *H* vygeneruje *ABC*. Dále buď ukončí derivování přepsáním neterminálů na příslušné terminálové symboly prostřednictvím pravidel *(5), (6), (7)* nebo pokračuje aplikováním pravidel *(1) (2)* a *(3)* v libovolném pořadí tak, aby úroveň derivačního stromu byla popsatelná některým regulárním výrazem z R. Avšak prostřednictvím derivace $_2\Rightarrow$ může dojít k potenciálnímu posunu *aA, bB* nebo *cC* a tudíž k záměně *a, b* a *c* ve výsledném řetězci. Jisté je pouze to, že výsledný řetězec bude začínat písmenem *a* a končit písmenem *c* – to vyplývá z algoritmu pro $_2\Rightarrow$ a definovanými omezeními (1), (2) a (3). Omezení zajišťují to, že při přepisování neterminálů nemůže dojít ke křížení. Tj. Např. řetězec derivovaný z *B, bB*, se nemůže ve větné formě vyskytnout před řetězcem derivovaným z *A, aA,* respektive za řetězcem derivovaným z *C, cC*. Krok 2 tohoto algoritmu v případě gramatiky H zaručuje, že první vygenerovaný symbol vlevo od A nebude nikdy posunut.

Například řetězec *abcabcabc* vygenerujeme následovně:

S $_2\Rightarrow$ ABC (4)$_2\Rightarrow$ aABC (1)$_2\Rightarrow$ aAbBC (2)$_2\Rightarrow$ aAbBcC (3)$_2\Rightarrow$ aAbcbBC (2)$_2\Rightarrow$ abcaAbBC (1)$_2\Rightarrow$ abcaAbBcC (3)$_2\Rightarrow$ abcaAbcbC (6) $_2\Rightarrow$ abcabcabC (5) $_2\Rightarrow$ abcabcabc (7).

Rozhodně možnost skoků při derivování nezaručuje jakékoliv pravidelné uspořádání znaků ve větě, a to i v případě aplikace omezení (1), (2), (3). V tomto příkladu jediné, co gramatika ošetřuje, je podmínka, aby každý vygenerovaný řetězec začínal symbolem *a* a končil symbolem *c*. Zároveň v této gramatice platí, že vygenerovaná věta má stejný počet *a, b* a *c*, což je zaručeno pomocí omezující množiny *R*.

Teorém: JC ($_2\Rightarrow$) = RE

Je zřejmé z Churchovy teze, že pro libovolnou gramatiku JTC G existuje Turingův stroj M, který přijímá α (G, $_2\Rightarrow$). Z toho vyplývá, že JC ($_2\Rightarrow$) ⊆ RE. Potřebujeme dokázat ještě opačné tvrzení, tj. RE ($_2\Rightarrow$) ⊆ JC.

Důkaz:

V důkazu budeme vycházet z Lemma 1.1:

a Lemma 1.2:

Dále využijeme algoritmu III. sloužícího pro převod TCG G na ekvivalentní gramatiku SCG G. Jumping mód verze 1 u TCG G je evidentně shodný s jumping módem verze 2 u SCG G. Pokračování důkazu bude využívat teorému 3.3 a teorému 3.5 ze [4]. Konkrétně se bude postupovat tak, že nejprve transformujeme využitím homomorfismu gramatiku G do nové podoby SCG gramatiky G´. Bude tudíž platit tvrzení, že L(G, $_1\Rightarrow$) = L(G´, $_1\Rightarrow$). Následně je dokázána ekvivalence mezi α(G´, $_1\Rightarrow$) a její podobou v jumping módu 1, tj. α(G´, $_1\Rightarrow$) = α(G´, $_2\Rightarrow$).

4.5 Derivační mód 3 (jumping mód verze 2)

V prvním derivačním módu není umožněno provádět žádné skoky. 2. mód již umožňuje provádět přechody se skoky $_j\Rightarrow$, ovšem pro omezení nedeterminizace jsou aplikovaná omezení (1), (2) a (3) a také je jednoznačně dána podoba algoritmu pro daný derivační mód. Derivační mód 3 bude dodržovat stejná omezení, avšak v jeho algoritmu budou změny. Podíváme se, jak se změní chování JTC G v módu 3.

Nechť $_3\Rightarrow$ reprezentuje derivační krok módu 3.

Mějme H = (G, R), kde G = (V, T, P, S) a R ⊆ V^* jako JTC gramatiku a nechť

$u = u_0A_1u_1 \ldots A_nu_n \in V^*$, $A_k \in N$, pro $k \geq 0$ a $n \geq k$,

máme k dispozici pravidla gramatiky pro každé A_k, $A_k \rightarrow x_k \in P$, $x \in V^*$,

$w_0A_1w_1A_2w_2 \ldots A_nw_n \in R$, $w_k \in V^*$ tj. řetězec se nachází na některé z úrovní derivačního stromu.

Dále platí:

Derivace módu 3:

$u_0A_1u_1A_2u_2 \ldots A_nu_n \; {}_3\Rightarrow^n \; v_0x_1v_1x_2v_2 \ldots x_nv_n$,

kde platí $u_0u_1 \ldots u_n = v_0v_1 \ldots v_n$, $u_0z_1 = v_0$ a $z_2u_n = v_n$, $z_1, z_2 \in V^*$.

Algoritmus pro mód 3 je následující:

Algoritmus:

1) $A_1, A_2, \ldots, A_n$ jsou smazány.
2) x_1 a x_n jsou vloženy dovnitř řetězců u_0 a u_n.
3) x_2 až po x_{n-1} jsou vloženy mezi nově vložené x_1 a x_n.

Pozn. ve všech krocích algoritmu jsou dodržena **omezení** (1) a (2) a (3) definována u předchozího módu.

Příklad:

Vezměme v úvahu následující gramatiku:

Mějme *JTC H = (G, R), G = ({S, A, C, a, b, c}, {a, b, c}, {(1) A → aAb, (2) C → cC, (3) S → AC, (4) A → ab, (5) C → c, (6) A → bAa }, S)*

R ⊆ {S, AC, aAbcC}.

Na první pohled bychom mohli mylně předpokládat, že se jedná o jazyk L_H *(H) = {w | occur (b, w) = occur (c, w) = occur (a, w), n ≥ 1}*.

Jelikož však použijeme derivační kroky módu 3, $_3\Rightarrow$, jedná se o jazyk L_H *(H) =* $\{\{a,b\}^{2n}c^n \mid n \geq 1\}$.

Proč je tomu tak? Jako první aplikujeme pravidlo (3), které přepíše startující neterminál *S*. Budeme mít k dispozici *AC*. Dle druhého kroku algoritmu jsou x_1 a x_n vloženy dovnitř řetězců u_0 a u_n. To znamená, že *C* bude vždy nejpravější neterminál, jelikož u_n = *epsilon*. V případě neterminálu A může docházet k jeho posuvu vlevo a díky tomu se mohou promíchat symboly *a* a *b* mezi sebou. Na konci však definitivně budou pouze znaky *c* ve shodném počtu s počtem znaků *a* a *b*. Ukažme si například posloupnost derivačních kroků, která vygeneruje řetězec *abababccc:*

S $_2\Rightarrow$ *AC (3)*$_3\Rightarrow$ *aAbC (1)*$_3\Rightarrow$ *aAbcC (2)*$_3\Rightarrow$ *aAbabcC (1)*$_3\Rightarrow$ *aAbabccC (2)*$_3\Rightarrow$ *abababccC (4)*$_3\Rightarrow$ *abababccc (5)*

Jak vidíme i z tohoto příkladu, jumping mód 3 nejvíce pomáhá při potřebě dosažení fixní pozice pro nejlevější a nejpravější neterminál. V praxi toho využijeme například v oblasti bioinformatiky při generování proteinových sekvencí, kde krajní podřetězce budou fixní a ve zbytku bude díky skokům umožněna variabilita. (viz kapitola 6.4.)

Teorém: JC ($_3\Rightarrow$) = RE

Je zřejmé z Churchovy teze, že pro libovolnou gramatiku JTC G existuje Turingův stroj M, který přijímá α (G, $_3\Rightarrow$), z toho vyplývá, že JC ($_3\Rightarrow$) ⊆ RE. Potřebujeme dokázat ještě opačné tvrzení, tj. RE ($_3\Rightarrow$) ⊆ JC.

Důkaz:

Důkaz bude analogický jako důkaz pro jumping mód verze 2. Dle algoritmu III. převedeme TCG G na ekvivalentní SCG G. Jumping mód verze 2 u TCG G je evidentně shodný s jumping módem verze 3 u SCG G. Pokračování důkazu bude využívat teorému 3.7 ze [4]. transformujeme využitím homomorfismu G do nové podoby SCG gramatiky G´, α(G, $_1\Rightarrow$) = α(G´, $_1\Rightarrow$). G´ bude mít pravidla:

1) (S) → ($S_1$11$$11S_2)
2) (A) → (C_i(w)) pro všechna A → w ∈ P_i, i = 1, 2.
3) (a, $, $, a) → ($, ε, ε, $), a = 0, 1;
4) ($) → (ε).

Následně je ve [4] dokázána ekvivalence mezi α(G´, $_3\Rightarrow$) a její podobou v jumping módu 1, tj. α(G´, $_1\Rightarrow$) = α(G´, $_3\Rightarrow$).

4.6 Derivační mód 4 (obecný jumping mód)

Ve čtvrtém derivačním módu ponecháme úplnou volnost skokům při provádění derivačních kroků. Tj. Při generování řetězce terminálních symbolů bude moci JTC G aplikovat libovolný ze čtyřech módů skoků definovaných u jumping grammars. (Derivační módy)

Nebudou již aplikována ani omezení (1), (2) a (3) jako tomu bylo u předchozích módů. Hlavní otázkou nyní bude, jak se změní vyjadřovací síla této verze JTC G oproti standardní sekvenční TCG G. Bude její síla rovněž shodná s RE? Na tyto otázky odpoví následující odstavce:

Nechť $_4\Rightarrow$ reprezentuje derivační krok módu 4. (jumping mód ve verzi 3)

Mějme H = (G, R), kde G = (V, T, P, S) a $R \subseteq V^*$ jako JTC gramatiku a nechť

$w_0A_1w_1A_2w_2 \ldots A_nw_n \in R$, $w_k \in V^*$ tj. řetězec se nachází na některé z úrovní derivačního stromu.

Proveď derivační krok $A_k \Rightarrow_j x_k$ pro každé $A_k \in R$, pro $k \geq 0$ a $n \geq k$.

V tomto derivačním módu platí jediné **omezení**, a sice $x_1 \ldots x_n$ se nachází na některé z úrovní derivačního stromu nebo $x_1 \ldots x_n \in T^*$.

Kompletní algoritmus má podobu následující:

Algoritmus:

1) $A_1, A_2, \ldots, A_n$ jsou smazány.
2) $x_1, x_2, \ldots x_n$ jsou umístěny na libovolné místo uvnitř větné formy.

V případě derivačního módu 4 nemůžeme napsat o jazyku žádnou bližší charakteristiku než tu, která je kontrolovaná skrze množinu R. Ve 4. derivačním módu již jsou generované řetězce omezovány pouze množinou regulárních výrazů. Pořadí a pozice všech neterminálních a terminálních symbolů je nedeterministická. Uveďme si zajímavý příklad:

Příklad:

S přihlédnutím k velkému nedeterminismu skoků u tohoto módu uvedeme demonstraci derivačních kroků na již dobře známé kontextové gramatice s jednoduchou sadou pravidel.

