VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ VERZE SKÁKAJÍCÍCH AUTOMATŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUBOMÍR OŠMERA
AUTHOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ VERZE SKÁKAJÍCÍCH AUTOMATŮ

NEW VERSIONS OF JUMPING AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUBOMÍR OŠMERA
AUTHOR

VEDOUCÍ PRÁCE prof. RNDr. ALEXANDER MEDUNA
SUPERVISOR

BRNO 2016

Abstrakt

Cílem této bakalářské práce je návrh a výzkum nových verzí skákajících automatů, konkrétně se jedná o verze obecných skákajících automatů s jasně specifikovanými kritérii směru a velikosti skoku. Práce zkoumá jejich sílu v porovnání s ostatními typy automatů a odhaluje ekvivalentní modely gramatik. Důvodem vypracování této práce je výzkum a snaha o vylepšení některých vlastností obecného skákajícího automatu. Následně je v práci zváženo využití tohoto formálního prostředku za účelem zpracování určitého typu jazyků a využití v průběhu syntaktické analýzy.

Abstract

The main goal of this thesis is introduction and investigation of extended version of jumping automata. These versions are specified by strictly size and direction of jump. This thesis examine their power and perform comparison with other automata types. Also there are shown equivalent grammar models. Main motivation for this thesis are research and effort to improve some features of general jumping finite automata. This work consider using this automata types for specific language families and syntactic analysis process.

Klíčová slova

Automat, skákající, gramatika, k-left, k-right, startující stav, syntaktická analýza, jazyk, modifikované konečné automaty.

Keywords

Automaton, jumping, grammar, k-left, k-right, starting state, syntactic analysis, language, modified finite automata.

Citace

Ošmera Lubomír: Nové verze skákajících automatů, bakalářská práce, Brno, FIT VUT v Brně, 2016

Nové verze skákajících automatů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Prof. RNDr. Alexandra Meduny, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lubomír Ošmera
Datum 12. 5. 2016

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.

© Lubomír Ošmera, 2015/2016
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů..

Obsah

1 Úvod ... 2
1.1 Motivace ... 2
1.2 Princip skákajících automatů ... 2
1.3 Organizace ... 3
1.4 Modifikace ... 4
1.5 Použité značení ... 5
2 Základní pojmy a definice ... 6
2.1 Definice ... 6
2.1.1 Základní definice ... 6
2.1.2 Složitější definice a definice vycházející ze základních definic ... 7
2.2 Metodika důkazů ... 11
3 Síla skákajících konečných automatů ... 13
3.1 Rozbor síly skákajících automatů ... 13
3.2 Grafické znázornění vztahu automatů a rodin jazyků ... 15
3.3 Kam až sahá síla nově navržených automatů ... 15
3.3.1 Porovnání síly s GJFA ... 16
3.4 Souhrn poznatků ... 17
3.5 Grafické znázornění $_{\text{k-left}}$ GJFA v množinách jazyků ... 18
4 Zpracování derivačních stromů aneb užití k-left a k-right automatů při syntaktické analýze ... 19
4.1 Kde je zádrhel ... 19
4.2 Existuje nějaká možnost? ... 20
4.3 Výsledné automaty a ukázka jejich práce ... 21
5 Možné zádrhely a problémy při užití k-GJFA v praxi ... 23
5.1 Pozice pro startující stav ... 23
5.2 Nedeterminismus při hledání počátečního stavu ... 24
6 Důkazy algoritmů pro tvorbu skákajících automatů a gramatik ... 26
6.1 Konstrukce automatu ... 26
6.2 Konstrukce gramatiky ... 28
6.3 K-right automaty a gramatiky ... 30
7 Závěr ... 31

1 Úvod

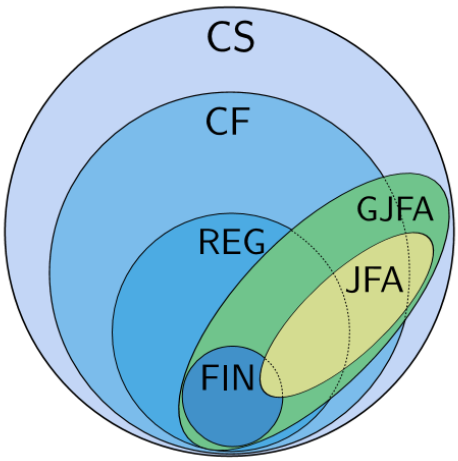
1.1 Motivace

Co může být motivem k zavedení modifikací obecných konečných automatů? Prvním kritériem je síla daného typu konečného automatu, tj. jaké rodiny jazyků dokáže daný automat zpracovávat. Obecné konečné automaty pokrývají zpracování regulárních jazyků (*REG*). Kromě *REG* existují ještě další složitější rodiny jazyků – bezkontextové (*CF*) a kontextové jazyky (*CS*), se kterými si konečné automaty neporadí. Omezujícím limitem bránícím zpracování ostatních rodin jazyků je, že nemohou startovat čtení vstupní pásky ze kteréhokoliv místa. Konečný automat zpracovává vstup v jednom směru, tj. zleva doprava a startuje zpracovávání od nejlevějšího symbolu. Pokud se dostane při zpracovávání do situace, že nemá pro vstupní symbol žádné pravidlo, které by uplatnil, jeho práce končí neúspěšně, tj. daný jazyk není přijímán konečným automatem.

Uvažme ale např. situaci, kdy máme na vstupu řetězec *bc* a množina pravidel konečného automatu obsahuje pravidla $<Q>c \rightarrow <P>$, $<P>b \rightarrow <F>$. Uvažujme počáteční stav $<Q>$ a koncový stav $<F>$. Nemáme k dispozici pravidlo pro kombinaci aktuálního stavu $<Q>$ a vstupního symbolu *b***,** tudíž bychom prohlásili jazyk za nepřijímaný tímto automatem. Jenže vstupní symbol *b* máme v pravidlech, a sice v kombinaci se stavem $<P>$**,** do kterého bychom se přitom mohli dostat přečtením následujícího symbolu *c*. Jinými slovy, kdybychom přeskočili znak *b* a ve chvíli, kdy se pomocí uplatnění pravidla $<Q>c \rightarrow <P>$ dostanu do stavu $<P>$, dokážu přečíst i zbývající symbol *a*. Tudíž přečtu celou pásku a prohlásím jazyk za jazyk přijímaný tímto konečným automatem.

1.2 Princip skákajících automatů

Již existuje model automatu, který dokáže potlačit některé negativní vlastnosti a limity obecných konečných automatů? Takový model byl navržen prostřednictvím vědeckého článku. [1] Jedná se o tzv. skákající automaty (*JFA* a *GJFA*). Jejich síla, tj. které rodiny dokáže tato modifikace zpracovávat, je zobrazena graficky na Obrázek 1-1. Specifikaci *JFA* a *GJFA* popisují Definice 18 a 19.

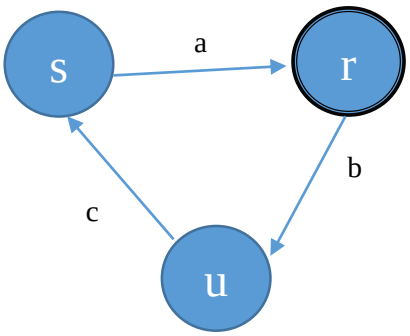


Obrázek 1-1 Síla skákajících automatů [2, s. 11]

Rozdíl mezi zpracováváním vstupu konečných automatů a navržené modifikace, tzv. skákajících automatů znázorňuje *Příklad 1*, kde jsou základní přednosti skákajících automatů představeny a vysvětleny.

Příklad 1

Mějme jednoduchý konečný automat $M = (\{<S>, <R>, <U>\}, \{a, b, c\}, R, <S>, \{<R>\})$, kde množina $R = \{<S>a \rightarrow <R>, <R>b \rightarrow <U>, <U>c \rightarrow <S>\}$ (Obrázek 1-2). Pokud bychom měli posoudit, zda řetězec *abca* bude přijat tímto automatem, pak odpověď zní ano. Jenže stačí, abychom pouze zaměnili dva znaky tohoto řetězce, například kdyby místo *abca* bylo na vstupní pásce *acba*, v tu chvíli automat nebude schopen vstupní pásku zpracovat.



Obrázek 1-2 Model konečného automatu

Nyní proběhne demonstrace zpracování vstupní pásky skákajícím konečným automatem. Ten má povoleno zpracovávat vstup odkudkoliv a „přeskakovat“ znaky, pro něž nemáme ve spojitosti s aktuálním stavem k dispozici žádná pravidla. Síla automatu se mnohonásobně zvýší. Konkrétní zpracování vstupu by vypadalo následovně:

$<S>acba \Rightarrow <R>cba$. Automat M nemá pravidlo pro stav $<R>$ se vstupem c. Přeskočíme tedy tento znak (provedeme skok vpravo) a podíváme se, zda bude možno nyní některé z pravidel uplatnit. Pokud ne, budeme pokračovat dalšími skoky. Vyhovující konfigurace, u níž můžeme pokračovat dál ve zpracovávání pásky, je $c<R>ba$.

$c<R>ba \Rightarrow c<U>a$. Opět stejná situace. Chybí pravidlo pro kombinaci stavu $<U>$ a vstupu c. Přeskočíme tentokrát směrem vlevo o jeden znak. Výsledná konfigurace bude $<U>ca$.

$<U>ca \Rightarrow <S>a$.

Následně dokončíme zpracování vstupní pásky přečtením zbývajícího symbolu.

$<S>a \Rightarrow <R>$.

Přečetli jsme kompletně vstupní pásku, proto můžeme prohlásit, že řetězec *acba* je přijímán touto modifikací KA.

Cílem této práce je návrh nových verzí skákajících automatů, přesněji řečeno jejich omezujících verzí. Obecný návrh skákajících automatů omezíme směrem jeho skoku, maximálním počtem zpracovávaných znaků a četností skoků daného automatu během zpracovávání vstupní pásky.

1.3 Organizace

Tato práce představí modifikace obecných skákajících automatů, tzv. k-left a k-right skákající automaty. Pro zkrácení bude dále v textu místo k-left a k-right skákající automaty užito označení *k-GJFA*. Omezení těchto verzí spočívá v počtu znaků, které přeskakují. Jejich skoky mají neustále

stejný směr, v případě k-left se jedná o směr vlevo a v případě k-right automatů o směr vpravo. Při každém kroku *k-GJFA* je proveden příslušný skok, tj. na rozdíl od obecných automatů si *k-GJFA* nemůže zvolit, zda skok z důvodu potřeby provede nebo ne. Jak tyto úpravy ovlivní sílu skákajících automatů a jaké modely jazyků tento typ automatu dokáže zpracovat? Je možno tento automat použít i při syntaktické analýze nebo náročnějších analýzách řetězců? Jaké jsou jeho zádrhely a naopak přednosti? Na všechny tyto otázky v průběhu práce bude zodpovězeno.

Úvodní kapitoly pokrývají základní definice a pojmy potřebné pro popis nových verzí skákajících automatů. Rovněž i seznámení s principy jejich funkce a metodiky důkazů směřující k úvahám, potvrzení jejich síly a dalších vlastností.

Následuje jádro této práce, a sice porovnávání a zkoumání síly *k-GJFA* v porovnání s ostatními známými typy automatů. Dokáží vykrýt i slabiny obecných skákajících automatů? Velkou slabinou *GJFA* a *JFA,* která bude rozebrána v kapitole 3, je až přílišná obecnost těchto automatů. Pokud chceme najít formální prostředek, který popisuje konkrétní jazyk, hledáme jednoduše řečeno model popisující právě daný jazyk. Přílišná rozmanitost možností obecných skákajících automatů při zpracování vstupu mnohdy způsobí, že nedokážeme najít model popisující právě konkrétní jazyk, ale některou s jeho nadmnožin, což je nevyhovující. Může tedy omezení ve formě pevně daného skoku v některých ohledech zvýšit sílu skákajících automatů? Nebo se jedná čistě o podmnožinu obecných skákajících automatů, která může být využívána pro nějaký specifický účel v teorii formálních jazyků? Úvahy a příklady z kapitol zabývajících se silou navržených automatů budou zakončeny grafickým znázorněním vztahu rodin jazyků ke *k-GJFA*. Obrázek bude doplněn o souhrnnou tabulku porovnávající sílu jednotlivých typů automatů.