Mějme *JTC H = (G, R), G = ({S, A, B, C, a, b, c}, {a, b, c}, {(1) A → aA, (2) B → bB, (3) C → cC, (4) S → ABC, (5) A → a, (6) B → b, (7) C → c}, S), R ⊆ {S, ABC, aAbBcC}.*

Pokud bychom neměli k dispozici množinu R, gramatika H by byla ekvivalentní se skákající verzí bezkontextové gramatiky. Tj. generovaný jazyk by měl podobu regulárního jazyka obsahující symboly a, b, c v libovolném pořadí a také v libovolném počtu těchto znaků uvnitř řetězce. Pokud doplníme množinu regulárních výrazů R, pak se jedná o jazyk L_H *(G, R) = {w | occur (b, w) = occur (c, w) = occur (a, w), n ≥ 1}*. Tj. prostřednictvím omezení množinou *R* jsme v tomto případě místo regulárního jazyka obdrželi kontextový jazyk. *R* nám zajistila shodný počet symbolů *a, b, c* v libovolném pořadí. Kdybychom uvážili stejnou gramatiku *H*, avšak místo *JTC H* v módu 4 by se jednalo o klasickou gramatiku *TCG H,* generovaný jazyk by byl rovněž kontextovým jazykem, konkrétně L_H *(G, R) = {a^n b^n c^n | n ≥ 1},* tj. kromě shodného počtu symbolů v řetězci by byly symboly v definovaném pořadí.

Přichází zajímavá otázka v souvislosti se skoky: Je vyjadřovací síla JTC G módu 4 shodná s TCG G, tj. $\mathscr{L}(\Gamma_{JTC}, {}_4\Rightarrow) = \mathscr{L}(\Gamma_{TCG}, {}_s\Rightarrow)$?

Teorém: JC (4⇒) = RE

JC ($_4\Rightarrow$) ⊆ RE plyne z Churchovy teze. Budeme dokazovat: RE ⊆ JC ($_4\Rightarrow$).

Důkaz:

Myšlenka formy důkazu převzata z [3].

Pro každou TCG $H_1 = (G_1, R_1)$, $G_1 = (V_1, T, P_1, S_1)$ vytvoříme ekvivalentní JTC $H_2 = (G_2, R_2)$, $G_2 = (V_2 = V_1 \cup \{S_2, \$, \#, <, >\}, T, P_2, S_2)$ v módu 4.

$P_2 = \{S_2 \rightarrow \#S_1, \# \rightarrow <\$, <> \rightarrow \#, \# \rightarrow \varepsilon\} \cup \{\$\alpha \rightarrow >\beta, \alpha \rightarrow \beta \in P_1\}$.

R_2 pro JTC vytvoříme dle algoritmu III.

Gramatika H_2 nemůže při generování řetězce užít skoky, které by způsobily vygenerování věty w takové, že $w \notin L(H_1, {}_s\Rightarrow)$. Pokud by použila libovolný z jumping modes, ${}_{lj}\Rightarrow$ nebo ${}_{rj}\Rightarrow$, již by z dané větné formy nebylo možno žádnou sekvencí derivací získat větu. Tj. pro libovolnou větnou formu β bude platit:

$S \; {}_s\Rightarrow^* £ \; {}_j\Rightarrow \beta \; {}_s\not\Rightarrow^* w$,

$S \; {}_s\Rightarrow^* £ \; {}_s\Rightarrow \beta \; {}_s\Rightarrow^* w$,

Každá aplikace pravidla $\alpha \rightarrow \beta$ v H_1 můžeme v H_2 nasimulovat následovně:

$\ldots\#\ldots \alpha \ldots {}_j\Rightarrow \ldots <\$\alpha \ldots {}_j\Rightarrow \ldots <> \beta \ldots {}_j\Rightarrow \ldots\#\ldots\beta$.

Platí tvrzení: **$L(H_1, {}_s\Rightarrow) = L(H_2, {}_4\Rightarrow)$.**

Získáváme zajímavý výsledek, že možnost skoků u TCG gramatik nemá vliv na vyjadřovací sílu tohoto formálního modelu. V obou případech je vyjadřovací síla shodná s RE.

5 Skákající automaty a převodníky

5.1 Konečné převodníky

Zajímavým rozšířením klasických konečných a skákajících konečných automatů jsou konečné převodníky. Zjednodušeně řečeno, převodníky rozšiřují schopnost konečných automatů číst vstup o schopnost zápisu.

Převodník M (konečný převodník) je KA M obsahující 2 pásky a dvě hlavy. První páskou je klasická vstupní páska tak, jak ji známe z běžných končených automatů. Druhá páska je pro zápis a na začátku práce konečného automatu je tato páska prázdná. Anatomie obou pásek spočívá v rozložení 1 symbol = 1 políčko na pásce. Jaké výstupní symboly má převodník zapisovat na výstup je dáno převodní funkcí $\psi: \Sigma_I \rightarrow \Sigma_O^*$, kde Σ_I je abeceda vstupní pásky pro čtení a Σ_O abeceda pásky pro zápis (výstupní). Co je důležité poznamenat – výstupní a vstupní páska nemusí mít stejnou délku. Výstupní páska může být do nekonečna prodlužována vpravo. Zapisovací hlava má pozici vždy za posledním zapsaným políčkem. To konkrétně znamená, pokud máme na výstupní pásce zapsáno slovo *abeceda,* pak zapisovací hlava je umístěna na 8. pozici, tj. za posledním znakem *a*. Pozice zapisovací hlavy může být tedy na jiné pozici oproti čtecí hlavě – pozice obou hlav nejsou nijak synchronizovány. Konečnost tohoto formálního prostředku je ukrytá v konečné množině stavů a konečné množině pravidel, tak jako tomu je u konečných automatů. Navíc je samozřejmě konečná i množina překladových pravidel.

Jak tento formální prostředek pracuje?

Během přechodu mezi stavy automat nepřečte žádný nebo právě jeden symbol ze vstupní pásky a na základě převodní funkce zapíše na výstupní pásku jeden symbol, popř. řetězec. Následně se posouvá o jedno, případně o 0 polí vpravo a tímto způsobem přečte celou pásku. Pozice výstupní pásky se přesune vždy políčko za poslední zapsaný symbol bez ohledu na délku zapsaného řetězce. Případně pásku můžeme libovolně rozšiřovat vpravo. Pokud automat dojde na konec vstupní pásky a skončí v některém z koncových stavů, na výstupní pásce se nachází výsledný překlad.

Příklad:

Mějme konečný převodník $M = (\{s, f\}, \{1, 0\}, \{1: s1 \rightarrow s111, 2: s0 \rightarrow f0, s, \{f\})$ a translaci $T(M) = \{(1^i0, 1^{3i}0): i \geq 0\}$.

Jak bude vypadat převodní funkce u tohoto automatu:

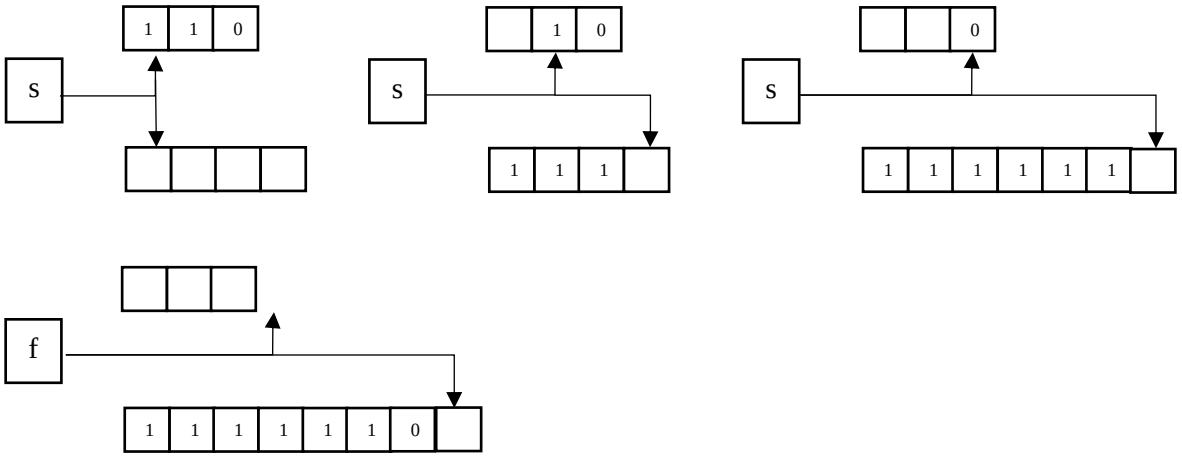
$\psi(1) = 111, \psi(0) = 0$

Vstup: 110, pozici hlav budeme značit v případě vstupní pásky tučným písmem, v případě výstupní podtržítkem.

Sekvence přechodů:

*s **1** 1 0 | _ (1) ||- s **1** 0 | 111_ (1) ||- s **0** | 111111_ (2) ||- f | 1111110_*

Pro ucelenější představu jednotlivé přechody budou zopakovány na obrázku níže:



Obr 5-1 Práce konečného převodníku (jednotlivé kroky čteme zleva doprava)

Nyní jsme tedy popsali účel a vlastnosti konečných převodníků, které vycházejí z konečných automatů a rozšiřují jejich vlastnosti. Nyní provedeme modifikaci tohoto typu převodníku na převodník vycházející ze skákajících automatů.

5.2 Skákající automaty

V případě běžných konečných automatů (Konečný automat) čteme vstupní pásku vždy kontinuálně, tzn. čteme ji sekvenčně symbol po symbolu, u každého z čtených symbolů provedeme přechod do následujícího stavu na základě pravidla a posuneme se čtecí hlavou přesně za pozici naposledy přečteného symbolu.

Příklad:

Mějme $M = (\{s, q, r\}, \{a, b, c\}, 1: sa \rightarrow q, 2: qb \rightarrow r, 3: rc \rightarrow s\}, s, \{s\})$, tento automat přijme libovolný řetězec patřící do jazyka $L(M) = \{\{abc\}^*\}$

Například řetězec *abcabc* bude přijat na základě této sekvence přechodů M:

s *a b c a b c (1)* |- ***q*** *b c a b c (2)* |- ***r*** *c a b c (3)* |- ***s*** *a b c (1)* |- ***q*** *b c (2)* |- ***r*** *c (3)* |- ***s***

Odlišnosti skákajících automatů

Co kdybychom uvážili jiný vstupní řetězec, např. permutaci řetězce *abcabc*: *bcabac*. Tento řetězec bychom již nemohli přijmout tímto konečným automatem, protože nemáme v M žádné pravidlo s kombinací stavu *s* a symbolu *b*.

Co kdybychom však umožnili čtecí hlavě se libovolně po pásce pohybovat a číst symbol ze kteréhokoliv místa na vstupní pásce, nikoliv pouze kontinuálně?

Pak bychom řetězec *bcabac* přijali pomocí této sekvence přechodů M:

b c ***s*** *a b a c (1)* |- ***q*** *b c b a c (2)* |- ***r*** *c b a c (3)* |- *b* ***s*** *a c (1)* |- ***q*** *b c (2)* |- ***r*** *c (3)* |- ***s.***

V tomto případě se jedná o skákající konečný automat - JFA (Skákající automaty). Jeho algoritmus kroků je:

1) Najdi libovolnou pozici pro aktuální stav q na vstupní pásce.
2) Proveď přechod na základě pravidla qa→ r ∈ R a opakuj kroky 1, 2 pro stav r.

Navíc na tomto příkladu vidíme, že jazyk přijímaný JFA M není regulární, ale je kontextový. Konkrétně se jedná o jazyk *L(M) = {w: occur (a, w) = occur (b, w) = occur (c, w)}.*

Lemma A: Nechť *w* představuje libovolný řetězec a *perm(w)* značí množinu všech permutací tohoto řetězce. Pro libovolný jazyk L přijímaným *JFA*, platí:

perm(L) = {perm(w): w ∈L}. [12]

Z toho bychom mohli vyvodit závěr, že JFA mají větší vyjadřovací sílu než KA. Ovšem to není pravda, např. neexistuje žádný JFA M, který přijímá např. jazyk $L_p = \{\{abc\}^*\}$ uváděný v příkladu výše. Přesto, že se jedná o jazyk regulární! Tím není tvrzeno, že nedokážeme sestrojit automat, který je schopen přijímat řetězce patřící do L_p, ale víme, že nedokážeme sestrojit JFA takový, který by přijímal právě tento jazyk. U sestrojeného JFA bude potíž v tom, že bude přijímat i libovolné permutace w ∈ L_p (lemma A), např. *cbacba.*

Kromě verze JFA máme ještě model obecných skákajících automatů (GJFA), tyto verze se od sebe liší tím, že v případě JFA M musí být všechna pravidla ve tvaru $pa \rightarrow q$, kde $a \in \Sigma$, ale oproti tomu v GJFA jsou pravidla ve tvaru $pw \rightarrow q$, kde w ∈ Σ^*. Jak to bude v tomto případě s jazykem L_p? GJFA jej rovněž nedokáže přijmout. Automat by mohl konkrétně vypadat například takto:

GJFA M = ({s}, {a, b, c}, {sabc → s}, s, {s}). Zdánlivě to vypadá, že automat přijímá právě tento jazyk, avšak zkusme si představit tento řetězec: *aabcbc*. GJFA M jej přijme touto sekvencí přechodů:

$a\ \mathbf{s}\ a\ b\ c\ b\ c \vdash_j \mathbf{s}\ a\ b\ c \vdash_j \mathbf{s}$, avšak $aabcbc \notin L_p$. Z toho vyplývá, že ani GJFA nemá větší vyjadřovací sílu než KA, tj. neplatí REG ⊂ GJF.

A jak je to s třídami jazyků GJF a JF? Víme určitě, že platí JF ⊆ GJF neboť GJFA je prostým rozšířením JFA o možnost přijímat více symbolů zároveň v rámci jednoho přechodu.

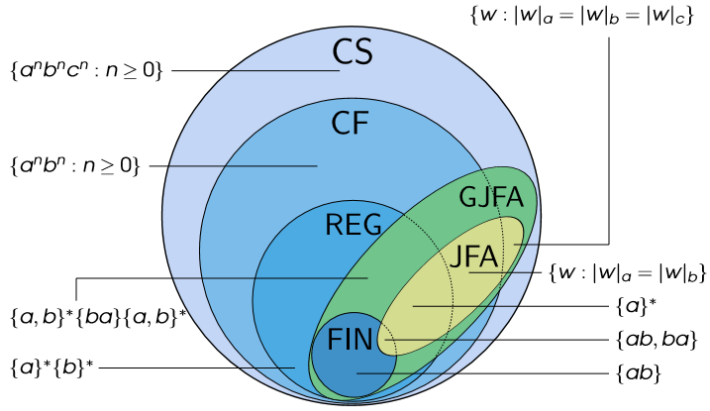
Platí JFA = GJFA? Neplatí.

Důkaz: Provedeme sporem: Najdeme např. regulární jazyk L_{REG}, který je přijímán automatem GJFA M, ale není přijímán žádným z automatů JFA N, tj. L_{REG} = L(M), $L_{REG} \neq$ L(N). $L_{REG} = \{a, b\}^*\{ba\}\{a, b\}^*$.

Automat *GJFA M = ({s}, {a, b}, {sba → q, qa → q, qb → q}, s, {q}).* Ale naproti tomu se nám nepodaří najít žádný JFA N, který by přijímal tento jazyk. U JFA N je omezující podmínka, že můžeme přijímat v jednom kroku pouze jeden symbol. Nemůžeme tedy nijak zaručit, že vygenerovaný řetězec bude obsahovat podřetězec ba. Znaky a, b se mohou libovolně „pomíchat“.

Z toho také mimo jiné vyplývá, že na rozdíl od případu konečných automatů, kdy model rozšířených konečných automatů (RKA), kde jsou pravidla ve tvaru $pw \rightarrow q$, $w \in \Sigma^*$ a konečných automatů (KA)

s pravidly ve tvaru $pa \rightarrow q$, $a \in \Sigma$ se jedná o ekvivalentní modely, tak v případě skákajících automatů a tříd jazyků přijímaných skákajícími automaty, je vztah tento: JF $\subset$ GJF.
Kompletní zařazení tříd jazyků přijímaných skákajícími automaty v kontextu tříd jazyků dle Chomského hierarchie zobrazuje tento diagram: [13]



Obr 5-2 Vyjadřovací síla jazyků přijímaných GJFA a JFA automaty v kontextu tříd jazyků dle Chomského **[12]**

Motivace v zavedení skákajících modelů automatů byla zejména v potenciálu aplikace těchto modelů v odvětvích bioinformatiky nebo syntaktické analýzy a jejich odlišná vyjadřovací síla oproti konečným automatům se zachováním jejich jednoduchosti.

Nyní zavedeme modifikaci skákajících automatů a vytvoříme skákající automaty omezené pravidly - controlled jumping finite automata, zkr. CJFA. Z nich následně vytvoříme model skákajících převodníků a tím získáme nové modely aplikovatelné především v oblasti bioinformatiky.

5.3 Skákající automaty CJFA

Tento model skákajících automatů vychází z GJFA. K definici obecných skákajících automatů doplníme omezení aplikace pravidel podle priorit a upřednostnění sekvenčních přechodů před skoky. Pojďme vytvořit definici CJFA.
Skákající automat s prioritou pravidel CJFA je pětice M = (Q, Σ, R, s, F):
Symboly a také konfigurace jsou definovány shodně jako v případě GJFA. (Obecné skákající automaty)

(1) Ke všem pravidlům p: (py, q) ∈ R přiřadíme prioritu pravidla e. Utvořené dvojice (p, e) uspořádáme relací ≥ dle priority e. Automat bude při provádění přechodů vždy upřednostňovat pravidla v pořadí daném prioritou jednotlivých pravidel. (vyšší e znamená vyšší prioritu) Mějme funkci C, která přiřazuje pravidlům p jejich prioritu, C: R → N definovanou takto: pro každé z pravidel p: (py, q): |y| = e. Tj. pokud budeme mít pravidla (saac,q), (st,r) a obě pravidla budou proveditelná na vstupu, pravidlo (saac, q) bude mít

prioritu 3, neboť |aac| = 3 a pravidlo (st, r) bude mít prioritu 1, neboť |t| = 1. Pak automat upřednostní pravidlo s prioritou 3: (saac,q), neboť 3 $\geq$ 1.

(2) Přechod u CJFA je definován:

Nechť $\chi = pyx$ a $\chi' = qx$ jsou dvě konfigurace CJFA M, p, q ∈ Q, x, y ∈ Σ^* a (py, q) ∈ R. Pak M může provést přechod $\chi \vdash \chi'$.

Pokud $\nexists \chi \vdash \chi'$ pro žádné z pravidel (py, q), pak může provést skok z xpyz do x´qz´, zapsáno jako: $xpyz \vdash_j x'qz'$.

Automat může provést skok jedině v případě, že nelze provést sekvenční přechod.

Pro přesnější popis práce automatu je zde uvedený jeho algoritmus.

Algoritmus:

Pro $\forall p \in R: \chi \vdash \chi'[p]$:

spočítej C(p) //spočítej prioritu pro každé pravidlo automatu

Seřaď pravidla dle priority sestupně.

while ($\chi \neq f$, kde $f \in F$)

reset i *//opět hodnotu i nastavíme na 0 kde 0-tý index je nejprioritnější pravidlo, n-tý index je pozice nejméně prioritního pravidla.*

For ($i \leq n$)

If $\exists \chi \vdash \chi'[p_i]$:

Proveď přechod $\chi \vdash \chi'[p_i]$

break

Else if $\exists p \in R: \chi \vdash_j \chi'[p_i]$:

Proveď přechod $\chi \vdash_j \chi'[p_i]$

break

Else

if i = n *// žádné z pravidel nemůžeme použít ani v sekvenčním, ani ve skokovém módu*

return (false) // vstupní řetězec w ∉ L(M)

++i

return (true)

Příklad:

Uvažme nyní jazyk *L = {aba, c}**. Sestrojíme automat *CJFA M = ({s}, {a, b, c}, {1: saba → s, 2: sc → s}, s, {s}).* Automat M přijímá zadaný jazyk L. Na ukázku přechodů vezměme například řetězec *ccabacaba:*

*c c **s** a b a c a b a (1)* $\vdash_j$ *c c c **s** a b a (1)* $\vdash_j$ ***s** c c c (2)* $\vdash_j$ ***s** c c (2)* $\vdash_j$ ***s** c (2)* $\vdash_j$ ***s.***

Dokážeme najít i GJFA, který přijímá daný jazyk?

Odpověď zní ano. Například tento:

GJFA M = ({s, q}, {a, b, c}, {1: saba → s, 2: sc → q}, s, {s, q}).

Pomůže nám nějakým způsobem aplikované omezení přednosti sekvenčních přechodů před skoky? Zkusme uvážit jazyk $L_p = \{\{abc\}^*\}$, u kterého už z předchozí kapitoly víme, že neexistuje GJFA M takový, že L(M) = L_p.

Mějme *CJFA M = ({s}, {a, b, c}, {sabc → s}, s, {s}).* Pomůže nám priorita sekvenčních přechodů před skoky. Odpověď zní ne. Důvod je ten, že jakmile automat nemůže provést sekvenční přechod, provede skok. Čili stejně jako GJFA přijímá například $aabcbc \notin L_p$.

Nechť $\mathscr{L}(\Gamma_{CJFA}, {}_s\Rightarrow)$ nazýváme třídou jazyků přijímaných CJFA automaty, zkráceně **CJF**. Příklad výše nás nabádá ke klíčové otázce, zda CJFA mají vyjadřovací sílu shodnou s GJFA.

Teorém: GJF = CJF

Abychom dokázali, že GJFA a CJFA přijímají stejnou třídu jazyků, musíme uvažovat, zda dokážeme pro CJFA najít ekvivalentní GJFA.

Důkaz:

První omezení u CJFA oproti GJFA spočívá v tom, že pokud existuje z aktuální konfigurace více proveditelných pravidel $p: ry \rightarrow q \in R$, mají vždy přednost proveditelná pravidla s největším $|y|$.

Pravidla z množiny *R* automatu *GJFA* tedy zmodifikujeme tak, že pro každý stav *r* vybereme všechna pravidla se stavem *r* na levé straně tohoto pravidla. Poté pro každé z dvojic pravidel $p_1: ry_1 \rightarrow q_1$ a $p_2: ry_2 \rightarrow q_2$, kde platí $|y_1| > |y_2|$ vytvoříme nový stav $r_{<y>}$, odebereme pravidlo $ry_2 \rightarrow q_2$. Přidáme nová pravidla $r \rightarrow r_{<y>}$ a $r_{<y>}y_2 \rightarrow q_2$. Tím zajistíme, že automat GJFA bude postupovat přesně podle priorit uvedených u CJFA.

Dalším omezením aplikovaným do CJFA je přednost sekvenčního přechodu oproti skokům. Nechť máme pravidlo $ry \rightarrow q$ a řetězec $w_1yw_2yw_3$. Bez ohledu na to, na jaké pozici se aktuálně nachází čtecí hlava, bude přesto výsledek shodný u GJFA a CJFA. Pokud $\chi \neq w_1\mathbf{r}yw_2yw_3$ nebo $\chi \neq w_1yw_2\mathbf{r}yw_3$, pak CJFA nemůže provést sekvenční přechod a provede skok. Následující konfigurace tedy bude $\chi' = w_1\mathbf{q}w_2yw_3$ nebo $\chi' = w_1yw_2\mathbf{q}w_3$.V případě GJFA jsou možnosti následující konfigurace totožné. Bez újmy na obecnosti můžeme předpokládat, že přednost sekvenčního přechodu oproti skoku nijak nezmění vyjadřovací sílu modelu skákajícího automatu.

Teorém GJF = CJF platí.

5.4 Skákající konečné převodníky

Stejně jako konečné převodníky rozšiřují konečné automaty o zapisovací pásku, tak skákající konečné převodníky jsou rozšířením skákajících automatů CJFA o možnost zápisu na zapisovací pásku.

Skákající převodník M je zjednodušeně CJFA M obsahující 2 pásky a dvě hlavy. První páskou je vstupní páska pro čtení, na které se můžeme pohybovat a číst z ní, tak jako v případě CJFA

s respektováním všech omezení, které má definované tento typ skákajícího automatu. Tj. sekvenční přechody mají prioritu před skoky a výběr pravidel se řídí prioritou pravidel získanou podle počtu symbolů přečtených v jednom kroku automatu. Zapisovací (výstupní) páska má strukturu i chování shodné s konečným převodníkem. 1 symbol = 1 políčko na pásce. Můžeme ji libovolně rozšiřovat vpravo a způsob zápisu je řízen převodní funkcí ψ: $\Sigma_I \rightarrow \Sigma_O^*$, kde Σ_I je abeceda vstupní pásky pro čtení a Σ_O abeceda pásky pro zápis (výstupní). Nutno poznamenat, že skoky prováděné na vstupní pásce nijak neovlivňují pohyb na výstupní pásce. Na výstupní pásce se zapisuje vždy kontinuálně, tak jako v případě konečných převodníku bez ohledu na aplikované skoky na vstupní pásce.