Jedna samostatná kapitola bude věnována úvahám o možných využitích při syntaktické analýze. Dokáží pomoci tyto automaty zpracovat řetězce vygenerované bezkontextovými gramatikami a zastoupit tak roli zásobníkových automatů, tj. modelu pro popis bezkontextových jazyků využívaným při syntaktické analýze? Pokud ne, jaké jsou limity a za jakého předpokladu můžeme tyto automaty pro daný účel využít?

Pokud navrhujeme novou verzi automatu a chceme provést její následnou implementaci, je možné, že se vyskytnou některé zádrhely při tvorbě algoritmů. Některé z neduhů budou rozebrány v kapitole 5. Rovněž zde zvážíme možný budoucí vývoj skákajících automatů, ať už z pohledu odstranění chyb, tak i z pohledu růstu jejich síly.

Poslední kapitola poskytne důkazy ekvivalence mezi skákajícími automaty a příslušnými gramatikami. Práci $_{k\text{-}left}$ *GJFA* na různě modifikovaných vstupech je možno vyzkoušet na přiloženém software.

1.4 Modifikace

Pojďme si pro úplné zavedení a vytvoření správného obrazu o *k-GJFA* provést demonstraci a vysvětlení jejich funkčnosti. Další důležité algoritmy, souvislosti s gramatikami a patřičné důkazy se nachází v poslední kapitole.

Algoritmus I. k-left skákající automat

1. Najdi pozici pro startující stav podle algoritmu IV.
2. Přijmi patřičné symboly dle pravidla na vstupní pásce nacházející se vpravo od aktuálního stavu.
3. Skoč vlevo o *k* symbolů.
4. Opakuj kroky 2-3, dokud nepřijmeš celý vstup.

Algoritmus II. k-right skákající automat

1. Najdi pozici pro startující stav podle algoritmu IV.
2. Přijmi patřičné symboly dle pravidla na vstupní pásce nacházející se vlevo od aktuálního stavu.
3. Skoč vpravo o *k* symbolů.
4. Opakuj kroky 2-3, dokud nepřijmeš celý vstup.

Původní návrh k-right skákajícího automatu byl takový, že bude stejně jako k-left verze automatu číst vstupy zleva doprava a provádět skok vpravo. Takto navržená verze však vynechává symboly umístěné na pravé straně od startujícího stavu.

Příklad 2

Demonstrace krok po kroku zpracování řetězce *cdabe* k-left skákajícím automatem $M = (\{<S>, <Q>, <U>, <Z>\}, \{a, b, c, d, e\}, R, <S>, \{<Q>\})$, množina $R = \{<S>e \rightarrow <U>, <U>ab \rightarrow <Q>, <Q>cd \rightarrow <Z>\}$, $k = 2$.

1. Najdeme pozici pro startující stav, postavím stav před symbol *e*. Výsledná konfigurace: *cdab<S> e*.
2. Provedeme pravidlo $<S>e \rightarrow <U>$ a skočíme o 2 znaky vlevo: *cd<U>ab*.
3. Provedeme pravidlo $<U>ab \rightarrow <Q>$ a skočíme o 2 znaky vlevo: *<Q>cd*.
4. Provedeme poslední z pravidel $<Q>cd \rightarrow <Z>$, skončíme tak v koncovém stavu, celý vstupní řetězec je přijat.

Příklad 3

Demonstrace krok po kroku zpracování řetězce *ecdab* k-right skákajícím automatem $M = (\{<S>, <Q>, <U>, <Z>\}, \{a, b, c, d, e\}, R, <S>, \{<Q>\})$, množina $R = \{<S>e \rightarrow <U>, <U>dc \rightarrow <Q>, <Q>ba \rightarrow <Z>\}$, $k = 2$.

1. Najdeme pozici pro startující stav, postavím stav za symbol *e*. Výsledná konfigurace: *e<S>cdab*.
2. Provedeme pravidlo $<S>e \rightarrow <U>$ a skočíme o 2 znaky vpravo: *cd<U>ab*.
3. Provedeme pravidlo $<U>dc \rightarrow <Q>$ a skočíme o 2 znaky vpravo: *ab<Q>*.
4. Provedeme poslední z pravidel $<Q>ba \rightarrow <Z>$, skončíme tak v koncovém stavu, celý vstupní řetězec je přijat.

1.5 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)
- Stavy budou zobrazeny v špičatých závorkách značené velkými písmeny, např. *<S>*.
- Neterminály budou značeny velkým písmenem.
- Řetězce a symboly budou značeny malými písmeny.

Základní pojmy a definice

Pro vytváření nových modelů pro popis jazyků je důležité zavést základní pojmy a definice, na kterých následně budou postaveny složitější definice popisující výše zmíněné skákající automaty a pro ně ekvivalentní gramatiky. Základní definice jsou převzaty ze zdroje [3], [4] a [5]. Složitější definice související s obecnými skákajícími automaty jsou převzaty z [1], případně z [6].

1.6 Definice

1.6.1 Základní definice

Definice 1 (abeceda)

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

Definice 2 (řetězec)

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

Definice 3 (délka řetězce)

Nechť x je řetězec nad abecedou $\sum$, délka řetězce x, značeno |x|, je definována:
Pokud $x = \varepsilon$, pak $|x| = 0$.
Pokud $x = a_1, a_2, \ldots, a_n$, pak $|x| = n$ pro $n \geq 1$ a $a_i \in \sum$ pro všechna $i = 1, \ldots, n$.

Definice 4 (konkatenace)

Nechť x a y značí řetězec nad abecedou $\sum$. Konkatenace x a y, označena jako xy, je řetězec získaný přidáním řetězce y k řetězci x.
Pro každé $w \in \sum^*$, $w\varepsilon = \varepsilon w = w$.

Definice 5 (mocnina řetězce)

Nechť x představuje řetězec nad abecedou $\sum$. Pro $i \geq 0$, i-tá mocnina řetězce x, značeno x^i, je definována:
$x^0 = \varepsilon$.
Pro $i \geq 1$: $x^i = xx^{i-1}$.

Definice 6 (reverzace řetězce)

Nechť x představuje řetězec nad abecedou $\sum$. Reverzace řetězce x, značeno reversal(x), je definována:
Pokud $x = \varepsilon$, pak $\text{reversal}(\varepsilon) = \varepsilon$.
Pokud $x = a_1 \ldots a_n$, pak $\text{reversal}(a_1 \ldots a_n) = a_1 \ldots a_n$ pro $n \geq 1$ a $a_i \in \sum$ pro všechna $i = 1, \ldots, n$.

Definice 7 (jazyk)

Podle teorie formálních jazyků je pojem jazyk definován jako množiny posloupností obsahujících symboly.

Nechť $\sum^*$ značí množinu všech řetězců nad $\sum$. Každá podmnožina $L \in \sum^*$ je jazyk nad $\sum$.
Pouze v případě, že jazyk L obsahuje konečný počet řetězců, je konečný, jinak je jazyk nekonečný.

Definice 8 (mocnina jazyka)

Nechť L je jazyk nad abecedou $\sum$. Pro $i \geq 0$, i-tá mocnina jazyka L, označena L^i, je definována:
$L^i = \{\varepsilon\}$ pro $i = 0$.
$L^i = LL^i$ pro $i \geq 1$.

Definice 9 (iterace jazyka)

Nechť L je jazyk nad abecedou $\sum$. 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 10 (regulární výrazy)

Zjednodušeně lze regulární výrazy považovat za nadstavbu nad jazyky.
Nechť $\sum$ je abeceda. Regulární výrazy (zkráceně RE) nad $\sum$ a jazyky, které značí, jsou rekurzivně definovány takto:
$\emptyset$ je regulární výraz označující prázdnou množinu.
ε je RE značící jazyk $\{\varepsilon\}$.
a, kde $a \in \Sigma$, je RE značící jazyk $\{a\}$.
(r^*) je RE značící jazyk L_r^*.

Definice 11 (regulární jazyk)

Nechť L je jazyk nad $\sum$, pak L můžeme prohlásit za regulární jazyk, zkráceně REG, v případě, že existuje regulární výraz r, který tento jazyk značí.

1.6.2 Složitější definice a definice vycházející ze základních definic

Definice 12 (obecná gramatika)

Obecná gramatika (zkráceně GG) je čtveřice, $G = (V, T, P, S)$, kde V je abeceda, $T \subseteq V$, P je konečná množina relací z $V^* - T^*$ do V^*, a $S \in V - T$. Častěji pro lepší přehlednost místo relačního zápisu $(x, y) \in P$ nebo x P y, zapisujeme $x \rightarrow y$ a nazýváme tento výraz pravidlem. Z toho vyplývá, že P je množina pravidel nad gramatikou G.

Definice 13 (Jazyk generovaný obecnou gramatikou)

Nechť $G = (V, T, P, S)$ představuje GG a r je relace nad V^*. L(G, r) je jazyk, který je generován gramatikou G prostřednictvím pravidel r z množiny P:
$L(G, r) = \{ x \mid x \in T^*, S\, r^*\, x\}$.
Nechť Γ_{GG} představuje množinu všech GG a $X \subseteq \Gamma_{GG}$. Pak $\Phi(\boldsymbol{X}, r) = \{ L \mid L(G, r), G \in \boldsymbol{X}\}$.

Definice 14 (derivace)

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

Definice 15 (bezkontextová gramatika)

Nechť G je GG.
G je bezkontextová gramatika (zkráceně CFG), právě tehdy, když platí pro každé x $\rightarrow$ y $\in$ P, platí že **x $\in$ N**.

Definice 16 (bezkontextový jazyk)

Nechť G = (V, T, P, S) značí CFG, pak jazyk generovaný CFG G, značeno CF, definujeme jako:
CF = {w: w $\in T^*$, S $\Rightarrow^*$ w}.

Definice 17 (lineární gramatika)

G je lineární gramatika (zkráceně LG), pokud G je CFG a pro každé x $\rightarrow$ y $\in$ P platí, že y $\in T^*NT^* \cup T^*$.
Nechť k $\geq$ 0. G je a k-left lineární gramatika ($_{k\text{-left}}$LG zkráceně), pokud G je CFG a pro každé x $\rightarrow$ y $\in$ P platí, že y $\in T^kNT^* \cup T^*$.
G je k-right lineární gramatika (zkráceně $_{k\text{-right}}$LG), pokud G je CFG a pro každé x $\rightarrow$ y $\in$ P platí, že y $\in T^*NT^k \cup T^*$.

Definice 18 (obecné skákající automaty)

Obecný skákající automat, zkráceně GJFA, je pětice, M = (Q, Σ, R, s, F), kde Q je konečná množina stavů, $\sum$ značí vstupní abecedu, R je konečná množina pro kterou platí R $\subseteq$ Q $\times$ Σ^* $\times$ Q, s $\in$ Q představuje počáteční stav, a F značí konečnou množinu koncových stavů.
Prvky množiny R jsou nazývány pravidla automatu M. Místo relačního zápisu (q, a, p) $\in$ R, kde q $\in$ Q, p $\in$Q a a $\in\sum$, spíše volíme zápis ve tvaru qa $\rightarrow$ p $\in$ R.

Definice 19 (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.
Nechť k $\geq$ 0. M je k-left skákající konečný automat, ($_{k\text{-left}}$ GJFA zkráceně) za předpokladu, že M je GJFA a pro každé pravidlo pxy $\rightarrow$ q $\in$ R platí, že x $\in \Sigma^*$ a |x|= k.
M je k-right skákající konečný automat ($_{k\text{-right}}$ GJFA zkráceně) pokud M značí GJFA a pro každé pravidlo pyx $\rightarrow$ q $\in$ R platí, že x $\in \Sigma^*$ a |x|= k.