Pojďme nyní formálně definovat všechny důležité prvky a vlastnosti skákajícího konečného převodníku:

Definice:

Skákající konečný převodník, zkráceně CJFT, je pětice M = (Q, Σ, R, s, F), kde symboly Q, Σ, s, F jsou definovány totožně jako v případě konečných převodníků, $R \subseteq Q\ \Sigma_I^* \times Q\ \Sigma_O^*$, tvar pravidla (py, qz) $\in$ R zapisujeme častěji ve tvaru py → qz.

Konfigurace převodníku M, χ, je definována shodně jako u konečného převodníku. (Přechod a konfigurace u převodníku).

Vstupním automatem je CJFA M_I definovaný následovně:

Nechť M = (Q, Σ, R, s, F) je skákající konečný převodník. Vstupní CJFA automat M_I náležící k M:

M_I = (Q, Σ_I, R_I, s, F), kde Σ_I je vstupní abeceda převodníku M, R_I = {py → q: py → qx ∈ R, x ∈ Σ_O^*}.

Nechť $\chi = \chi_I$ | y a $\chi' = \chi_I'$ | yz jsou dvě konfigurace převodníku M a nechť r: qy → pz ∈ R, pak M udělá přechod z χ do χ', zapsaný jako $\chi \vdash_j \chi'[qy \rightarrow pz]$, zkr. $\chi \vdash_j \chi'$, pokud M_I může provést přechod $\chi \vdash_j \chi'[qy \rightarrow p]$. Tak jako je tomu v případě ostatních dříve definovaných automatů a převodníků, k dispozici máme i $\vdash_j^n$, $\vdash_j^+$, $\vdash_j^*$. Translace u skákajícího převodníku je definována shodně jako v případě translace konečného převodníku. (Translace u převodníku)

Příklad:

Mějme za úkol následující. Přeložit přijímané řetězce jazyka $L_1 = \{(a, b, c)^*\}$ na řetězce jazyka $L_2 = \{\{a\}^*\{b\}^*\{c\}^*\}$.

Tento účel splní např. převodník $M = (\{s, q, f\}, \{a, b, c\}, \{1: sa \rightarrow sa, 2: sb \rightarrow qb, 3: qb \rightarrow qb, 4: qc \rightarrow fc, 5: fc \rightarrow fc\}, s, \{f\})$.

Pro demonstraci vezměme v úvahu např. $w = aaccbca$, $w \in L_1$. Sekvence přechodů převodníku M překládající řetězec w na w', $w' \in L_2$:

s*aaccbca** | _ (1) ⊢ **s*a*ccbca** | a_ (1) ⊢ ccbcsa*** | aa_ (1) ⊢ ccs***b*c** | aaa_ (2) ⊢ q***c*cc** | aaab_ (4) ⊢ ***f*cc** | aaabc_ (5) ⊢ ***f*c** | aaabcc_ (5) ⊢ ***f*** | aaabccc_

Pozn. Pozici hlav značíme v případě vstupní pásky tučným písmem, v případě výstupní podtržítkem.

Složitější příklad, který bude demonstrovat zejména klíčové vlastnosti vstupního CJFA v podobě priorit a preference sekvenčních přechodů před skoky ukazuje příklad aplikace formálního modelu při detekci proteinů v bioinformatice.

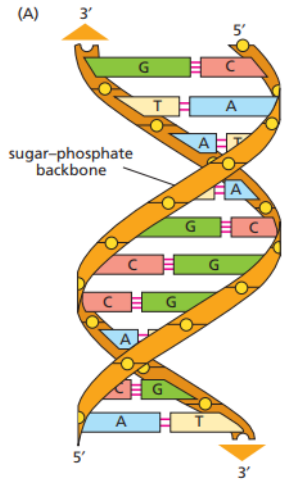
6 Aplikace v oblasti bioinformatiky

6.1 DNA struktura

Popsané teoretické formální modely nyní budeme aplikovat v oblasti bioinformatiky, konkrétně v DNA výpočtech.

DNA je deoxyribonukleová kyselina. Její hlavní rolí je ukládání informací. Je přítomna ve všech buněčných organismech od jednoduchých bakterií (prokaryotní organismy) až po lidi a zvířata (eukaryotní organismy). Jedná se o obrovskou makromolekulu, komplementární dvoušroubovici, sestavenou z malých stavebních bloků, nukleotidů – adenin (A), thymin (T), cytosin (C), guanin (G). Těmto stavebním blokům se také říká báze. I když se DNA molekuly skládají pouze z těchto 4 jednoduchých bází, ve skutečnosti se jedná o velice dlouhé sekvence čítající miliony nukleotidů. Z toho informace potřebná pro další reprodukci, pro přenos genetické informace, je uložena na relativně krátkém úseku DNA sekvence. Tuto část nazýváme genomem. U člověka například se jedná o 46 takovýchto molekul jako součást jednoho genomu. Potom ještě se v bioinformatice používá termín gen. Gen je jednoduše podmnožina genomu, konkrétně se jedná o úsek, ve kterém je zakódován protein.

Co znamená, že dvoušroubovice je komplementární? Komplementarita nám říká, jakým způsobem na sobě budou závislé dvě strany (vlákna) DNA sekvence svázané uvnitř dvoušroubovice. Thymin (T) je vždy svázán právě s Adeninem (A) a naopak A s T, Guanin (G) je vždy svázán s Cytosinem (C) a samozřejmě i obráceně C s G. Pokud známe nebo se nám podaří získat jednu stranu DNA sekvence, je jednoduché zjistit z ní podobu druhé strany. Např. pokud dostaneme sekvenci *ATCC*, pak druhá strana sekvence DNA bude *TAGG*.

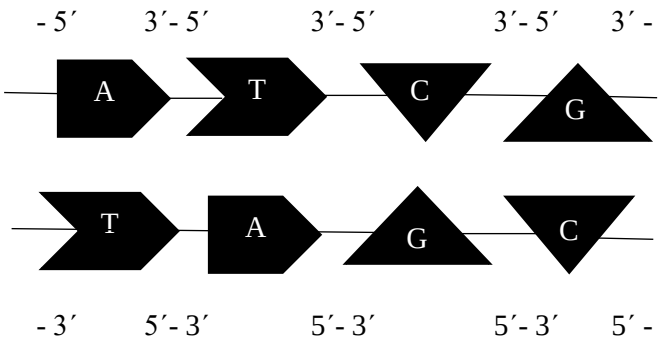


Obr 6-1 Příklad úseku dvoušroubovice DNA, zdroj **[14]**

Této vlastnosti využívá replikace DNA, kdy dochází k rozdělení dvoušroubovice na dvě samostatné sekvence s tím, že se následně na základě komplementarity dotvoří druhá strana sekvence.

Nyní pronikneme drobně do chemické stavby jednotlivých nukleotidů a popíšeme si, jakým způsobem se po chemické stránce vytváří vazba mezi jednotlivými nukleotidy, respektive jakým způsobem je tvořeno propojení jednotlivých nukleotidů do lineární sekvence. Vždy dochází k vazbě mezi volnou skupinou fosforečnanů (PO_4), což představuje 5′ pozici a volnou skupinou hydroxylovou (OH), což představuje pozici 3′.

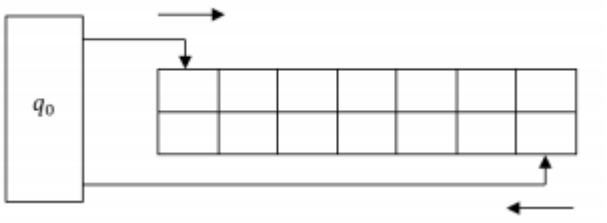
Mějme následující příklad:



Obr 6-2 Nukleotidy, chemické spojení, komplementarita

Ukázkovou sekvenci DNA můžeme číst ze dvou směrů, záleží na tom, jestli vlákno čteme v normálním nebo reverzním směru. Dále také se čtení liší v závislosti na výběru vlákna ze dvou možných komplementárních vláken. Sekvenci můžeme tedy přečíst jako ATCG (vrchní vlákno ve směru zleva doprava), TAGC (spodní vlákno zleva doprava), CGAT (spodní vlákno v reverzním směru – zprava doleva), ale také jako GCTA (horní vlákno zprava doleva). Pro zavedení systému ve způsobu čtení sekvence DNA byl zaveden jednotný standard, že sekvence jsou vždy zobrazovány v odborné literatuře a ve vědeckých článcích ve směru 5′ - 3′. Závěr z toho je ten, že sekvenci z obrázku 5-1 zapíšeme podle standardu buď jako ATCG nebo jako CGAT. Tyto dvě sekvence jsou navzájem komplementární a v odborné terminologii se nazývají palindromy.

Formální prostředek, který je možné aplikovat v tématice palindromů je Watson-Crick automat [15], pojmenovaný po dvou držitelích Nobelovy ceny za klíčové objevy v DNA Jamesi Watsonovi a F. Crickovi. Tento model automatu dokáže zkontrolovat u dvou sekvencí, zda jsou komplementární a mohou být součástí dvoušroubovice DNA. Jen takové sekvence patří do jazyka přijímaného tímto automatem. Jedná se o speciálně navržený automat pro tento účel vybavený speciální vlastností, že čte pásku současně ze dvou směrů dvěma souběžnými čtecími hlavami.



Obr 6-3 Princip čtení pásky Watson-Crick automatu

Definice:

Watson-Crick konečný automat je šestice M = (V, ρ, Q, s, F, δ), kde

V je abeceda, Q je konečná neprázdná množina stavů. Platí, že $V \cap Q = \phi$,

$\rho \subseteq V \times V$ značí symetrickou relaci, $s \in Q$ značí počáteční stav, $F \subseteq Q$ představuje množinu koncových stavů, δ je množina přechodů. $\delta: Q \times \binom{V^*}{V^*} \rightarrow P(Q)$ je mapování takové, že

$\delta: \left(s, \binom{x}{y}\right) \neq \phi$ pro všechny trojice $(s, x, y) \in Q \times V^* \times V^*$.

Zápis $s' \in \delta\left(s, \binom{x}{y}\right)$ interpretujeme jako: ze stavu s automat přijmutím znaku x na horní sekvenci DNA a přes y na dolní sekvenci DNA přejde do stavu s´.

Přechod u Watson-Crick automatu je definován následovně:

Pro $\binom{x}{y}, \binom{u}{v}, \binom{w}{z} \in \binom{V^*}{V^*}$, s, s´∈ Q, pak píšeme $\binom{x}{y} s \binom{u}{v} \binom{w}{z} \vdash \binom{x}{y} \binom{u}{v} s' \binom{w}{z}$, právě tehdy a jen tehdy, když $s' \in \delta\left(s, \binom{u}{v}\right)$.

transitivní a reflexivní uzávěr, $\vdash^*$ resp. $\vdash^+$, je definován analogicky jako u klasických konečných automatů.

Pokud se vrátíme k příkladu úseku DNA dvoušroubovice na Obr **6-2**, máme tam tuto část DNA sekvence:

$\longrightarrow$

A T C G

C G A T

$\longleftarrow$

Předpokládejme, že

$s' \in \delta\left(s, \binom{G}{C}\right), \delta\left(s, \binom{C}{G}\right), \delta\left(s, \binom{T}{A}\right), \delta\left(s, \binom{A}{T}\right), \delta\left(s', \binom{G}{C}\right), \delta\left(s', \binom{C}{G}\right), \delta\left(s', \binom{A}{T}\right),$

$\delta\left(s', \binom{T}{A}\right)$.

Pak posloupnost přechodů tohoto automatu bude mít tuto podobu:

$s \binom{A\,T\,C\,G}{C\,G\,A\,T} \vdash \binom{A}{T} s' \binom{T\,C\,G}{C\,G\,A} \vdash \binom{A}{T}\binom{T}{A} s' \binom{C\,G}{C\,G} \vdash \binom{A}{T}\binom{T}{A}\binom{C}{G} s' \binom{G}{C} \vdash \binom{A}{T}\binom{T}{A}\binom{C}{G}\binom{G}{C} s', s' \in F$.

Watson-Crick nám dokáže dát odpověď na otázku, zda jsme na vstup dostali dva palindromy DNA nebo nikoliv.

V této oblasti bychom mohli užít i obecných skákajících automatů – GJFA (Obecné skákající automaty) popř. by nám postačili i skákající konečné automaty - JFA, které na rozdíl od GJFA mohou v každém kroku přijímat nanejvýš jeden symbol.

Připomeňme, že automat M přijímá řetězec w tehdy a jen tehdy, když řetězec celý přečte a skončí v některém z koncových stavů. Proto JFA automat, který v případě, že sekvence na vstupu nejsou palindromy DNA, řetězec nedokáže přijmout.

Příklad automatu JFA, který by řešil popsanou úlohu, je tento: *JFA M = ({s, r}, {T, A, C, G}, R, s, {s})*, kde množina *R = {sA → r, rT → s, sG → r, rC → s}.*

***s** A T C G C G A T* |– ***r** T C G C G A T* |– *C **s** G C G A T* |– *C **r** C G A T* |– *C **s** G A T* |– ***r** C A T* |– ***s** A T* |– ***r** T* |– *s*

M přijímá palindromickou sekvenci z Obr **6-2**. Avšak tento automat příjme i chybné sekvence, jako např.

A T T A

C G C G

Tyto dvě sekvence nejsou palindromy DNA, přesto však je JFA M přijme pod touto sekvencí přechodů:

s** A T T A C G C G* |– ***r** T T A C G C G* |– *T **s** A C G C G* |– *rT C G C G* |– *C **s** G C G* |– *C **r** C G* |– *C **s** G* |– ***r** C* |– ***s

Zde už vnímáme obrovský přínos Watson-Crick automatů jednak v jejich paralelním čtení pomocí dvou čtecích hlav. A zejména pak ve správnosti vyhodnocení. Pokud Watson-Crick automat příjme řetězec, znamená to, že vstupní sekvence jsou palindromy DNA. Pokud nepřijme, sekvence nejsou palindromy. Pokud nepřijmeme řetězec pomocí JFA, znamená to, že sekvence nejsou palindromy, avšak pokud přijmeme řetězec prostřednictvím JFA, ještě to neznamená, že sekvence jsou korektní palindromy DNA. JFA pouze ověřuje, zda je v sekvenci přítomný shodný počet dvojic nukleotidů adenin (A) – thymin (T) a cythosin (C) – guanin (G), tj. jazyk přijímaný JFA M je právě tento:

$L(M) = \{w \in \{A, T, C, G\}^* \mid occur(A, w) = occur(T, w), occur(C, w) = occur(G, w)\}$

Jedná se o automat, který má stejnou vyjadřovací sílu, jako skákající lineární gramatika *G = ({E, F, H, I, c, g, a, t}, {c, g, a, t}, {E → gF, F → cE, E → H, H → aI, I → tH, H → ε}, E).*

Pozn. u gramatiky G jsou z důvodů dodržení jednotnosti s definicemi gramatik ponechány nukleotidy malým písmenem. Velkým písmenem jsou v gramatice označeny neterminály.

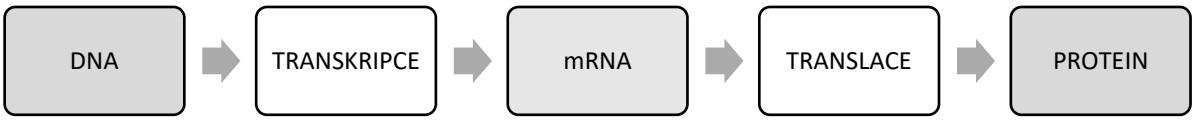
6.2 Detekce proteinů

Posuneme se do další části, která pojednává o tvorbě a rozpoznávání proteinů, které jsou základem všech živých organismů na Zemi. Některé z proteinů zajišťují pohyb (aktin, myozin), další důležitý protein je hemoglobin, který rozvádí kyslík po těle a pak jsou např. i proteiny, které napomáhají imunitnímu systému. Základními stavebními kameny proteinů jsou aminokyseliny. Pořadí jednotlivých aminokyselin poté určuje konkrétní strukturu proteinu.

Celý proces vzniku proteinů (proteosyntéza) začíná tvorbou tzv. message RNA (mRNA) pomocí transkripce. Transkripcí rozumíme přepis genetické informace z DNA do mRNA. Je k tomu zapotřebí

krátký úsek z celé DNA sekvence, u eukaryotních organismů k tomu postačí dokonce jen část informace z celé DNA zakódovaná uvnitř jednoho genu. [16] Pokud se podíváme blíže, jak tento proces probíhá, tak spuštění procesu probíhá v jádře za přítomnosti chromozómů, které obsahují DNA. Prostřednictvím enzymu Helikáza dojde k „rozpletení" vláken dvoušroubovice DNA ve vybraném místě. Další důležitý enzym Polymeráza následně kopii získané informace zapíše do RNA. Při přepisu dojde k nahrazení nukleotidu Thymin (T) na Uracil (U), jinak se informace včetně pořadí jednotlivých nukleotidů oproti DNA nijak nezmění. Po dodatečných chemických úpravách bude výsledkem tohoto procesu mRNA. Následně je úkolem buňky vyhledat v sekvenci místo, ve kterém je protein zakódován. U eukaryotických buněk (složitější organismy jako lidé, zvířata) tato sekvence bude obsahovat jeden konkrétní gen, uvnitř kterého máme zakódovaný protein. U tohoto typu buněk však není jednoduché rozpoznávání proteinů, jelikož proteiny nejsou zakódovány v sekvenčním pořadí, sekvence mRNA obsahuje prázdná místa, a dokonce mohou být jednotlivé proteiny v sobě navzájem zanořené. U prokaryotních buněk (bakterie, jednoduché organismy) sekvence mRNA naopak obsahuje několik genů za sebou, ne pouze jeden, avšak zřetězených sekvenčně za sebou bez přerušování, prázdných míst a zanořování jednotlivých proteinů do sebe, na rozdíl od eukaryotních buněk.

Navazující část se nazývá translace. Ta probíhá za pomoci ribozómů v cytoplazmě a jak již název napovídá, tato fáze překládá – konkrétně získané mRNA na kód proteinu. Jak bylo zmíněno výše, RNA skladba je podobná DNA, pouze místo nukleotidu thyminu (T) je přítomen nukleotid uracil (U). Zde už nás nebudou zajímat konkrétní nukleotidy, ale spíše jejich trojice. Každá trojice nukleotidů, kterou nazýváme kodon, představuje jednu z 20 aminokyselin. Ty představují stavební kameny pro proteiny.



Obr 6-4 Průběh tvorby proteinu

Problém nastává v tom, jak pomocí 4 typů nukleotidů specifikovat pomocí jejich trojice právě 20 typů aminokyselin. Počet trojic, které je možno z 4 nukleotidů vygenerovat je roven 4^3. Řešení je zřejmé: jedna konkrétní aminokyselina je specifikována více než jedním kodonem. Můžeme prohlásit, že dochází k degeneraci genetického kódu. Tomu rozumíme tak, že můžeme dedukovat sekvenci aminokyselin jednoznačně z DNA, resp. RNA sekvence, ale naopak není možné jednoznačně zjistit konkrétní sekvenci nukleotidů ze zadané sekvence aminokyselin. Seznam všech 20 aminokyselin a jejich kódování pomocí nukleotidů zachycuje Tabulka 1 [17]:

Kodon	Aminokyselina
UUU, UUC	F: Phe (Phenylalanin)
UUA, UUG, CUU, CUC, CUA, CUG	L: Leu (Leucin)
AUU, AUC, AUA	I: Ile (Isoleucin)
AUG	M: Met (Methionin)
GUU, GUC, GUA, GUG	V: Val (Valin)
UCU, UCC, UCA, UCG, AGU, AGC	S: Ser (Serin)
CCU, CCC, CCA, CCG	P: Pro (Prolin)
ACU, ACC, ACA, ACG	T: Thr (Threonin)
GCU, GCC, GCA, GCG	A: Ala (Alanin)
UAU, UAC	Y: Tyr (Tyrosin)
UAA, UAG, UGA	STOP KODON
CAU, CAC	H: His (Histidin)
CAA, CAG	Q: Gln (Glutamin)
AAU, AAC	N: Asn (Asparagin)
AAA, AAG	K: Lys (Lysine)
GAU, GAC	D: Asp (Asparatická kyselina)
GAA, GAG	E: Glu (Glutamická kyselina)
UGU, UGC	C: Cys (Cystein)
UGG	W: Trp (Tryptophan)
CGU, CGC, CGA, CGG, AGA, AGG	R: Arg (Arginin)
GGU, GGC, GGA, GGG	G: Gly (Glycin)

Tabulka 1 Aminokyseliny a jejich kódy

Translace trojic nukleotidů na aminokyseliny může probíhat třemi různými způsoby, záleží, kde začneme překládat. Jestli již od prvního nukleotidu, od druhého nebo až od třetího. Jinak řečeno, máme posuvné okénko (reading frame) o šířce 3 nukleotidy a pro každou ze třech možných počátečních pozic obdržíme úplně rozdílnou sekvenci aminokyselin.

Příklad: Mějme sekvenci mRNA:

AGGGUUACCAUCUC

- první způsob: *AGG GUU ACC AUC UC = Arg-Val-Thr-Ile-*
- druhý způsob: *A GGG UUA CCA UCU C = -Gly-Leu-Pro-Ser-*
- třetí způsob: *AG GGU UAC CAU CUC = -Gly-Tyr-His-Leu-*

V reálných biologických procesech je pomocí kontrolních signálů ve většině případů docíleno toho, že pouze jedním ze tří možných umístění reading frame se nám podaří detekovat korektně protein. Nevhodně zvolený reading frame se nejčastěji projeví tím, že stop kodon je příliš blízko start kodonu a sekvence proteinu je příliš krátká. Takto krátké proteiny se nevyskytují až na pár výjimek, kterými jsou např. proteiny imunitního systému.

V oblasti aminokyselin bychom mohli využít GJFA (Obecné skákající automaty). Ty by nám dokázali odpovědět na otázku, zda se v získané sekvenci mRNA nacházejí určité aminokyseliny.

Příklad:

Potřebovali bychom zjistit, zda se v sekvenci mRNA nachází aminokyselina Tyrosin, která je součástí velmi důležitých stresových hormonů – adrenalinu a nonadrenalinu. Musíme mít na paměti reading frame a jeho vliv na detekci jednotlivých aminokyselin. Předpokládejme, že nemáme k dispozici žádnou pravděpodobnostní statistiku výskytu, která by nám napověděla pozici aminokyseliny v řetězci. Jednoznačnou odpověď nám poskytne jednoduchý *GJFA M = ({s, f}, {U, A, C, G}, R, s, {f})*, kde množina $R = \{sUAU \rightarrow f, sUAC \rightarrow f, fU \rightarrow f, fA \rightarrow f, fC \rightarrow f, fG \rightarrow f\}$.

Automat provede skok libovolné délky, aby přijal kodon aminokyseliny Tyrosin, kterým je buď UAU nebo UAC a tím zároveň automat přejde do koncového stavu. Následně automat přijímá zbytek sekvence po jednom znaku. Demonstrujme na úseku mRNA z předchozího příkladu:

$AGGGU\ \mathbf{s}\ \mathbf{UAC}CAUCUC \vdash AGGGU\ \mathbf{f}\ CAUCUC \vdash AGGGU\ \mathbf{f}\ AUCUC \vdash AGGGU\ \mathbf{f}\ UCUC \vdash^{*} \mathbf{f}.$

Pro libovolné jiné aminokyseliny či jejich skupiny a sekvence bychom použili analogickou metodu. Zároveň platí zajímavé tvrzení, že neexistuje žádný JFA, kterým bychom tuto úlohu dokázali vyřešit, respektive neexistuje JFA takový, který přijímá jazyk $\{\{U, A, C, G\}^{*}\{UAC, UAU\}\{U, A, C, G\}^{*}\}$.