Definice 20 (zásobníkový automat)

Zásobníkový automat, značeno ZA, je sedmice: M= (Q, Σ, Γ, R, s, S, F).
Q značí konečnou množinu stavů, Σ je vstupní abeceda, Γ je zásobníková abeceda, R představuje konečnou množinu pravidel ve tvaru Apa $\rightarrow$ wq, kde A $\in$ Γ, p, q $\in$ Q, a $\in$ Σ $\cup\{\varepsilon\}$, w $\in \Gamma^*$, s $\in$ Q představuje počáteční stav, S $\in$ Γ je počáteční symbol na zásobníku a F $\subseteq$ Q značí množinu koncových stavů zásobníkového automatu.

Definice 21 (Konfigurace)

Konfigurace M, značena KAM je libovolný řetězec $\Sigma^*Q\Sigma^*$.
X značí množinu všech konfigurací M.

Definice 22 (levý a pravý skok)

Binární skoková relace$\sum$, symbolicky zapsáno |-, nad X je definována následovně:
Nechť x, y, u $\in \Sigma^*$, a $\in \Sigma^*$ a qa $\rightarrow$ p $\in$ R; pak M provede tzv. levý skok z konfigurace xyqau do xpyu, symbolicky zapsáno jako xyqau $_l$|- xpyu.
M provede tzv. pravý skok z konfigurace xqayu do xypu, zapsáno jako xqayu $_r$|- xypu.
Nechť ϖ, $\omega \in$ X; pak M provede skok z ϖ to ω, zapsáno jako ϖ |- ω, kdykoli je k dispozici ϖ $_l$|- ω nebo ϖ $_r$|- ω.

Nechť x, y, u $\in \Sigma^*$, |y| = k, a $\in \Sigma^*$ a qa $\rightarrow$ p $\in$ R; pak M provede $_k$left skok z konfigurace xyqau do xpypu, symbolicky zapsáno jako xyqau $_l$|-$_k$ xpyu.
M provede $_k$right skok z konfigurace xqayu do xypu, symbolicky zapsáno jako xqayu $_r$|-$_k$ xypu.
Standardně rozšiřujeme zápis z |- na |-^{m}, kde m $\geq$ 0; následně na základě binární relace |-^{m}, definujeme |-$^+$ a |-*. Nechť w $\in \Sigma^*$. M přijímá w, pokud platí, že sw |-* f a f $\in$ F. Analogicky můžeme přidat symboly $_l$|-, $_r$|-
Dále platí:
L(M, |-) = {vw $\in \Sigma^*$| v, w $\in \Sigma^*$, vsw |-* f with f $\in$ F}
L(M, $_l$|-) = {vw $\in \Sigma^*$| v, w $\in \Sigma^*$, vsw $_l$|-* f with f $\in$ F}
L(M, $_l$|-$_k$) = {vw $\in \Sigma^*$| v, w $\in \Sigma^*$, vsw $_l$|-$_k$ * f with f $\in$ F}
L(M, $_r$|-) = {vw $\in \Sigma^*$| v, w $\in \Sigma^*$, vsw $_r$|-* f with f $\in$ F}
L(M, $_r$|-$_k$) = {vw $\in \Sigma^*$| v, w $\in \Sigma^*$, vsw $_r$|-$_k$ * f with f $\in$ F}

Definice 23 (lineární jazyk)

Nechť G = (V, T, P, S) značí LG, pak jazyk generovaný LG G, značeno LIN, definujeme jako:
LIN = {w: w $\in T^*$, S $\Rightarrow^*$ w}.
Nechť L je jazyk. Tento jazyk označujeme lineárním jazykem, značeno LIN, pokud existuje $_{k\text{-left}}$ GJFA nebo $_{k\text{-right}}$ GJFA, který tento jazyk přijímá.

Definice 24 (přijímaný jazyk)

Nechť M= (Q, Σ, R, s, F) je jedním z typů konečných automatů. Jazyk přijímaný daným automatem M, L(M), je definován: L(M) = {w: w $\in \Sigma^*$, sw |–* f, f $\in$ F.

Definice 25 (pravidlový strom)

Pravidlový strom představuje grafické znázornění pravidel gramatiky s tím, že levá strana představuje kořen stromu a pravá strana pravidla syny tohoto stromu.

Definice 26 (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 27 (nejednoznačná gramatika)

Nechť $G = (N, T, P, S)$ je BKG. Pokud existuje řetězec $x \in L(G)$ s více jak jedním derivačním stromem, potom G je nejednoznačná. V opačném případě G je jednoznačná.

1.7 Metodika důkazů

Pro provádění důkazů tvrzení v teorii formálních jazyků využíváme 3 různé principy dokazování. Tyto principy vycházejí z výrokové logiky. Značky, které výroková logika obsahuje a výsledky jednotlivých výrokových operací se nacházejí v Tabulka 1-1 Význam značeka Tabulka 1-2. Principy důkazů jsou podrobně vysvětlovány v [7, s. 15-23].

Značka	Význam značky
s1, s2	dokazované formulace.
negace s	!s
ekvivalence	<=>
implikace	=>
logický součin	and
logický součet	or
logická pravda	1
logická lež	0

Tabulka 1-2 Logické operace

s1	s2	and	or	=>	<=>
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Tabulka 1-1 Význam značek

Důkaz kontrapozicí

Tato forma důkazu je založena na následujícím předpisu:
(s1 => s2) <=> (!s2 => !s1)
Čili správnost tvrzení s1 => s2 je ověřováno důkazem, že platí !s2 => !s1.

Důkaz kontradikcí

Forma dokazování tvrzení vede skrze důkaz této formulace:
(!s2 and s1) => 0.

Důkaz matematickou indukcí

Tato práce bude využívat právě tuto formu důkazu. Jádro důkazu spočívá v tvrzení, že formulace s(n) je pravdivá pro všechna přirozená čísla, $n \geq b$, kde b je kladné celé číslo. Důkaz má následující části:
Báze – dokážeme tvrzení pro konkrétní hodnotu, např. 1 nebo 0.
Indukční hypotéza – předpokládáme, že formulace je platná pro všechna i = b, …, j, kde j je celé číslo a zároveň platí, že $j \geq b$.
Indukční krok – v této fázi dokazujeme, že formulace s(j + 1) je pravdivá na základě indukční hypotézy stanovené v předchozím kroku.

Příklad 4

Provedení důkazu tvrzení s(n):

$1 + 3 + 5 + \ldots + 2n - 1 = n^2$

Báze:

s(1): $1 = 1^2$, tvrzení je pravdivé.

Indukční hypotéza:

Vytvoříme předpoklad, že tvrzení platí pro všechna i = 1, …, j, kde j je přirozené číslo.

Indukční krok:

Musíme tvrzení dokázat také pro formulaci s(j+1):

$1 + 3 + 5 + \ldots + (2j - 1) + (2(j+1) - 1) = (j + 1)^2$

Z indukční hypotézy je potvrzena ztučněná část formulace úspěšně:

$\mathbf{1 + 3 + 5 + \ldots + (2j - 1)} + (2(j+1) - 1) = \mathbf{j^2} + 2j + 1$

Pokud vyhodnotíme zbytek výrazu, zjistíme, že platí rovnost:

$(2(j+1) - 1) = 2j + 1$

Důkaz pomocí matematické indukce je tedy dokončen.

2 Síla skákajících konečných automatů

Jak již bylo v úvodu nastíněno, síla skákajících konečných automatů mnohonásobně překračuje sílu běžných konečných automatů. Sílu *GJFA* a *JFA* v souvislosti s různými rodinami jazyků zobrazuje Obrázek 3-1, který graficky znázorňuje sílu *GJFA* a *JFA*. Tato kapitola podá odpovědi na otázky:

- Proč nedokáží pokrýt tyto automaty celou množinu regulárních jazyků?
- Jsou silnější *JFA* nebo *GJFA*?
- Jsou *JFA* a *FIN* vzájemně se doplňující nástroje?
- Jaká je síla zavedené modifikace $_{k\text{-}left}$ *GJFA* a $_{k\text{-}right}$ *GJFA* v porovnání s ostatními typy skákajících automatů?

Lemma 1 (permutace přijímaných jazyků)

Nechť w představuje libovolný řetězec a perm(w) značí množinu všech permutací tohoto řetězce. Pro libovolný jazyk L přijímaným *GJFA* nebo *JFA*, platí:
perm(L) = {perm(w) : w ∈ L} [6]

2.1 Rozbor síly skákajících automatů

Část myšlenek této podkapitoly je převzata z vědeckého článku pojednávajícím o obecných skákajících automatech. [1] Některé otázky a úvahy ohledně síly obecných skákajících automatů, ze kterých vychází kapitola 2.3, obsahuje publikace [6].

Síla GJFA

Existuje některý jazyk, který nedokáže přijmout žádný obecný skákající automat?
Uvažme jazyk $\boldsymbol{L = \{A^P, p \text{ je prvočíslo}\}}$.
Nelze najít žádný *GJFA*, který přijímá jazyk *L*. Pokud totiž budeme tvořit pravidla, tak nám nepomůže možnost libovolně přeskakovat a vracet se při čtení pásky, jelikož nás omezuje kardinalita abecedy o velikosti pouze 1 u symbolu *A*.

Platí REG ⊆ GJFA?

Pokud jsme zmínili velké přednosti skákajících automatů oproti klasickým konečným automatům, dokáže tato modifikace automatů přijmout jakýkoliv regulární jazyk?

Příklad 5

Uvažme jazyk $\{a\}^*\{b\}^*$.
Zkusme pro tento jazyk najít automat, který tento jazyk přijímá.

GJFA $H = (\{<S>, <F>\}, \{a, b\}, R, <S>, <F>\})$, $R = \{<S>b \rightarrow <F>, <S>a \rightarrow <F>, <F>b \rightarrow <F>, <F>a \rightarrow <F>\}$.

Zdánlivě to vypadá, že navržený automat *H* přijímá daný jazyk. Automat *H* ale ve skutečnosti přijímá mnohem silnější jazyk, tedy i množinu řetězců, která nepatří do jazyka $\{a\}^*\{b\}^*$, jelikož může při zpracování řetězců libovolně skákat na obě dvě strany. Příjme i řetězce, ve kterých jsou symboly *a* a *b* proházeny, např. *abaab*. Pro upřesnění tento automat popisuje jazyk $\{a,b\}^*$. $\{a\}^*\{b\}^* \subseteq \{a,b\}^*$.

Další tvar automatu, který byl zvažován v souvislosti s tímto jazykem, je verze automatu, která požaduje jako první symbol *a* a následně symboly *b*. Takovým automatem je:

GJFA I = ({<S>, <F>}, {a, b}, R, <S>, <F>}), R = {<S>a → <Q>, <Q>a → <Q>, <Q>b → <F>, <F>b → <F>}.

Ani tato modifikace však nezabrání přijímání řetězců s proházenými symboly. Pouze zajistí, že jako první budeme číst symbol *a*, ale nezaručuje další postup automatu při čtení pásky. Skoky u *GJFA* jsou naprosto náhodné a libovolné. Platí tedy, že ***REG ⊈ GJFA*.**

Porovnání síly JFA a GJFA

K porovnání síly *GJFA* a *JFA* připomeňme definici **Lemma 1**, že jazyk *L* přijímaný *GJFA*, příp. *JFA*, musí být schopen přijmout tento jazyk i v případě libovolné permutace obsažených řetězců. Tedy musí platit, že jazyk *L* je přijímaný skákajícím automatem právě tehdy, když platí, že *L = perm(K)*, kde *K* je regulární jazyk.
Pro každý jazyk přijímaný *GJFA* tato podmínka platí. Existuje však takový jazyk, u kterého tato podmínka platí, avšak není přijímán žádným *JFA*. Příkladem takového jazyka může být jazyk $U = \{a,b\}^*\{ba\}\{a,b\}^*$. Nedokážeme najít žádný *JFA*, který dokáže přijmout jazyk *U*. Avšak dokážeme najít *GJFA*, který přijmout jazyk *U* dokáže. A sice

GJFA H = ({<S>, <F>}, {a, b}, R, <S>, <F>}), R = {<S>ba → <F>, <F>a → <F>, <F>b → <F>}.