6.3 Možné vstupy

V předchozí části byl popsán proces od původní DNA sekvence až po samotný protein z částečně biologického pohledu a částečně pohledu teoretické informatiky a formálních modelů. Jak ale opravdu bude v reálu vypadat vstup, který obdržíme a který nám poslouží pro další práci.

Máme 3 varianty vstupů, které můžeme obdržet a ke každé variantě musíme přistupovat jinými metodami zpracování.

Nejsložitější, ale také bohužel nejčastější případ je, že dostaneme k dispozici jednotlivé fragmenty DNA sekvence. Jejich následné zpracování vyžaduje znalost konkrétního organismu a stavby jeho genu. Cílem u této varianty je poskládat ze získaných malých fragmentů gen pro následnou analýzu. V této práci se budeme věnovat hlavně dalším dvěma variantám, a to případu, kdy obdržíme na vstupu DNA genom a variantě, kdy obdržíme výsledek transkripce DNA.

Při obdržení genomu DNA

Našim cílem bude získat z tohoto vstupu mRNA sekvenci k dalšímu zpracování. Prvním krokem k získání sekvence mRNA je zjistit, kde v dané DNA sekvenci začíná transkripce. Tyto sekvence značí speciální start a stop signály. Úkolem tedy bude tyto signály vyhledat a co nejpřesnější metodou je detekovat. Signály určující start transkripce se nazývají promotóry. U **prokaryotních** buněk se nachází přesně před místem TSS (transcription start site) a můžeme je rozpoznat podle charakterizující krátké sekvence nukleotidů na pevně daných pozicích. Sekvence přirozených a hybridních promotorů jsou určeny pravděpodobnostní tabulkou. Nejčastěji vypadají následovně:

30 pozic zleva od prvního nukleotidu m RNA	**10 pozic zleva od prvního nukleotidu m RNA**
TTGACA	TATAAT
TTTACA	TATATT
TTGACA	TTAACT
TTGACA	GATACT
TTGATA	TATAAT
TTGACA	TATAAT
TTGACA	TTTAAT

Tabulka 2 Sekvence promotorů **[18]**

U eukaryotních buněk je detekce těchto signálů podstatně složitější, jelikož se jedná jen o velmi krátké sekvence na místech, které nejsou tak dobře specifikovatelných jako u prokaryotních buněk.
Naprostá většina genů obsahuje kromě startovacích sekvencí transkripce i ukončovací sekvence. Tato sekvence obsahuje palindromatickou strukturu o délce 7 – 20 nukleotidů + 6 uracilů za sebou.
V této části zpracování vstupu se nám tolik nepodaří využít skákající modely. V úvahu připadá pouze využití JTC v některém z módů, která by tvořila náhodně sekvence DNA s tím, že by na stanovená místa umístila startovací a stop signály pro transkripci. mRNA sekvence by však poté měli pouze učebnicovou strukturu použitelnou např. pro testování nástrojů na detekci signálů, nikoliv však obsahující reálné geny, které kódují proteiny. Zde připadá v úvahu využití některého z pravděpodobnostních modelů. Případně můžeme z obdrženého genomu přímo získávat sekvence proteinů na základě znalostí a předpokladů, které o struktuře DNA a proteinů máme, a tím jsou zejména tyto dva fakty [19]:

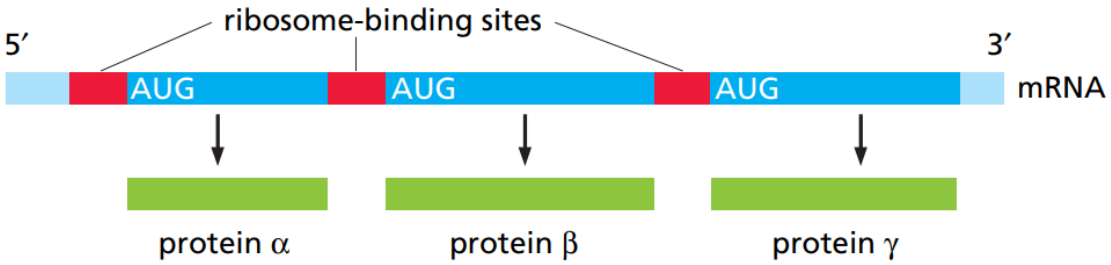
- Proteiny jsou většinou ohraničeny start kodonem a stop kodonem uvnitř sekvence DNA / RNA.
- Většina proteinů má délku větší než 21 kodonů, jsou však i výjimky, např. v případě nervového nebo imunitního systému.

Na základě těchto dvou znalostí dokážeme stvořit Turingův stroj, který by nám na základě toho našel všechny sekvence, které uvedeným dvěma bodům vyhovují a s velkou pravděpodobností se jedná o proteiny. Tento Turingův stroj je implementován a jeho výstupy popsány v přiloženém programu.

První programy na hledání proteinů takto opravdu fungovaly, dnes je však tato metoda považována za naivní, jelikož neobsahuje rozšíření o práci s pravděpodobností a statistikou výskytů kodonů.

Při obdržení výstupu transkripce

V případě této varianty dostáváme na vstup sekvenci mRNA a našim cílem bude v této sekvenci detekovat konkrétní aminokyseliny a proteiny z nich sestavené. Opět vycházíme ze známého faktu v oblasti bioinformatiky, že většina kódových sekvencí začíná aminokyselinou Methionin, která má zkratku AUG (ATG) a vždy končí jedním z tzv. STOP kodonů. (Tabulka 1 Aminokyseliny a jejich kódy). U prokaryotních buněk obsahuje mRNA v sobě několik genů sekvenčně za sebou a uvnitř těchto genů jsou zakódovány jednotlivé proteiny. Proteiny jsou zakódované rovněž sekvenčně za sebou, tak jak je vidět na obrázku níže. [14]



Obr 6-5 Struktura mRNA u prokaryotních buněk **[14]**

Zde se tedy přímo nabízí využít konečné skákající převodníky, které úlohu získání všech proteinových sekvencí z obdržené mRNA vyřeší.

Pro zjednodušení budeme uvažovat, že sekvenci mRNA jsme obdrželi na vláknu A ve standardním směru 5´- 3´. Pokud bychom dostali mRNA bez jakékoliv další specifikace směru a vlákna, museli bychom brát v úvahu i reverzní směr a například při hledání start kodonu bychom kromě AUG museli vyhledávat i UAC, GUA nebo CAU.

Na vyřešení úlohy můžeme použít skákající převodník $M = (Q, \Sigma, R, s, F)$,

$Q = \{s, q, f\}$, $\Sigma = \{A, U, C, G, F, L, I, M, V, S, P, T, A, Y, H, Q, N, K, D, E, C, W, R, G\}$, $F = \{f\}$.

Množina pravidel R vychází z tabulky aminokyselin - Tabulka 1 Aminokyseliny a jejich kódy

$R = \{$

$sAUG \rightarrow qM,$

$qAUG \rightarrow qM,$

$qUUU \rightarrow qF, qUUC \rightarrow qF, qUUA \rightarrow qL, qUUG \rightarrow qL, qCUU \rightarrow qL, qCUC \rightarrow qL, qCUA \rightarrow qL, qCUG \rightarrow qL, qAUU \rightarrow qI, qAUC \rightarrow qI, qAUA \rightarrow qI, qGUU \rightarrow qV, qGUC \rightarrow qV, qGUA \rightarrow qV, qGUG \rightarrow qV, \ qUCU \rightarrow qS, qUCC \rightarrow qS, qUCA \rightarrow qS, qUCG \rightarrow qS, qAGU \rightarrow qS, qAGC \rightarrow qS, qCCU \rightarrow qP, qCCC \rightarrow qP, qCCA \rightarrow qP, qCCG \rightarrow qP, qACU \rightarrow qT, qACC \rightarrow qT, qACA$

→ qT, qACG→ qT, qGCU→ qA, qGCC→ qA, qGCA→ qA, qGCG→ qA, qUAU→ qY, qUAC→ qY, qCAU→ qH, qCAC→ qH, qCAA→ qQ, qCAG→ qQ, qAAU→ qN, qAAC→ qN, qAAA→ qK, qAAG→ qK, qGAU→ qD, qGAC→ qD, qGAA→ qE, qGAG→ qE, qUGU→ qC, qUGC→ qC, qUGG→ qW, qCGU→ qR, qCGC→ qR, qCGA→ qR, qCGG→ qR, qAGA→ qR, qAGG→ qR, qGGU→ qR, qGGC→ qR, qGGA→ qR, qGGG→ qR,

qUAA→ f, qUAG→ f, qUGA→ f, fAUG → qM,

fU→ f, fA→ f, fC→ f, fG→ f

}

Mějme vstupní mRNA sekvenci: *UUGCAAGAAUGGUAGCUGUUAAAGCUACAACCACAGAA GAAGAGACAGAAAUUCCCGCGAAAUAAGAGAUCGCGGAAUGUGUUGGAAGAGUAGAA UAGA.*

Tato sekvence mRNA obsahuje dva geny. První gen obsahuje protein se sekvencí *MVAVKATTTEEETEIPAK*. Pokud bychom tuto proteinovou sekvenci hledali na dostupných internetových databázích, kde nejvýznamnější z nich je veřejně dostupná databáze proteinů www.uniprot.org Vyhledáváním na tomto webu bychom zjistili, že sekvence patří bakteriálnímu proteinu Autolysin. Druhý z genů obsahuje protein s krátkou sekvencí *MCWKSRI*.

Zde je pouze zvýrazněn sběr a překlad jednotlivých proteinů. Kompletní sekvenci kroků skákajícího převodníku a jeho překladu na proteinové sekvence pak demonstruje přiložená webová aplikace (viz přílohy). Červeně jsou značeny nekódové sekvence a dalšími barvami pak jednotlivé proteiny:

UUGCAAGAAUGGUAGCUGUUAAAGCUACAACCACAGAAGAAGAGACAGAAAUUCCCGCGAAAUAAGAGAUCGCG GAAUGUGUUGGAAGAGUAGAAUAGA.

Převodník v sekvenci rozpozná 2 konkrétní proteiny:

AUGGUAGCUGUUAAAGCUACAACCACAGAA – MVAVKATTTEEETEIPAK
AUGUGUUGGAAGAGUAGAAUAG – MCWKSRI

Lze navržený převodník využít i při translaci v případě eukaryotních buněk? Bohužel nelze. U eukaryotních buněk mRNA obsahuje pouze jeden konkrétní gen. Avšak na rozdíl od prokaryotních buněk zde nebudou jednotlivé proteiny sekvenčně za sebou, ale získaná mRNA může obsahovat jeden protein, více proteinů, navíc mohou být i navzájem v sobě zanořeny jejich sekvence a detekce Methioninu (AUG) ještě nemusí znamenat výskyt proteinu. U eukaryotních buněk se musíme opřít o pravděpodobnostní model, který vychází ze statistik výskytů sekvencí, jakým jsou například Markovy pravděpodobnostní modely (HMM). [19] Jinak by se jednalo o pouhou naivní metodu, která by úspěšně detekovala protein jen v několika málo procentech případů. Zde tedy skákající převodník nebude mít praktický přínos.

6.4 Využití skákajících gramatik v oblasti DNA

Podobně, jako byla demonstrována aplikace skákajících lineárních gramatik pro účel generování sekvence DNA v článku [3], můžeme ke stejnému účelu využít elegantnější způsob v podobě JTC gramatik v 4. derivačním módu.

Taková gramatika by vypadala následovně: *JTC H = (G, R), G = ({S, A, B, C, D}, {a, t, c, g}, {(1) A→ aA, (2) B →tB, (3) C →cC, (4) D →gD, (5) S →ABCD, (5) A→a, (6) B →t, (7) C →c, (8) D→g}, S), R ⊆ {S, ABCD, aAtBcCgD, aAtBcg, atcCgD, cCgD, aAtB}.*

Terminály a, t, c, g u gramatiky symbolizují stejnojmenné nukleotidy A, T, C, G. Gramatika H vygeneruje sekvenci, kde je shodný počet A a T a shodný počet C a G.