Naopak nedokážeme najít žádný jazyk, pro který bychom našli přijímající *JFA* a neexistoval žádný *GJFA*, který tento jazyk přijímá.
Z toho vyplývá, že *GJFA* je silnější než *JFA*, ***JFA ⊆ GJFA***.
Vyvstává zde otázka. Proč nelze najít automat *JFA*, který přijímá tento jazyk? Uvažujme nyní *JFA* automat *I*:

I = ({<S>, <F>}, {a, b}, R, <S>, <F>}), R = {<S>b → <F>, <S>a → <F>, <F>b → <F>, <F>a → <F>}.

Tento *JFA* dokáže přijmout jakýkoliv řetězec jazyka U. Je to na první pohled matoucí. Ano, tento automat skutečně dokáže přijmout daný jazyk. Ve skutečnosti ale automat *I* přijímá mnohem širší množinu řetězců, konkrétně jazyk $\{a,b\}^*$. Mezi řetězce, které přijme, patří např. i řetězec *aaaaaa*, který nevyhovuje předpisu jazyka *U*.

Je FIN podmnožinou JFA?

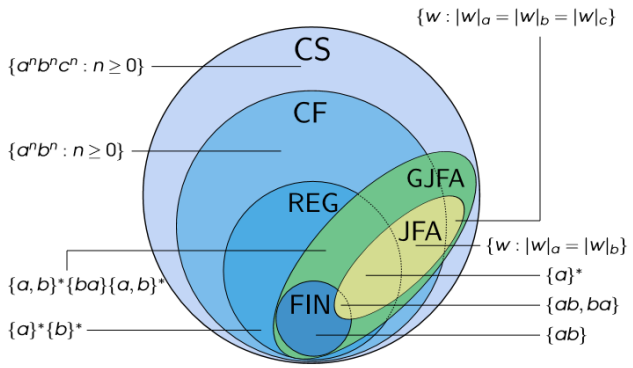
K tomu, abychom tuto myšlenku vyvrátili, zkusíme najít takový jazyk, který dokáže přijmout *FIN*, ale *JFA* nikoliv. Takovým jazykem je např. jazyk *K = {ab}*.

Je FIN podmnožinou GJFA?

Není možné najít žádný jazyk, který by *FIN* zpracoval a *GJFA* nikoliv, proto platí, že ***FIN ⊆ GJFA***.

2.2 Grafické znázornění vztahu automatů a rodin jazyků

K souhrnu otázek ohledně vztahů nově navržených automatů s rodinami jazyků využijeme obrázek z prezentace o skákajících automatech [2, s. 11].



Obrázek 3-1 Vztah rodin jazyků a jednotlivých typů automatů

2.3 Kam až sahá síla nově navržených automatů

V této podkapitole porovnáme modifikace skákajících automatů $_{k\text{-}left}$ *GJFA* a $_{k\text{-}right}$ *GJFA* s dalšími verzemi konečných automatů a následně zvážíme jejich sílu v rámci jednotlivých rodin formálních jazyků. Více podrobností o rodinách jazyků poskytne [8].
Poznámka: V případě uvažování jednotlivých množin jazyků a síly automatů vč. ukázek budeme pracovat s verzí $_{k\text{-}left}$ *GJFA*. U $_{k\text{-}right}$ *GJFA* jsou závěry analogické.

Porovnání síly s JFA:

$_{k\text{-}left}$ *GJFA* jsou omezením *GJFA*. Oproti *JFA* mají tu výhodu, že nemusí číst ze vstupní pásky v jednom kroku pouze 1 symbol. *JFA* má naopak výhodu, že může skákat libovolným směrem a pak také ještě jednu na první pohled ne zcela zřejmou výhodu. Obecné verze automatů *GJFA* a *JFA* mají skok zcela dobrovolný, tj. v každém kroku při čtení vstupu se mohou rozhodnout, zda budou skákat nebo ne. Oproti tomu $_{k\text{-}left}$ *GJFA* skáčou vždy v daném směru o *k*. Zkusíme tedy nyní porovnat jejich sílu. Uvažme ještě jednou jazyk $U = \{a,b\}^*\{ba\}\{a,b\}^*$, pro který neexistuje *JFA* automat. Najdeme k-left automat takový, který tento jazyk přijmout dokáže?

Příklad 6

Mějme $_{k\text{-}left}$ *GJFA* $H = (\{<S>, <F>\}, \{a, b\}, R, <S>, <F>\})$, $R = \{<S>ba \rightarrow <F>, <F>aa \rightarrow <F>, <F>bb \rightarrow <F>, <F>ba \rightarrow <F>, <F>ab \rightarrow <F>\}$, $k = 1$, $w = aaaababbbb$, $w \in U$, $k = 1$

Tento automat dokáže přijmout pouze určitou množinu řetězců tohoto jazyka. Přesněji řečeno právě tuto podmnožinu: $V = \{a,b\}^l\{ba\}\{a,b\}^k$, kde $l \geq k$. Proč nedokáže přijmout jazyk *U*? Jelikož v jazyce *U* se vyskytují i řetězce, kdy má levá strana od části řetězce *ba* rozdílnou délku oproti pravé straně. Rozeberme si to na příkladu:

1. Jazyk $V = \{a,b\}^k\{ba\}\{a,b\}^k$, kde $k > 0$. Pokud obě strany řetězce od podřetězce *ba* mají stejnou délku, automatem H je dokážeme přijmout.
2. Jazyk $V = \{a,b\}^l\{ba\}\{a,b\}^k$, kde $l > k$. Dokáže tento automat přijmout i takovéto řetězce? Např. *aaaaaaaaaababbb*? Ano, dokáže.
3. Velké omezení nastane v opačném případě, pokud budeme mít jazyk $V = \{a,b\}^k\{ba\}\{a,b\}^l$, kam patří např. řetězec. *ababbbbbbbbb*. Startovací stav musíme umístit před *ba*.

Práce automatu krok po kroku:

*a***<S>***babbbbbbbbb* |- **<F>***abbbbbbbbb*. V tomto stavu bohužel končíme. Sice máme k dispozici v množině *R* pravidlo *<F>ab → <F>*, ale nemůžeme uskutečnit další skok vlevo. Skoky u těchto automatů jsou povinné. Automat *H* tedy nedokáže přijmout tento jazyk.

Pokud se vrátíme zpět k porovnávání síly *JFA* a $_{k\text{-}left}$ *GJFA*, ani pro podmnožinu jazyka *U*, jazyk *V*, však neexistuje žádný *JFA*, který tento jazyk přijímá. Právě jsme dokázali, že $\boldsymbol{JFA \not\subseteq {}_{k\text{-}left} GJFA}$.

Platí opačné tvrzení? Je pravda, že $\boldsymbol{JFA \subseteq {}_{k\text{-}left} GJFA}$? Nikoliv. Abychom tvrzení doložili, najděme jazyk, který dokáže přijmout *JFA* a zároveň pro něj nenajdeme $_{k\text{-}left}$ *GJFA*, který tento jazyk dokáže přijmout.

Nalezení takového jazyka je snadné, stačí pouze zmodifikovat jazyk *U*.

Uvažme jazyk $C = \{a\}^*\{b\}\{a\}^*$. *JFA*, který tento jazyk dokáže přijmout, je např. tento:

I = ({<S>, <F>}, {a, b}, R, <S>, <F>}), R = {<S>b → <F>, <F>a → <F>}.

Naproti tomu ale nedokážeme najít žádný $_{k\text{-}left}$ *GJFA* pro jazyk *C*.

Z toho vyplývají 2 závěry: $\boldsymbol{JFA \not\subset {}_{k\text{-}left} GJFA}$ a $\boldsymbol{{}_{k\text{-}left} GJFA \not\subset JFA}$. A mj. jsme při zkoumání jazyků *U* a *C* ve vztahu $_{k\text{-}left}$ ***GJFA*** prokázali tvrzení, že $\boldsymbol{REG \not\subset {}_{k\text{-}left} GJFA}$.

2.3.1 Porovnání síly s GJFA

Logický předpoklad je tvrzení $_{k\text{-}left}$ *GJFA* ⊆ *GJFA*. Zde ale dojdeme k překvapivému závěru, neboť toto tvrzení vyvrátíme. Z Obrázek 1-1 víme, že pro jazyk $\{a^nb^n, n > 0\}$ nedokážeme najít *GJFA*, který jej přijímá. Důvodem je, že navržený automat přijme i řetězce, které mají proházené symboly *a* a *b*. Nastavená omezení $_{k\text{-}left}$ *GJFA* automatu však tento neduh eliminuje. Dokážeme tedy najít $_{k\text{-}left}$ *GJFA* automat pro tento jazyk, a sice:

$_{1\text{-}left}$ *GJFA J = ({<S>, <F>}, {a, b}, R, <S>, <F>}), R = {<S>ab → <F>, <F>ab → <F>}.*

Pokud by automat *J* byl místo $_{k\text{-}left}$ *GJFA* obecný *GJFA*, přijímal by i chybné sekvence podřetězců, např. *abaaaabbb*. Tímto jsme tedy dokázali překvapivý fakt, že $\boldsymbol{{}_{k\text{-}left} GJFA \not\subseteq GJFA}$.

Porovnání síly s FIN

Vztah *FIN* se skákajícími automaty je velmi zajímavý. Doposud víme, že *FIN* ⊆ *GJFA* a naopak *FIN* ⊄ *JFA*. Platí tvrzení, že *FIN* ⊆ $_{k\text{-}left}$ *GJFA*? Pro $X = \{a\}^*\{ab\}\{b\}^*$ dokážeme najít *FIN*, který tento jazyk přijímá, např.

M = (<S>, <F>}, {a, b}, {<S>a → <Q>, <Q>a → <Q>, <Q>ab → <F>, <F>b → <F>}, <S>, <F>)

Zde je podobná situace jako v případě zkoumání jazyka U ve vztahu k $_{k\text{-}left}$ *GJFA* v předešlé sekci. Jazyk *X* nedokážeme zpracovat pomocí $_{k\text{-}left}$ *GJFA*, protože jsme omezeni směrem skoku vlevo a nutností tohoto skoku. K-left verze automatu nedokáže přijmout jakýkoliv řetězec patřící do *X*, kde je levý

podřetězec od *ba* kratší než pravý podřetězec, tj. $\{a,b\}^k\{ab\}\{a,b\}^l$, kde $l > k$. Platí tedy, že ***FIN*** $\not\subset$ ***k-left GJFA*** a zároveň ***k-left GJFA*** $\not\subset$ ***FIN***.

2.4 Souhrn poznatků

Vysvětlivky k tabulce

V levém sloupci tabulky máme jednotlivé typy konečných automatů, jejichž síla je porovnávána s ostatními typy automatů umístěných v záhlaví tabulky. Postupně procházíme jednotlivé řádky tabulky.

Symbol + znamená, že automat z levého sloupce je silnější než porovnávaný automat. Např. v prvním řádku vidíme + u *JFA*. Neexistuje žádný jazyk, který by *JFA* dokázal přijmout a zároveň *GJFA* ne.

Symbol – znamená, že automat je slabší než porovnávaný automat. Tj. opačná situace oproti předchozímu případu.

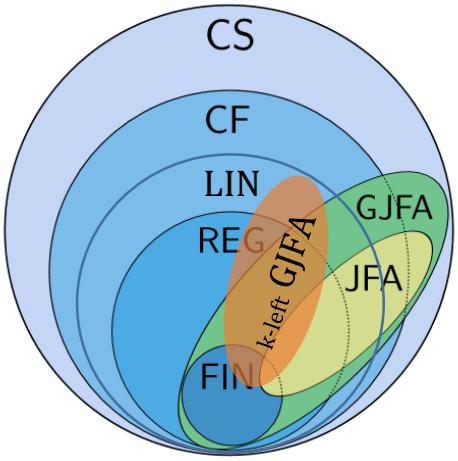
Poslední možností je místo znaků + – uvedený v buňce konkrétní jazyk. Jedná se o příklad jazyka, který zapříčiňuje, že automat není silnější než porovnávaný automat. Např. automat *k-left GJFA* není silnější než *JFA*, jelikož existuje např. jazyk $C = \{a\}^*\{b\}\{a\}^*$, který *JFA* dokáže přijmout, ale naopak *k-left GJFA* nikoliv. Zároveň není ani slabší, protože i v opačném směru najdeme příklady jazyků. Platí tedy mezi těmito jazyky ***JFA*** $\not\subset$ ***k-left GJFA*** a zároveň ***k-left GJFA*** $\not\subset$ ***JFA.***

Typ automatu	GJFA	JFA	k-left GJFA	FIN
GJFA		+	$\{a^n b^n, n > 0\}$	+
JFA	-		$\{a,b\}^l\{ba\}\{a,b\}^k$, kde $l > k$	$\{ab\}$
k-left GJFA	$\{a,b\}^*\{ba\}\{a,b\}^*$	$\{a\}^*\{b\}\{a\}^*$		$\{a\}^*\{ab\}\{a\}^*$
FIN	-	$\{w : \lvert w\rvert a = \lvert w\rvert b = \lvert w\rvert c\}$	$\{a,b\}^l\{ba\}\{a,b\}^k$, kde $l > k$	

Tabulka 3-1 Souhrn síly konečných automatů

2.5 Grafické znázornění $_{\text{k-left}}$ GJFA v množinách jazyků

Automaty *k-GJFA* jsou ekvivalentním modelem k-left a k-right lineárních gramatik. Důkazy tohoto tvrzení jsou popsány v 6. kapitole. Lineární gramatiky generují lineární jazyky, které jsou podmnožinou bezkontextových jazyků.



Obrázek 3-2 Grafické znázornění pozice k-left a k-right GJFA v kontextu formálních jazyků

3 Zpracování derivačních stromů aneb užití k-left a k-right automatů při syntaktické analýze

V předchozí kapitole jsme dospěli k závěru, že k-left a k-right skákající automaty dokáží zpracovávat některé bezkontextové jazyky, dokonce i takové, které obecné verze skákajících automatů nedokáží přijmout, např. $\{a^n b^n: n \geq 0\}$. V této části textu budou zváženy možnosti, kterými by bylo možné zpracování řetězců pocházejících z derivačních stromů pomocí tohoto formálního prostředku umožnit.

3.1 Kde je zádrhel

Příklad 7

*Nechť CFG G= (N, T, P, E), kde N= {E, F, T}, T= {i, +, *, (,)}, P= {E → E+T, E → T, T → T*F, T → F, F → (E), F → i}* [3, s. 11 kapitola 6]

K-left a k-right skákající automaty jsou ekvivalentním modelem ke k-left a k-right lineárním gramatikám. Lineární gramatiky představují podmnožinu bezkontextových gramatik omezených tím, že pro každé x → y ∈ P platí, že $y \in T^k N T^* \cup T^*$.

Nyní tedy budeme uvažovat, jaké jsou možné důvody, proč nelze použít k-left a k-right automaty k syntaktické analýze. Uvažme nyní důvod, proč jazyk generovaný *CFG G* nedokáže přijmout žádný konečný automat. Odpovědí jsou páry závorek generovaných pravidlem *F → (E)*, se kterými si konečné automaty nedokáží poradit. Závorky ale na první pohled nejsou zábranou pro přijmutí řetězce $_{k\text{-}left}$ *GJFA* automatem.

Dokážeme najít $_{k\text{-}left}$ *GJFA* např. pro jazyk generovaný gramatikou popsanou níže, která obsahuje páry kulatých závorek:

CFG Y = (N, T, P, E), kde N= {E, F, T}, T= {i, +, (,)}, P= {S → (E), E → iFi, F → +T, T → iF, T → ε}.

Gramatika generuje jazyk *L = { ´(´ i*n´)´, n > 0}*

Jak je zřejmé, ani povinný skok v jednom směru nepředstavuje tak razantní omezení, neboť zásobníkové automaty využívají podobnou metodu při zpracovávání řetězců, tzv. shiftování, a mj. také čtou vstupní pásku pouze v jednom směru. Co je ale velkou překážkou $_{k\text{-}left}$ *GJFA* využití při syntaktické analýze je fakt, že nedokáží zpracovat žádnou část řetězce, která byla vygenerována pravidlem s dvěma neterminály na pravé straně, např. *E → E+T*.

Podobné konstrukce pravidel však jsou však hlavní náplní derivačních stromů, jakožto hlavního prostředku syntaktické analýzy. Závěrem je, že skákající automaty není možné využít pro zpracování složitějších matematických nebo programových konstrukcí. Abychom mohli skákající automaty využít při syntaktické analýze, museli bychom uvážit takovou modifikaci skákajícího automatu, která by umožňovala vytvořit ekvivalentní pravidlo automatu pro veškerá pravidla gramatiky, která mají na pravé straně 2 neterminály.

3.2 Existuje nějaká možnost?

Najdeme v teorii formálních jazyků nějakou variantu, při které by pomocí *k-GJFA* bylo možno zpracovat složitější výraz nebo jednoduché programové konstrukce? Dokážeme najít řešení, které vytvoří ekvivalentní pravidlo *k-GJFA* k pravidlu gramatiky s více neterminály na levé straně? Tyto otázky jsou klíčové pro vyřešení úvahy, zda je možno tento formální prostředek využít při syntaktické analýze.

Jedno z možných řešení, kterými bychom dokázali takováto pravidla v některých situacích přijmout, je duplikování *k-GJFA* u takových pravidel derivačního stromu, která mají více než jeden neterminál na pravé straně. Takto spojíme potenciál $_{k\text{-}right}$ *GJFA* i $_{k\text{-}left}$ *GJFA* dohromady.

Příklad 8

Předvedeme si demonstraci návrhu a činnosti popisovaného řešení.
Mějme *CFG* $G = (N, T, P, U)$, *kde* $N = \{U, A, B\}$, $T= \{i, +, *\}$, $P= \{ U \rightarrow A+B, A \rightarrow i*A, B \rightarrow i+B, B \rightarrow i, A \rightarrow i\}$.
K bezkontextové gramatice vytvoříme ekvivalentní sestavu automatů *k-GJFA* $M_i,..,M_n$ pro $n \geq 1$ a M_i je *k-GJFA* pro všechna $i = 1, \dots , n$. Jak vytvoříme sestavu automatů?

Algoritmus III. Tvorba sestavy k-GJFA

Metoda:

$Q := N \cup \{S\};$
$\sum := T;$

Konstrukce množin R:

Algoritmus je velmi podobný Algoritmu IV., pravidla mají modifikovaný tvar. V místě, kde se nachází 2 a více neterminálů na pravé straně, budeme automat duplikovat:
For each $A \rightarrow A\ c\ B \in P$ **and** $A \rightarrow x$ **and** $B \rightarrow x$**; where** $x \in T^*$, $A \in N\ U\ \{S\}$, $B \in N$, $c \in T^*$, $|x+c| = k$:

Duplikuj *k-GJFA* automat na $_{k\text{-}right}$ *GFFA* $M_{A,}$ $_{k\text{-}left}$ *GJFA* M_B**:**
Takovým pravidlem je v našem případě $U \rightarrow A+B$, budeme tedy duplikovat automat a vzniknou 2 dílčí verze $_{2\text{-}left}$ *GJFA* M_B a $_{2\text{-}right}$ *GJFA* M_A.
V místě, kde došlo k duplikaci automatu, vytvoříme ke každému z nově vytvořených dílčích automatů epsilon pravidlo pro startující stav daného automatu ($S\varepsilon \rightarrow S$).
add $S\varepsilon \rightarrow S$**; where** $<S> \in Q$ **to every** *k-GJFA***.**
For $_{k\text{-}right}$ *GJFA* M_A:
For $A \rightarrow x$ **add** $<A>cx \rightarrow <U>$.
For $_{k\text{-}left}$ *GJFA* M_B:
For $B \rightarrow x$ **add** $<B>cx \rightarrow <U>$.

V našem případě tak vzniknou pravidla $<B>+i \rightarrow <U>$ pro M_B a $<A>+i \rightarrow <U>$ pro M_A.
Dále pro každý automat vytvoříme pravidla tímto způsobem:
For every $_{k\text{-}right}$ *GJFA:*

For each $A \rightarrow x\ B\ \varepsilon \in P$**; where** $x, y \in T^*$, $A \in N \cup \{S\}$, $B \in N$, $|x| = k$:
add $<B>$ **reversal**$(x) \rightarrow <A>$, **where** $<A> \in Q$, $<B> \in Q$.

For every $_{k\text{-}left}$ *GJFA:*

For each $A \rightarrow x\,B\,\varepsilon \in P$**:**
ADD $<B>$ **reversal**$(x) \rightarrow <A>$.

Důležitým rozdílem oproti Algoritmu I. je, že pravidlo automatu má provedenou reverzaci u řetězce terminálů.
Přidáme $<B>+i \rightarrow <B>$ do $_{2\text{-}left}$ *GJFA* M_B a $<A>*i \rightarrow <A>$ do $_{2\text{-}right}$ *GJFA* M_A.

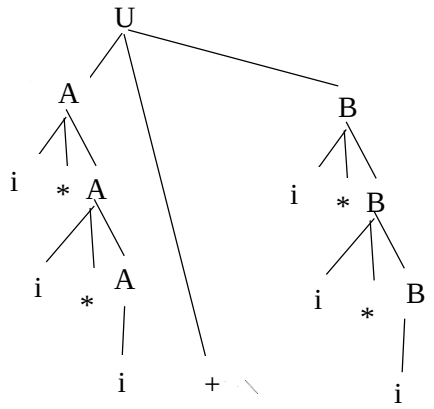
3.3 Výsledné automaty a ukázka jejich práce

$_{2\text{-}right}$ *GJFA* $M_A = (<S>, <A>, <U>\}, \{*, i, +\}, \{<A>*i \rightarrow <A>, \mathbf{<S>\varepsilon \rightarrow <S>}, \mathbf{<A>+i \rightarrow <U>}\}, <S>, <U>)$,

$_{2\text{-}left}$ *GJFA* $M_B = (<S>, <B>, <U>\}, \{+, i\}, \{<B>+i \rightarrow <B>, \mathbf{<B>+i \rightarrow <U>}, \mathbf{<S>\varepsilon \rightarrow <S>}\}, <S>, <U>)$.

Příklad 9

CFG vygeneruje derivační strom:



Vygenerovaný řetěz:
i * i * i + i + i + i
Práce skákajících automatů M_A a M_B: startující stav prvního z automatů postavíme vždy na začátek vstupu a startující stav druhého z duplikovaných automatů na konec vstupu. Automaty při čtení kráčí proti sobě:

M_A	M_B
<S> i * i * i + i + i + i	i * i * i + i + i + i <S>
i * <S> i * i + i + i + i	i * i * i + i + i <S> + i
i * <A> i + i + i + i	i * i * i + i <S> + i
i + <A> i + i + i	i * i * i <S> + i
<U> i + i + i	i * i * i <U>

Omezení tohoto algoritmu:

Sestava skákajících automatů je jedna z možností, jak dosáhnout zpracování základních matematických výrazů nebo programových konstrukcí, jejichž derivační stromy obsahují pravidla s dvěma neterminály na pravé straně. Mezi velké výhody tohoto formálního prostředku patří zejména rychlost zpracování řetězců. Vyšší rychlost je způsobena současným zpracováváním vstupu na více místech zároveň díky rozdělení jednoho formálního prostředku na více jeho duplikátů.