Příklad derivace řetězce *A T C G C G A T:*

S $_4$⇒ ABCD $_4$⇒ aABCD $_4$⇒ aAtBCD $_4$⇒ aAtBcCD $_4$⇒ aAtBcCgD $_4$⇒ aAtBcgDc $_4$⇒ aAtBcgcg $_4$⇒ atBcgcga $_4$⇒ atcgcgat

JTC gramatiky v ostatních módech by mohly mít využití pro generování testovacích DNA sekvencí obsahující kódové proteinové sekvence. Takové výstupy by mohli být využity na optimalizaci a testování výše popsaných nástrojů rozpoznávající proteiny.

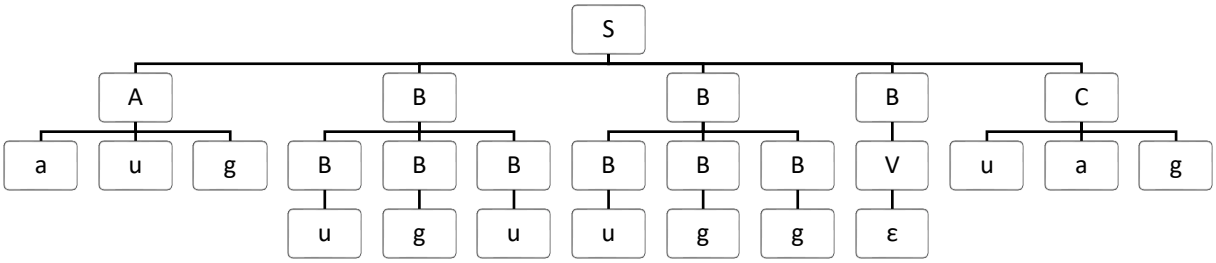
JTC v derivačním módu 3 může posloužit ke generování proteinových sekvencí. Gramatika bude generovat takové sekvence RNA, které začínají START kodónem AUG a končí jedním ze stop kodónu. Uvnitř obsahují platné kodony (trojice nukleotidů identifikující aminokyselinu).

Mějme *JTC H = (G, R), G = ({S, A, B, C, V, a, u, c, g}, {a, u, c, g}, {(1) S→ ABBBC, (2) B → BBB, (3) B → V, (4) B → c, (5) B → g, (6) B → a, (7) B → u, (8) A → aug, (9) C → uaa, (10) C → uag, (11) C → uga, (12) V → epsilon}, S), R ⊆ {S, ABBBC, aug(VB)*uaa, aug(VB)*uag, aug(VB)*uga, (VB)*}.*

Tato gramatika pomocí množiny R ošetří to, že uvnitř start kodonu a stop kodonu musí být vždy platné trojice nukleotidů (3 nukleotidy = 1 kodon). Algoritmus JTC v módu 2 následně zajistí, že počáteční kodon a jeden ze stop kodonu (Tabulka 1) vždy budou na hranicích řetězce. Pojďme provést demonstraci vygenerování krátké proteinové sekvence *AUGUGUUGGUAG* gramatikou *H:*

S $_3$⇒ ABBBC (1) $_3$⇒ augBBBC (8) $_3$⇒ augBBBuag (10)$_3$⇒ augBBBBBuag (2) $_3$⇒ augBBBBBBBuag (2) $_3$⇒ augBBBBBBVuag (3) $_3$⇒ auguBBBBBVuag (7) $_3$⇒ augugBBBBVuag (5) $_3$⇒ auguguBBBVuag (7) $_3$⇒ auguguuBBVuag (7) $_3$⇒ auguguugBVuag (5) $_3$⇒ auguguuggVuag (5) $_3$⇒ auguguugguag (12)

Pro větší názornost vyobrazme derivační strom, abychom viděli jednotlivé úrovně stromu:



Obr 6-6 derivování proteinu „auguguugg“

7 Závěr

V rámci této diplomové práce byly nejdříve obecně prozkoumány formální modely moderní teoretické informatiky. U klasických formálních modelů, tj. konečných automatů a gramatik platí, že vždy pracují se vstupem kontinuálně. Konkrétně konečné automaty čtou symboly na vstupní pásce v jednom směru postupně symbol po symbolu. Gramatiky při aplikaci pravidel umístí sekvenci na pravé straně pravidla přesně na původní pozici přepisované levé strany. Nemohou ji umístit na žádné jiné místo v rámci větné formy. V dnešní době již je tato vlastnost nedostačující vzhledem k požadavkům. Např. v rámci počítačových systémů máme scénář, kdy přečte proces p informaci z libovolného místa v paměti, zpracuje a zapíše na výstup. V rámci DNA řetězce nás zajímají některé konkrétní úseky a je příliš náročné v rozumném čase číst celou sekvenci DNA od začátku do konce symbol po symbolu. Podobných příkladů najdeme nejen ve světě IT nespočet. To je jasným ukazatelem toho, že je zapotřebí postupně zavádět nové formální modely, které se umí této potřebě přiblížit. Můžeme použít Turingovy stroje a obecné gramatiky, které mají vyjadřovací sílu shodnou se třídou rekurzivně spočetných jazyků. Nebo se nabízí na mnohé účely efektivnější varianta. Ponechat jednoduchost klasických konečných automatů a bezkontextových gramatik a přitom jim přidat schopnost neřešit úlohy kontinuálně, ale použít přitom skoky. Tím vznikají modely skákajících automatů a gramatik, kterými se tato práce z velké části zabývá. K již existujícím skákajícím gramatikám a automatům byly v rámci práce vytvořeny modifikace, které mají přínos zejména v oblasti bioinformatiky – zpracování DNA. Pokud uvážíme zachování jednoduchosti pravidel, tj. pravidla shodná s pravidly bezkontextové gramatiky, uvážíme model bezkontextových gramatik skákajících. Rozšířením bezkontextových gramatik o možnost skoků však paradoxně nezískáme vyšší vyjadřovací sílu, než mají „neskákající“ bezkontextové gramatiky. Např. nenajdeme žádnou bezkontextovou gramatiku, která by vygenerovala jazyk {a}*{b}*, který je jazykem regulárním.

Modifikace uvedená v této práci vychází z tzv. Tree controlled grammars (TCG). TCG mají pravidla ve tvaru pravidel bezkontextové gramatiky a navíc obsahují kontrolní regulární množinu R. Jazyk generovaný tímto typem gramatik L má charakteristiku, že derivační strom pro w □□ L má všechny úrovně vyjma poslední odpovídající některému z regulárních výrazů v množině R. Tree Controlled gramatiky mají vyjadřovací sílu shodnou s RE. Bylo vytvořeno několik skákajících módů TCG, které postupně dovolují větší volnost s používáním skoků. První z módů nedovolují žádné skoky nebo skoky omezené na konkrétní oblast větné formy nebo délku. Poslední skákající mód již dovoluje libovolné skoky a jejich libovolný počet. Inspirací k tvorbě důkazů k jednotlivým skákajícím módům byly skákající gramatiky s rozptýleným kontextem (JSCG). Tento typ gramatik vychází z paralelních gramatik s rozptýleným kontextem (SCG). Má společný tvar pravidel, avšak namísto omezením skrz kontrolní množinu používá omezení v podobě paralelního přepsání několika neterminálů v jednom derivačním kroku. TCG a SCG gramatiky mají v sekvenčním módu shodnou vyjadřovací sílu. Při

dokazování vyjadřovací síly zavedených skákajících TCG gramatik nazvaných JTC (jumping tree controlled) byl vytvořen algoritmus převádějící tvar gramatiky v TCG na SCG. Výsledkem zkoumání vyjadřovací síly JTC je, že všechny skákající módy JTC mají sílu shodnou s třídou RE. Dokonce i JTC v posledním módu, tj. JTC s neomezenými skoky. To bylo dokázáno skrze nalezení ekvivalentní sekvenční gramatiky k libovolné JTC gramatice v obecném skákajícím módu. U TCG gramatik tedy možné skoky nezvýší ani nesníží vyjadřovací sílu těchto gramatik. Možnost skoků však bude užitečná u některých praktických aplikací.

V další části práce byly vytvořeny modifikace skákajících automatů a převodníků. Modifikace rozšířila standardní definici skákajících automatů o priority pravidel. Zavedené priority pravidel částečně potlačují nedeterminismus skoků obecného skákajícího automatu na pásce. Byly zavedeny skákající automaty s prioritou pravidel (CJFA). Následně byl na základě této verze skákajících automatů vytvořen skákající převodník. Ten přesně simuluje popsaný praktický scénář s procesem čtoucím paměť v počítačovém systému. Převodník přečte z libovolného místa na pásce informaci, zpracuje pomocí převodní funkce a zapíše na výstup. Následně je ukázaná využitelnost zavedeného převodníku při provádění translace mRNA u prokaryotních buněk. Na vstup převodníku bude vložena sekvence mRNA vzniklá při DNA transkripci a následně se na výstup zapíše výsledek translace, tj. kódy jednotlivých proteinů. Tím však nekončí využití skákajících modelů v oblasti bioinformatiky. Můžeme pomocí nich detekovat přítomnost jednotlivých aminokyselin v DNA sekvenci, kontrolovat korektnost DNA sekvence, neboť ta se vyznačuje komplementaritou, konkrétními odvoditelnými chemickými vazbami mezi jednotlivými nukleotidy a shodným počtem typů nukleotidů A-T a C-G uvnitř sekvence. I skákajících gramatik se nám podaří využít v oblasti bioinformatiky např. pro generování DNA sekvencí obsahujících přítomné proteiny či konkrétní aminokyseliny.

Jaký je další možný směr a budoucí vývoj skákajících formálních modelů? Ještě stále existují některé otazníky týkající se porovnávání skákajících modelů s jejich sekvenčními prostředky. Tj. např. jestli $\mathscr{L}(\Gamma CFG, {}_{j}\Rightarrow) \subseteq \mathscr{L}(\Gamma CSG, {}_{j}\Rightarrow)$ je ostrá? Nebo jakou vyjadřovací sílu mají JSCG v obecném skákajícím módu. Zatím máme důkazy pouze ke skákajícím módům s konkrétními omezeními. V derivačním kroku nemohou pozice pravých stran pravidel jednotlivých přepisovaných neterminálů se vzájemně křížit. Velký potenciál skákajících modelů tkví především ve faktu, že jejich vyjadřovací síla není přímo totožná s některou třídou v Chomského hierarchii. U mnohých skákajících modelů se jedná o třídy jazyků, které leží někde na pomezí jednotlivých tříd jazyků dle Chomského. Dobře to zachycuje např ukázka tříd jazyků přijímanými skákajícími automaty na obr. 5.2.

Dalším potencionálně zajímavým směrem je uvážit možnost skoků u dalších již zmodifikovaných formálních modelů. Jaký je další potenciál skoků u formálních prostředků? Když se vrátíme do oblasti bioinformatiky, kde rozhodně skákající jazykové modely mají velký potenciál, tak kromě uvedených aplikací v rámci této práce ještě existují mnohé další. Při hlubším zkoumání ale narážíme na určité limity. Když se vrátíme např. k detekci proteinů, tak zavedený model skákajícího převodníku dokáže detekovat a přeložit sekvence proteinů z mRNA u prokaryotních buněk, tj. buněk jednoduchých

organismů, převážně bakterií. U tohoto typu buněk jsou však jednotlivé proteiny ve vzniklé mRNA sekvenčně za sebou, nijak se navzájem neprolínají, nekříží mezi sebou. Navíc neobsahují prázdná místa mezi částmi sekvence nebo místa, která je nutná před překladem ze sekvence vystřihnout – iotrony, extrony. To však neplatí pro buňky eukaryotní, kde přesně k tomuto jevu docházet může. Tyto Buňky jsou pro nás bližší, neboť se jedná o buňky složitějších organismů, včetně našich lidských buněk. Jak se projevují prázdná místa? Kde bývají nejčastěji umístěná? Jak rozpoznáme sekvenci, kterou je nutno před překladem proteinu vystřihnout? Jak můžeme zkontrolovat správnost vstupu, zda neobsahuje chyby? A konečně jak vlastně můžeme prohlásit, že nalezená sekvence je opravdu platná proteinová sekvence? Na podobné otázky se špatně hledá odpověď. V oblasti biologie nejsou nějaká striktní přesně daná pravidla. Většina poznatků, na kterých jsou založeny implementace nástrojů, vychází z pravděpodobnosti / statistiky. Např. víme, že většina proteinů je dlouhá alespoň 21 kodonů, kratší sekvence s největší pravděpodobností nebudou proteinové sekvence. Jsou ale i výjimky, např. v případě buněk imunitního systému, které mají kratší proteinové sekvence. Nebo další příklad – pro proteiny jsou typické některé opakující se sekvence nukleotidů, výskyt konkrétních aminokyselin apod., které zvyšují pravděpodobnost, že se jedná o protein. Takovéto poznatky již biologie má. Avšak skákající formální modely ani Turingovy stroje s nimi nijak nepracují. Abychom tedy mohli využít skákající modely na složitějších praktických příkladech a jejich přesnost zvýšili, museli bychom je ještě rozšířit o schopnost pracovat s pravděpodobnostními tabulkami, tak jako to dělají hojně používané modely v bioinformatice – Markovy pravděpodobnostní modely (HMM). Následně bychom mohli modely užít i na složitějších aplikacích, jako např. již zmíněné zpracování eukaryotních buněk nebo při hledání místa pro provedení transkripce.