Mezi hlavní nevýhody tohoto prostředku patří fakt, že ho lze využít pouze ve dvou typech případů. V případě, že gramatika obsahuje pouze pravidla končící na pravé straně neterminálem, např. $A \rightarrow xB$ nebo v případě, že pravidla neobsahují žádný netereminál na pravé straně, např. $A \rightarrow x$. Pokud by gramatika obsahovala pravidlo ve tvaru $A \rightarrow xBy$, tento princip bychom nemohli aplikovat. V případě naší zkoumané verze skákajících automatů také musí platit pro pravidla $A \rightarrow x$ a $A \rightarrow AcB$, že $|x+c| = k$.

Komplikace se může vyskytnout při implementaci tohoto formálního prostředku v případě, že tvoříme automat ekvivalentní ke gramatice, která je nejednoznačná. Z nejednoznačné gramatiky vznikne sestava nedeterministických skákajících automatů. U jednotlivých duplikací automatů bude obtížné určit, kde má jejich zpracovávání končit a která část již patří dalšímu z automatů. Další možné obtíže při implementaci týkající se obecně *k-GJFA* jsou popsány v následující kapitole.

Pokud bychom měli shrnout výsledky úvah a algoritmů uvedených v této kapitole, pak můžeme prohlásit, že sestava *k-GJFA* dokáže nahradit zásobníkové automaty u základních výrazů a konstrukcí, přesněji u takových konstrukcí, u kterých se nevyskytují žádné z obtíží popsaných výše. V takovém případě dokáže sestava *k-GJFA* značně urychlit zpracování řetězců.

5 Možné zádrhely a problémy při užití k-GJFA v praxi

5.1 Pozice pro startující stav

Pokud začínáme zpracovávat vstup pomocí některého automatu ze skupiny skákajících, přichází jako první otázka na řadu – Odkud začít číst vstup? Nebo jinými slovy, do jakého místa vstupní pásky postavit startující stav.

Příklad:

GJFA M = ({<S>, <A>, <B>, <C>}, {u, v, w}, {<B>uvw → <A>, <C>vw → <B>, <S>uuuu → <C>}, <S>, <A>), na vstupu máme řetězec *uvuuuuwvw*.

Kam se postavíme, pokud chceme začít číst vstupní pásku automatem *M*? Při užití klasických konečných automatů nebo zásobníkových automatů jsme vždy startující stav postavili na začátek vstupní pásky zleva. Pokud bychom to udělali i v případě tohoto automatu, vypadala by aktuální konfigurace takto:
<S>uvuuuuwvw
Nedokázali bychom v danou chvíli uplatnit žádné z pravidel. Ale z definice *GJFA* víme, že pokud se dostane automat do jednoho ze stavů, ke kterému nemá pro aktuální vstupní symboly na pásce k dispozici žádné pravidlo, může kterýmkoliv směrem přeskočit a hledat na vstupní pásce místo takové, na které dokáže uplatnit některé z pravidel. Startující stav tím pádem může přesunout na kterékoliv místo. Pokud se podíváme na vstup, je zřejmé, že výslednou konfigurací vhodnou pro další úspěšné pokračování zpracovávání vstupu je:
uv <S>uuuuwvw.
U obecných skákajících automatů (*GJFA*) je problematika velikosti, množství a směru skoků, ale také hledání pozice startujícího stavu na vstupní pásce zcela náhodná.

Jiná situace je u aplikovaného omezení *k-GJFA*. Zde už je problematika skoků naprosto jednoznačná. Samotný skok má pevně danou velikost na základě parametru *k* a také pevně daný směr skoku na základě upřesnění left vs. right. *GJFA* mají možnost skoku, tj. v každé konfiguraci automat sám rozhodne, zda je potřeba skok provést nebo ne. Oproti tomu u modelu *k-GJFA* je skok povinný při každém kroku tohoto automatu. Jak vyřešit nalezení startujícího stavu implementačně?

Uvažme, že automat z našeho příkladu je nyní *k-left* a *|k|= 1*. Při zpracovávání pásky má povinnost při každém kroku skákat o velikost parametru *k* vlevo. Z toho vyplývá, že pozice startujícího stavu tohoto automatu je vždy *k*počet kroků* při pozici zleva.
Tzn.

u v <S> u u u u w v w |- u <...> v <C> w v w |- u <C> v w v w |- <...> u <B> v w |- <B> u v w |- <A>.

Vidíme, že jsme v tomto případě dvakrát skákali o *k*.

Definice 28 (Počáteční pozice startujícího stavu k-GJFA)

Počáteční pozice startujícího stavu, P<S>, je definována jako: P<S> = k * počet_skoků.

Hodnota *k* je v našem případě 1 a automat provedl 2 skoky, čili pozice *P<S> = 1 * 2 = 2*.
Startující stav posuneme o 2 symboly vpravo od začátku vstupní pásky.
Implementačně můžeme najít startující stav $_{k\text{-}left}$ *GJFA* pomocí algoritmu:

Algoritmus IV. Hledání místa startujícího stavu v k-GJFA

1. Postav startující stav na začátek pásky zleva.
2. Zkoušej postupně všechna pravidla z R a hledej proveditelné pravidlo.
3. Pokud bylo nalezeno proveditelné pravidlo – pozice startujícího stavu je aktuální pozice, v opačném případě posuň pozici startujícího stavu <S> o velikost parametru *k* a opakuj body 2 a 3.

Pro k-right verzi je princip stejný s rozdílem směru. Celý tento algoritmus je uplatňován v přiložené aplikaci.

5.2 Nedeterminismus při hledání počátečního stavu

V kapitole 5.1 je představen algoritmus, který lze využít pro vyhledání pozice počátečního stavu. Tento algoritmus však skrývá určitý nedeterminismus. V teorii formálních jazyků to nepředstavuje až takovou potíž, jako z hlediska implementace. Kam se postavíme s počátečním stavem, pokud se nachází podřetězec přijímaný na základě pravidla skákajícího automatu na více místech?

Příklad 10

Kam se postavíme startujícím stavem *<S>*, pokud máme na vstupu řetězec *ababab* a v množině pravidel *R* daného skákajícího automatu se nachází pravidla *<S>ab → <F>, <F>ab → <F>*?
Pokud bychom pracovali podle zavedeného Algoritmus IV. dostaneme tuto konfiguraci: *<S>ababab*. V tom případě ale nedokážeme zpracovat vstupní řetězec, neboť nemůžeme provést následný povinný skok vlevo.
Vyřeší problém v takovém případě změna, při které se postavím startujícím stavem před nejpravější podřetězec, který můžeme z daného stavu přijmout?
Tedy konfigurace bude *abab<S>ab*. Nyní již vstup zpracujeme bez potíží. Můžeme prohlásit obecně, že tato změna v Algoritmus IV. eliminuje výskyty nedeterminismu? Bohužel ne. Odpověď je zdůvodněna v příkladu níže.

Příklad 11

Mějme řetězec *w = abaaabaaab*, který je vygenerovaný *LG G = (N, T, P, I)*, kde *N= {I, C, U}, T = {a, b}, P = { I → abCab, C → aaUaa, U → ab}*. Příslušný automat k této gramatice je automat $_{2\text{-}left}$ *GJFA M = ({<S>, <I>, <U>, <C>}, {a, b}, {<S>ab → <U>, <C>abab → <I>, <U>aaaa → <C>},<S>, <I>)*
Pokud postavíme startující stav před poslední podřetězec vyhovující pravidlu, dostaneme konfiguraci: *abaaabaa<S>ab*. Konfigurace po provedení jednoho kroku automatu bude vypadat takto: *abaaab<U>aa*. Dále už není možné použít žádné z pravidel automatu, tudíž výsledkem je, že automat nepřijímá řetězec *w*. **Ale tento závěr není správný.** Byl způsoben pouze volbou špatné pozice pro startující stav. Správné zpracování řetězce *w* skákajícím automatem *M* krok po kroku je následující:

abaa<S>abaaab => ab<U>aaaaab => <C>abab => <I>.

$_{k\text{-}left}$ *GJFA* nemohou libovolně přeskakovat a posouvat se v umístěních stavů oproti obecným *GJFA* a *JFA*. Algoritmus pro správné vyhodnocení počátečního stavu je klíčový a může být předmětem dalšího zkoumání společně s různými dalšími modifikacemi skákajících automatů. V ukázkové aplikaci, která je součástí práce, je využíván princip s lineární časovou složitostí, tj. postupné zkoušení pozice startujícího stavu zleva doprava.

Tato lineární metoda je však velmi neefektivní. Kromě toho je velmi nevhodná k užití v situaci, která se dá označit téměř jako patová pro zjištění pozice startujícího stavu tímto algoritmem. Jedná se o $_{k\text{-}left}$ *GJFA* obsahující epsilon pravidlo u startujícího stavu, tedy <S> *ε*. Při výskytu zmíněného pravidla nelze žádným způsobem deterministicky určit, kam startující stav umístit. Jedinou možností je tedy postupně se posouvat po vstupní pásce a zkoušet všechny možné kombinace.

Příklad 12

LG Y = (N, T, P, E), kde N= {L, E, F, T}, T= {i,+, (,)}, P= {L → (E), E → iFi, F → +T, T → i F, T → ε}.

Příslušný *k-GJFA* k dané gramatice je:

$_{1\text{-}left}$ *GJFA O = ({<S>, <L>, <E>, <F>, <T>}, {i, +, (,)}, {<S> ε → <T>, <F>i → <T>, <T>+ → <F>,<F>ii → <E>, <E>() → <L>}, <S>, <L>).*

Mějme řetězec *w = (i+i+i+i)* vygenerovaný gramatikou *Y*.

Kam ale v tomto případě umístíme počáteční stav automatu *O*? Pokud bychom chtěli naimplementovat i zpracování řetězců a jazyků pomocí automatů, které obsahují epsilon pravidla, bez dalších optimalizací by dříve popisovaná lineární metoda byla velmi časově náročná, tj. museli bychom startující stav umísťovat postupně zleva doprava před každý symbol a zkoušet, zda v dané konfiguraci lze řetězec přijmout a případně posouvat pozici o jeden krok dále. Konkrétně v našem případě bychom postupně testovali konfigurace:

<S>(i+i+i+i), (<S>i+i+i+i), (i<S>+i+i+i), (i+<S>i+i+i), (i+i<S>+i+i),...,(i+i+i+i)<S>ε.

V přiložené aplikaci je z výše uvedených důvodů prováděna pouze demonstrace *k-GJFA* skákajících automatů bez epsilon pravidel.

6 Důkazy algoritmů pro tvorbu skákajících automatů a gramatik

6.1 Konstrukce automatu

Algoritmus V. Tvorba $_{k\text{-left}}$ GJFA z $_{k\text{-left}}$ LG

Vstup: $_{k\text{-left}}$ *LG G = (N, T, P, S);*
Výstup: $_{k\text{-left}}$ *GJFA M= (Q, ∑, R, s, F)* takový, že **L(G) = L(M)f**

Metoda:

$Q := N \cup \{S\};$
$\sum := T.$

Konstrukce množiny R:

For each $A \rightarrow xBy \in P;$ **where** $x, y \in T^*, A \in N \cup \{S\}, B \in N, |x| = k$:
add $<B>xy \rightarrow <A>$, **where** $<A> \in Q, <B> \in Q.$
For each $A \rightarrow x \in P; |x| = k:$
add $<S>x \rightarrow <A>;$ **where** $<S> \in Q.$

Lemma I.

Algoritmus V. je korektní. Tudíž, pro každý $_{k\text{-left}}$ *LG*, existuje ekvivalentní $_{k\text{-left}}$ *GJFA*.

Důkaz

K prokázání této lemmy provedeme důkaz Claimu A a Claimu B.

Claim A

If $A =>^n x^nBy$ **in** G, **where** $n \geq 1$, G je $_{k\text{-left}}$ LG, $k >= 1, A \in N, B \in N, x \in T^k, |x| = k, y \in T^*.$
then $x^{n-1}<B>xy \vdash^n <A>$ **in** M, **where** $<A> \in Q, <B> \in Q$, M je $_{k\text{-left}}$ $GJFA$.

Báze

Nechť $n = 1$:
If $A =>^1 x^1By$ in G **then** $<B>xy \vdash^1 <A>$ in M.
Na základě Algoritmus V. (první věta konstrukce množiny R) toto **tvrzení platí**.

Indukční hypotéza

Předpokládejme, že máme přirozené číslo $c \geq 1$ a toto tvrzení platí pro všechna čísla $1 \leq n \leq c$**.**

Indukční krok

Dokažme, že platí tvrzení i pro $c + 1$.
Nechť $n = c + 1:$

If $A =>^{c+1} x^{c+1}By$ **in** G **then** $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M, **where** $A \in N$, $B \in N$, $x \in T^k$, $|x| = k$, $<A> \in Q$, $<B> \in Q$, M je $_{k\text{-}left}$ $GJFA$, G je LG.
Vyjádříme $A => x^{c+1}By$ **in** G jako:
$A =>^c x^c B´z =>^1 x^c x^1 Byz$, **where** $B´ \in N$, $z \in T^*$.
Vyjádříme $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M jako:
$x^c x^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z \vdash^c <A>$, **where** $<B´> \in Q$.

Shrnutí a závěr důkazu

Tvrzení **If** $A =>^c x^c B´z$ **in** G **then** $x^{c-1}<B'>x\,z \vdash^c <A>$ **in** M platí na základě indukční hypotézy.
Tvrzení **If** $x^c B´z => x^c x^1 Byz$ **in** G **then** $x^c<B>xyz \vdash^1 x^{c-1}<B'>xz$ **in M.** plyne z definice konstrukce pravidel množiny R.
Na základě principu indukce bylo **tvrzení dokázáno**.

Claim B

If $x^{n-1}<B>xy \vdash^n <A>$ **in** M**, where** $n \geq 1$, $<A> \in Q$, $<B> \in Q$, $x \in T^k$, $|x| = k$, $y \in T^*$, M je $_{k\text{-}left}$ $GJFA$ **then** $A =>^n x^n By$ **in** G, **where** G je $_{k\text{-}left}$ LG, $k >= 1$, $A \in N$, $B \in N$.

Báze

Nechť $n = 1$:
If $<B>x^1y \vdash^1 <A>$ **in** M **then** $A =>^1 x^1By$ **in** G.
Na základě Algoritmu V. (první věta konstrukce množiny R) toto **tvrzení platí**.

Indukční hypotéza

Předpokládejme, že máme přirozené číslo $c \geq 1$ a toto tvrzení platí pro všechna čísla $1 \leq n \leq c$**.**

Indukční krok

Dokažme, že platí tvrzení i pro $c + 1$.
Nechť $n = c + 1$:
If $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M **then** $A => x^{c+1}By$ **in** G, **where** $A \in N$, $B \in N$, $x \in T^k$, $|x| = k$, $<A> \in Q$, $<B> \in Q$, M je $_{k\text{-}left}$ $GJFA$, G je LG.
Vyjádříme $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M jako:
$x^c x^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z \vdash^c <A>$, **where** $<B´> \in Q$.
Vyjádříme $A => x^{c+1}By$ **in** G jako:
$A =>^c x^c B´z =>^1 x^c x^1 Byz$, **where** $B´ \in N$, $z \in T^*$.

Shrnutí a závěr důkazu

Připomeňme, že pravidlo ve tvaru $<B>xy \rightarrow <A> \in R$ vzniká na základě algoritmu V. z gramatickho pravidla $A \rightarrow xBy \in P$. Tím pádem je platná konstrukce **if** $x^c x^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z$ **in M then** $x^c B´z =>^1 x^c x^1 Byz$ **in G**.
Na základě indukční hypotézy platí věta **if** $x^{c-1}<B>xy \vdash^c <A>$ **then** $A =>^c x^c B´z$ in **G.**
Principem indukce bylo prokázáno, že **tvrzení platí**.

Příklad

Nechť $k = 1$. Uvažujme $_{k\text{-}left}$ *LG* $G = (\{A, B, C\}, \{u, v, w\}, P, S)$, kde *P* obsahuje množinu pravidel: $A \rightarrow uBvw$, $B \rightarrow vCw$, $C \rightarrow uuuu$. Startující neterminál je *A*.
K převodu $_{k\text{-}left}$ *LG* na ekvivalentní $_{k\text{-}left}$ *GJFA* $M = (\{<S>, <A>, <B>, <C>\}, \{u, v, w\}, R, <S>, <A>)$, budeme postupovat podle algoritmu konstrukce množiny R (Algoritmus V.):
$A \rightarrow uBvw$ **in** *P* má ekvivalent $<B>uvw \rightarrow <A>$ **in** *R*.
$B \rightarrow vCw$ **in** *P* má ekvivalent $<C>vw \rightarrow <B>$ **in** *R*.
$C \rightarrow uuuu$ **in** *P* má ekvivalent $<S>uuuu \rightarrow <C>$ **in** *R*.
Derivace krok po kroku:
$A => uBvw => uvCwvw => uvuuuuwvw$.
Práce automatu krok po kroku:
$uv<S>uuuuwvw \vdash u<C>v\,wvw \vdash <B>uvw \vdash <A>$.

6.2 Konstrukce gramatiky

Algoritmus VI. Tvorba $_{k\text{-}left}$ LG z $_{k\text{-}left}$ GJFA

Vstup: $_{k\text{-}left}$ *GJFA* $M = (Q, \sum, R, s, F)$
Výstup: $_{k\text{-}left}$ *LG* $G = (N, T, P, S)$ taková, že **L(M)f = L(G)**

Metoda:

$N := Q;$
$T := \sum.$

Konstrukce množiny P:

For each $<B>xy \rightarrow <A> \in R;$ **where** $<A> \in Q$, $<B> \in Q$, x a $y \in \sum^*$, $|x| = k$:
add $A \rightarrow xBy;$ **where** $A \in N \cup \{S\}$, $B \in N$.
For each $<S>x \rightarrow <A> \in R$; **where** $<S> \in Q$, $|x| = k$:
add $A \rightarrow x$.

Lemma II.

Algoritmus V. je korektní. Tudíž, pro každý $_{k\text{-}left}$ *GJFA*, existuje ekvivalentní $_{k\text{-}left}$ *LG*.

Důkaz:

K prokázání lemmy 2 provedeme důkaz Claimu A a B.

Claim A

If $x^{n-1}<B>xy \vdash^n <A>$ **in** *M*, **where** $n \geq 1$, $<A> \in Q$, $<B> \in Q$, $x \in T^k$, $|x| = k$, $y \in T^*$, *M* je $_{k\text{-}left}$ *GJFA*
then $A =>^n x^nBy$ **in** *G*, **where** *G* je $_{k\text{-}left}$ *LG*, $k >= 1$, $A \in N$, $B \in N$.

Báze

Nechť $n = 1$:
If $<B>x^1y \vdash^1 <A>$ **in** M **then** $A \Rightarrow^1 x^1By$ **in** G.
Na základě Algoritmus VI. (první věta konstrukce množiny P) **tvrzení platí.**

Indukční hypotéza

Předpokládejme, že máme přirozené číslo $c \geq 1$ a toto tvrzení platí pro všechna čísla $1 \leq n \leq c$**.**

Indukční krok

Dokažme, že platí tvrzení i pro $c + 1$.
Nechť $n = c + 1$:
If $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M **then** $A \Rightarrow x^{c+1}By$ **in** G, **where** $A \in N$, $B \in N$, $x \in T^k$, $|x| = k$, $<A> \in Q$, $<B> \in Q$, M je $_{k\text{-}left}$ $GJFA$, G je LG.
Vyjádříme $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M jako:
$x^cx^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z \vdash^c <A>$, **where** $<B'> \in Q$.
Vyjádříme $A \Rightarrow x^{c+1}By$ **in** G jako:
$A \Rightarrow^c x^c B'z \Rightarrow^1 x^cx^1Byz$, **where** $B' \in N$, $z \in T^*$.

Shrnutí a závěr důkazu

Na základě Algoritmus VI. (první věta konstrukce množiny P) platí, že $A \rightarrow xBy \in P$ je ekvivalentní pravidlu automatu $<B>xy \rightarrow <A> \in R$. Tím pádem je platná konstrukce **if** $x^cx^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z$ **in M then** $x^c B'z \Rightarrow^1 x^cx^1Byz$ **in G**.
Na základě indukční hypotézy platí věta **if** $x^{c-1}<B>xy \vdash^c <A>$ **then** $A \Rightarrow^c x^c B'z$ in **G.**
Principem indukce bylo prokázáno, že **claim A platí.**

Claim B

If $A \Rightarrow^n x^nBy$ **in G**, **where** $n \geq 1$, G je $_{k\text{-}left}$ LG, $k >= 1$, $A \in N$, $B \in N$, $x \in T^k$, $|x| = k$, $y \in T^*$.
then $x^{n-1}<B>xy \vdash^n <A>$ in M, **where** $<A> \in Q$, $<B> \in Q$, M je $_{k\text{-}left}$ $GJFA$.

Báze

Nechť $n = 1$:
If $A \Rightarrow^1 x^1By$ in G **then** $<B>xy \vdash^1 <A>$ in M.
Na základě Algoritmus VI. (první věta konstrukce množiny P) toto **tvrzení platí**.

Indukční hypotéza

Předpokládejme, že máme přirozené číslo $c \geq 1$ a toto tvrzení platí pro všechna čísla $1 \leq n \leq c$**.**

Indukční krok

Dokažme, že platí tvrzení i pro $c + 1$.
Nechť $n = c + 1$:
If $A \Rightarrow^{c+1} x^{c+1}By$ **in** G **then** $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M, **where** $A \in N$, $B \in N$, $x \in T^k$, $|x| = k$, $<A> \in Q$, $<B> \in Q$, M je $_{k\text{-}left}$ $GJFA$, G je LG.
Vyjádříme $A \Rightarrow x^{c+1}By$ **in** G jako:

$A \Rightarrow^c x^c B'z \Rightarrow^1 x^c x^1 Byz$, **where** $B' \in N, z \in T^*$.
Vyjádříme $x^{c+1-1}<B>xy \vdash^{c+1} <A>$ **in** M jako:
$x^c x^{1-1}<B>xyz \vdash x^{c-1}<B'>x\,z \vdash^c <A>$, **where** $<B'> \in Q$.

Shrnutí a závěr důkazu

Tvrzení **If** $A \Rightarrow^c x^c B'z$ **in** G **then** $x^{c-1}<B'>x\,z \vdash^c <A>$ **in** M platí na základě indukční hypotézy.
Připomeňme si první větu konstrukce množiny P popsanou v Algoritmus VI. Gramatické pravidlo ve tvaru $A \rightarrow xBy \in P$ in G je vytvořeno z pravidla $<B>xy \rightarrow <A> \in R$ in M.
Z toho vyplývá pravdivost věty **if** $x^c B'z \Rightarrow x^c x^1 Byz$ **in** G **then** $x^c<B>xyz \vdash^1 x^{c-1}<B'>xz$ **in M.**
Na základě principu indukce bylo **tvrzení claimu B dokázáno**.

Příklad

Nechť $k = 1$. Uvažujme $_{k\text{-}left}$ *GJFA* $M = (\{<S>, <A>, <B>, <C>\}, \{u, v, w\}, R, <S>, <A>)$, kde R obsahuje množinu pravidel: $<B>uvw \rightarrow <A>$, $<C>vw \rightarrow <B>$, $<S>uuuu \rightarrow <C>$. Startujícím stavem je $<S>$.
K převodu $_{k\text{-}left}$ *GJFA M* na ekvivalentní $_{k\text{-}left}$ *LG* $G = (\{A, B, C\}, \{u, v, w\}, P, S)$, budeme postupovat podle algoritmu konstrukce množiny P (Algoritmus VI.):
$<B>uvw \rightarrow <A>$ **in** R má ekvivalent $A \rightarrow uBvw$ **in** P.
$<C>vw \rightarrow <B>$ **in** R má ekvivalent $B \rightarrow vCw$ **in** P.
$<S>uuuu \rightarrow <C>$ **in** R má ekvivalent $C \rightarrow uuuu$ **in** P.
Práce automatu krok po kroku:
$uv<S>uuuuwvw \vdash u<C>v\,wvw \vdash <B>uvw \vdash <A>$.
Derivace krok po kroku:
$A \Rightarrow uBvw \Rightarrow uvCwvw \Rightarrow uvuuuuwvw$.