Literatura

[1] *TID: Moderní teoretická informatika* [online]. b.r. [cit. 2019-01-09]. Dostupné z: http://www.fit.vutbr.cz/~meduna/work/doku.php?id=lectures:phd:tid:tid/

[2] *Materiály k předmětu IFJ: Formální jazyky a překladače - doplňující informace* [online]. b.r. [cit. Poslední aktualizace: 17. června 2017]. Dostupné z: www.fit.vutbr.cz/study/courses/IFJ/public/materials/

[3] ZBYNĚK, KŘIVKA a Alexander MEDUNA. Jumping Grammars. *International Journal of Foundations of Computer Science* [online]. 2015, **26**(6), 709-731 [cit. 2018-12-29]. DOI: 10.1142/S0129054115500409. Dostupné z: https://www.worldscientific.com/doi/abs/10.1142/S0129054115500409

[4] MEDUNA, Alexander a Ondřej SOUKUP. Jumping Scattered Context Grammars. *Fundamenta Informaticae* [online]. 2014, **2017**(152), 51-86 [cit. 2019-01-02]. DOI: 10.3233/FI-2017-1512. Dostupné z: https://content.iospress.com/articles/fundamenta-informaticae/fi1512

[5] MEDUNA, Alexander. *Automata and languages: theory and applications*. London: Springer, 2000. ISBN 18-523-3074-0.

[6] MEDUNA, Alexander. *Formal languages and computation: models and their applications*. 1. New York: CRC Press, 2014, xix, 295 s. ISBN 978-1-4665-1345-7.

[7] ZEMEK, Petr. Substituce a morfismy. *Publications.petrzemek.net* [online]. b.r. [cit. 2018-11-30]. Dostupné z: https://publications.petrzemek.net/articles/PZ_-_Substituce.a.Morfismy.pdf

[8] MEDUNA, Alexander. *Modern language models and computation: theory with applications*. New York, NY: Springer Science Business Media, 2017. ISBN 978-331-9630-991.

[9] MEDUNA, Alexander a J. TECHET. *Scattered context grammars and their applications*. 1. Billerica, MA: WIT Press, 2009. ISBN 978-184-5644-260.

[10] CULIK, K. a MAURER. *H.A. Computing*. Springer-Verlag, **1977**(19129). DOI: https://doi.org/10.1007/BF02252350. ISSN 0010-485X.

[11] ČEŠKA, M., T. VOJNAR, A. SMRČKA a A. ROGALEWICZ. *Teoretická informatika TIN: Studijní text*. Brno, 2018. Studijní text. FIT VUT v Brně.

[12] OŠMERA, Lubomír. *Nové verze skákajících automatů*. Brno, 2016. Bakalářská práce. FIT VUT v Brně. Vedoucí práce Alexander Meduna.

[13] JUMPING FINITE AUTOMATA. *International Journal of Foundations of Computer Science* [online]. 2012, **23**(07), 1555-1578 [cit. 2019-04-06]. DOI: 10.1142/S0129054112500244. ISSN 0129-0541. Dostupné z: http://www.worldscientific.com/doi/abs/10.1142/S0129054112500244

[14] ZVELEBIL, Marketa a Jeremy BAUM. *Understanding bioinformatics*. 1. New York: Garland Science, 2008. ISBN 08-153-4024-9.

[15] PAUL, Georghe, Arto SALOMAA a Grzegorg ROZEMBERG. *DNA computing: New computing paradigms*. 1. Berlin: Springer, 1998. ISBN 3-540-64196-3.

[16] Otevrenaveda.cz. *Nezkreslená věda II: 8. Proteosyntéza - od DNA k proteinu* [online]. Akademie věd ČR, 2015 [cit. 2019-03-28]. Dostupné z: https://www.youtube.com/watch?v=fqWs1aM7BQs

[17] CLAVERIE, Jean-Michel a Cedric NOTREDAME. *Bioinformatics for dummies*. 2. Hoboken, N.J.: Wiley Publishing, 2003. ISBN 07-645-1696-5.

[18] Molekulární biologie - studijní materiály, 4. přednáška. In: *Is.muni.cz* [online]. Brno: fi.muni.cz, 2015 [cit. 2019-04-01].

[19] MARTÍNEK, Tomáš. *Bioinformatika - materiály k předmětu: Rozpoznávání genů* [online]. In: . b.r. [cit. 2019-04-01]. Dostupné z: IS FIT VUT

Příloha A: Manuál k programu

Popis programu

Přiložený program obsahuje ukázku aplikace skákajících převodníků při translaci mRNA u prokaryotních buněk. Je demonstrována sekvence přechodů skákajícího převodníku popsaného v kapitole 6.3. mRNA u prokaryotních buněk obsahuje proteiny sekvenčně za sebou, navzájem se nekříží, můžeme tedy převodník plnohodnotně využít. To ale naopak neplatí pro eukaryotní buňky, kde se mohou jednotlivé sekvence proteinů vzájemně překrývat, start kodon AUG ještě nemusí potvrzovat sekvenci proteinu, musíme dále zvažovat např. délku nasbírané sekvence atd. Jediná možnost v případě eukaryotních buněk, jak získat platné sekvence proteinů je vyhledat všechny podsekvence začínající start kodonem AUG a končící jedním ze stop kodonů. Tuto práci demonstruje Turingův stroj hledající všechny takové podsekvence. Následně je nutno kriticky přistoupit k vyhledaným sekvencím a z nich vyloučit sekvence příliš krátké, tj. kratší než 21 kodonů, které s nejvyšší pravděpodobností nebudou sekvence proteinů. Delší sekvence již mohou být potencionálními proteiny. Potvrzení výsledku by však vyžadovalo kontrolu v biologických databázích, pro začátek bychom mohli použít např. generovanou databázi uniprot.org.

Jak program spustit a ovládat

Program je možno spustit na webovém serveru obsahujícím interpret php verze 5.0 a vyšší. Alternativně je projekt dostupný na http://www.stud.fit.vutbr.cz/~xosmer01/. Hned po spuštění projektu (načtení souboru index.php) se zobrazí manuál k programu. Následně v horním menu vybereme jednu z variant:

- Detekce pomocí TS – jedná se o demonstraci zpracování sekvence pomocí Turingova stroje
- Detekce skákajícím převodníkem – demonstrace translace mRNA pomocí skákajícího převodníku.
- Manuál – jedná se o domovskou stránku, manuál k programu

Postup práce

1. V textovém poli se již nachází ukázka RNA sekvence z kapitoly 6.3, obsahující protein Autolysin. Pokud chcete vidět její zpracování, zvolte *Zpracuj...*
2. Pokud chcete zadat sekvenci vlastní, pak zvolte *Vyčistit pole* a zadejte libovolnou sekvenci obsahující nukleotidy A, T, C, G, respektive namísto T písmeno U. Sekvence nesmí obsahovat žádné jiné znaky než vyjmenované a bílé znaky oddělující jednotlivé části sekvence. Při zadávání je tolerováno zadání pomocí malých písmen i velkých písmen, případně libovolné kombinacevelkých a malých písmen.
3. Zvolte *Zpracuj...*

Výstupy

Detekce pomocí TS:

Na výstupu se ukáže vámi zadaná sekvence RNA velkými písmeny očištěná o veškeré nadbytečné bílé znaky. V případě, že jste na vstup zadali sekvenci DNA (obsahující T), jsou všechna T nahrazena za U. Dále ukazuje tabulku. V levém sloupci tabulky jsou jednotlivé nalezené podsekvence začínající start kodonem AUG a končící stop kodonem. V pravém sloupci je vyhodnocení těchto nálezů, zda jednotlivé sekvence mohou být potencionálním proteinem. V případě, že ano, ukáže se proteinová sekvence odpovídající k vyhledané podsekvenci - využijte pro její čtení Tabulka 1. V případě, že ne, obdržíte důvod, proč sekvence nemůže nebo pravděpodobně není proteinem. Takovým důvodem bude buďto nalezení neplatné sekvence nebo příliš krátké sekvence.

Detekce skákajícím převodníkem:

Na výstupu se ukáže vámi zadaná sekvence RNA velkými písmeny očištěná o veškeré nadbytečné bílé znaky. Znaky T jsou nahrazeny znaky U. Tabulka ukazuje konfiguraci v jednotlivých krocích převodníku. Levý sloupec zobrazuje podobu výstupní pásky – tj. překlad. Pravý sloupec ukazuje vstupní pásku. Pozice čtecí hlavy a aktuální stav je značen pro lepší přehlednost červeným tučným písmem. Pokud je přijat stop kodon, na výstupní pásku je namísto symbolu „blank“ zapsán pro pohodlnější čtení znak #.

Příklad výstupu:

Jako jednoduchý příklad demonstrující výstupy převodníku zadáme krátkou sekvenci *AAAAAAAUGCCCCCCUAAAAUGUUUUAAUUUAUGUUUUUUUAA*, která obsahuje tři potencionální proteiny. (reálné proteiny jsou samozřejmě několikanásobně delší) Výstupní tabulka bude mít podobu jako tabulka níže. Levý sloupec ukazuje postupný zápis na výstupní pásku a v pravém sloupci můžeme vidět postupné zpracování vstupu.

Výstupní páska	Vstupní páska
M	AAAAAA**s**AUGCCCCCCUAAAAUGUUUUAAUUUAUGUUUUUUUAA
MP	AAAAAA**q**CCCCCCUAAAAUGUUUUAAUUUAUGUUUUUUUAA
MPP	AAAAAA**q**CCCUAAAAUGUUUUAAUUUAUGUUUUUUUAA
MPP#	AAAAAA**q**UAAAAUGUUUUAAUUUAUGUUUUUUUAA
MPP#M	AAAAAAA**f**AUGUUUUAAUUUAUGUUUUUUUAA
MPP#MF	AAAAAAA**q**UUUUAAUUUAUGUUUUUUUAA

MPP#MF#	AAAAAAA**q**UAAUUUAUGUUUUUUUAA
MPP#MF#M	AAAAAAAUUU**f**AUGUUUUUUUAA
MPP#MF#MF	AAAAAAAUUU**q**UUUUUUUAA
MPP#MF#MFF	AAAAAAAUUU**q**UUUUAA
MPP#MF#MFF#	AAAAAAAUUU**q**UAA
MPP#MF#MFF#	**f**AAAAAAAUUU
MPP#MF#MFF#	**f**AAAAAAUUU
MPP#MF#MFF#	**f**AAAAAUUU
MPP#MF#MFF#	**f**AAAAUUU
MPP#MF#MFF#	**f**AAAUUU
MPP#MF#MFF#	**f**AAUUU
MPP#MF#MFF#	**f**AUUU
MPP#MF#MFF#	**f**UUU
MPP#MF#MFF#	**f**UU
MPP#MF#MFF#	**f**U

Příloha B: Obsah CD

CD přiložené k diplomové práci obsahuje tyto soubory a složky:

- **src** – složka obsahující php skripty webové aplikace: index.php, turing.php, jumpingtranscuder.php, funkce.php
- **doc** - složka obsahující text práce ve formátu PDF a docx.