6.3 K-right automaty a gramatiky

Tato kapitola demonstruje veškeré důkazy na k-left lineárních gramatikách a skákajících automatech. Důkazy pro k-right prostředky jsou identické kromě rozdílu směru. Více o odlišnosti k-left a k-right GJFA včetně příslušných algoritmů poskytne kapitola 1.4

7 Závěr

Skákající automaty jsou bezpochyby zajímavé téma k prozkoumání. Jejich síla sahá daleko za regulární jazyky a nabízí se navržení a prozkoumání rozmanitého množství různých modifikací. Před vznikem této práce již byly navrženy obecné skákající automaty. Velkou předností těchto automatů je, že nemusí číst vstupní pásku pouze v jednom směru, navíc ani nemusí jejich činnost zahájit na začátku vstupu, nýbrž ze kteréhokoliv místa na vstupní pásce. Tyto rozmanité možnosti čtení vstupu jsou ale paradoxně zároveň příčina zásadně omezující sílu těchto automatů. Najdeme četné případy jazyků, pro které dokážeme navrhnout příslušný skákající automat, který daný jazyk dokáže přijmout, avšak z důvodu své přílišné obecnosti následně automat jako přijímané řetězce označí i řetězce nepatřící do daného jazyka. Zmíněný případ nastane např. u jazyka $\{a^n b^n : n \geq 0\}$.

V této práci jsou navrženy modifikace obecných skákajících automatů, tzv. k-left a k-right skákající automaty. Obecnost *GJFA* je omezena na pravidelné skoky pouze v jednom směru a o pevně daný počet symbolů na vstupní pásce. Skoky také oproti *GJFA* nejsou dobrovolné, ale jsou provedeny při každém kroku automatu.

Práce odhaluje ekvivalentní model k navržené verzi automatů, k-right a k-left lineární gramatiky. Byl proveden důkaz, že ke každé $_{k\text{-}left}$ *LG* příp. $_{k\text{-}right}$ *LG* dokážeme vytvořit ekvivalentní $_{k\text{-}left}$ *GJFA* příp. $_{k\text{-}right}$ *GJFA*. Princip tvorby ekvivalentních modelů popisují Algoritmus V. a Algoritmus VI. *K-GJFA* můžeme označit za modely pro popis jazyků vygenerovaných k-left a k-right lineárními gramatikami. Z toho vyplývá, že síla těchto automatů sahá až po množinu lineárních jazyků, která je podmnožinou bezkontextových jazyků.

Na základě zjištění, že *k-GJFA* dokáží přijmout i některé bezkontextové jazyky, byla zvážena myšlenka, zda lze tento model automatu využít při syntaktické analýze. *K-GJFA* si však poradí pouze se základními výrazovými konstrukcemi, např. výrazem sčítajícím hodnoty proměnné *i* (*i*+*i*+*i*+*i*+...). Ale jejich schopnost přijímat výrazy se zastaví již na tomto typu výrazu: *i*+*i***j*. Tj. výrazu, ve kterém se kombinují různé typy matematických operátorů nebo obsahují více párů závorek. Zmíněná slabina se projeví kromě matematických výrazu i u konstrukcí programovacího jazyka. Proč? Jelikož matematické konstrukce a úseky kódů programovacích jazyků jsou vygenerovány gramatikami, které obsahují dva a více neterminálů na pravé straně pravidla, např. $S \rightarrow AB$. Takováto pravidla gramatik však nedokážeme převést na ekvivalentní pravidla *k-GJFA*.

Protože *k-GJFA* dokáží zpracovat některé bezkontextové jazyky, byly zváženy možnosti a okolnosti, za kterých by bylo možné model *k-GJFA* při syntaktické analýze využít. Byly uvažovány takové možnosti, které by dokázaly eliminovat neschopnost zpracovávat pravidla gramatiky s více neterminály na pravé straně. Zároveň se musí jednat o variantu, která v některých ohledech předčí zásobníkové automaty, které se na zpracování bezkontextových jazyků využívají. *ZA* kromě standardního zpracování vstupu navíc pracuje s tzv. zásobníkem, který v průběhu práce využívá na ukládání symbolů a vracení zpět - to je důležité rozšíření oproti běžným konečným a skákajícím automatům. Bez tohoto rozšíření by nebylo možné zpracovat páry závorek ve výrazech ani kombinaci operátorů. Nicméně *ZA* může tímto způsobem pracovat se zásobníkem pouze tzv. LIFO přístupem (poslední symbol dovnitř, první ven). Na rozdíl od skákajících automatů musí *ZA* číst vstup od začátku a pouze v jednom směru.

Navržena byla varianta typu *k-GJFA*, která potlačí alespoň částečně omezení *ZA* a zároveň je pro specifické účely syntaktické analýzy vhodnější z hlediska optimalizace a rychlosti zpracování vstupu. Řešení má podobu sestavy *k-GJFA*, která v sobě pro účely syntaktické analýzy spojuje potenciál k-left i k-right skákajících automatů. Oproti zásobníkovým automatům vstup rychleji

zpracovávají, jelikož zpracování probíhá z více směrů paralelně. Nevýhodou je složitější implementace a možnost užití navrženého řešení pouze pro jednodušší výrazy a konstrukce.

Vzniklý text práce ukazuje rovněž srovnání jednotlivých typů automatů z hlediska jejich síly. Byly prokázány překvapivé závěry. Mezi nejdůležitější z nich patří vztahy mezi ${}_{k\text{-}left}GJFA$ a obecným *GJFA* a mezi ${}_{k\text{-}left}GJFA$ a *JFA*. Výsledkem zkoumání je, že ${}_{k\text{-}left}GJFA \not\subset GJFA$ a $JFA \not\subset {}_{k\text{-}left}GJFA$. Z prvního tvrzení vyplývá fakt, že tím, že omezíme možnosti obecných skákajících automatů, dokážeme získat modely skákajícího automatu pro některé z jazyků, které obecné *GJFA* nedokázaly zpracovat právě z důvodu jejich přílišné obecnosti skoků a libovolného principu zpracování vstupní pásky.

Co se týká doporučeného budoucího zkoumání *k-GJFA*, zejména je vhodné se zaměřit na jejich způsob implementace, konkrétně např. efektivního algoritmu pro hledání startujícího stavu. Dále navrhnout další verze skákajících automatů, které mohou být kompromisem mezi obecnými *GJFA* a velmi omezenými *k-GJFA*. Zajímavým předmětem zkoumání s motivací optimalizace překladačů a zpracování složitějších skupin jazyků je modifikace skákajících automatů za účelem zpracovávání bezkontextových případně kontextových jazyků. Uvažovat otázku, zda bychom mohli skákající automat obohatit o rozšíření, které používá zásobníkový automat, nebo rozšířit skákající automaty o jiné nástroje, které používají složitější formální prostředky pro zpracování jazyků.

Hlavní myšlenka dalšího výzkumu je tedy omezení přílišné obecnosti GJFA (optimalizaci skoků, konkrétně jejich počet a četnost) a zároveň zajistit, aby omezení nebylo příliš velké a nově navržené řešení tak mělo mnohonásobně větší sílu než k-*GJFA*. Po navržení takové modifikace můžeme do modelu zavést a ohodnotit přínos některých z rozšíření, která doposud využívají pokročilé formální prostředky.

Velkou motivací obecně pro výzkum a rozvoj novým formálních prostředků je přínos z hlediska širšího a přesnějšího popisu rodin formálních jazyků, dále eliminace negativních vlastností dočasných modelů a také optimalizace praktického nasazení těchto prostředků při lexikální a syntaktické analýze, konkrétně v překladačích a kompilátorech programovacích jazyků.

Literatura

[1] MEDUNA, Alexander a Petr ZEMEK. Jumping Finite Automata. *International Journal of Foundations of Computer Science* [online]. 2012, **23**(7), 1555-1578 [cit. 2016-04-27]. ISSN 0129-0541. Dostupné z: http://dx.doi.org/10.1142/S0129054112500244

[2] MEDUNA, Alexander a Petr ZEMEK. *Jumping Finite Automata* [online]. b.r.. Dostupné také z: http://talks.petrzemek.net/Petr-Zemek_-_2012-12-10_-_Jumping.Finite.Automata.pdf

[3] *Materiály k předmětu IFJ: Formální jazyky a překladače - doplňující informace* [online]. b.r. [cit. Poslední aktualizace: 22. září 2015]. Dostupné z: www.fit.vutbr.cz/study/courses/IFJ/public/materials/

[4] MELICHAR, Bořivoj. *Gramatiky a automaty*. Praha: Vydavatelství ČVUT, 1979. ISBN 3202.

[5] MEDUNA, Alexander. *Formal languages and computation: models and their applications*. 1. New York: CRC Press, c2014, xix, 295 s. ISBN 978-1-4665-1345-7.

[6] MEDUNA, Alexander a Petr ZEMEK. *Regulated grammars and automata*. New York: Springer, c2014, xx, 694 s. ISBN 978-1-4939-0368-9.

[7] MEDUNA, Alexander. *Language and automata theory and applications: 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. pPoceedings*. 2013th Edition. New York: Springer, 2013, p. cm. ISBN 36-423-7063-2.

[8] ROZENBERG, Grzegorz (ed.) a Arto SALOMAA (ed.). *Handbook of formal languages*. Volume 1. Berlin: Springer-Verlag, c1997, xx, 625 s. ISBN 35-406-1486-9.

Příloha A: Manuál k programu

Popis programu

Přiložený program obsahuje algoritmy pro převod $_{k\text{-}left}$ *GJFA* na ekvivalentní $_{k\text{-}left}$ *LG* a naopak. Tyto algoritmy jsou popsané a dokázané v technické zprávě v kapitole 6. Program také demonstruje krok po kroku činnost $_{k\text{-}left}$ *GJFA* na uživatelem zvoleném řetězci.

Jak program spustit a ovládat

Program je nutno spustit na stanici s nainstalovaným interpretem Python příkazem ve tvaru: *interpret program parametry,* kde *program* představuje název skriptu, v našem případě *jfa.py, interpret* je *python 3* a parametry programu jsou popsány níže.
Program je možno spouštět s těmito parametry:

--help	vytiskne nápovědu k programu na standardní výstup
--input=*filename*	určuje vstupní soubor
--output=*filename*	určuje výstupní soubor
-a2g, --automata2grammar	převede GJFA na vstupu na lineární gramatiku
-g2a, --grammar2automata	převede lineární gramatiku uvedenou na vstupu na GJFA
-a, --automaton	předvede ukázkové zpracování vstupního řetězce uvedeného

ve vstupním souboru mezi znaky "!" příslušným GJFA.

Poznámky k parametrům programu

- parametry --*automata2grammar* a --*grammar2automata*: je nutno zadat právě jeden z těchto parametrů pro specifikování typu převodu
- pokud není zadán vstupní soubor, použije se místo něj standardní vstup, pokud není zadán výstupní soubor, výsledek se bude zapisovat na standardní výstup
- parametr --*automaton* je možno kombinovat s parametrem --*automata2grammar* i --*grammar2automata.* Pokud je zadán současně s --*grammar2automata,* provede se ukázkové zpracování řetězce ze vstupního souboru automatem ekvivalentním k LG specifikované ve vstupním souboru. Pokud je zadán s parametrem --*automata2grammar,* demonstrace řetězce je provedena pomocí $_{k\text{-}left}$ GJFA uvedeného přímo ze vstupního souboru

Příklady spuštění

python 3 jfa.py --help #vytiskne nápovědu k programu

python 3 jfa.py --grammar2automata --input=gramatika_linear #převede $_{k\text{-}left}$ *LG* ze vstupního souboru *gramatika_linear* na ekvivalentní $_{k\text{-}left}$ *GJFA* a výsledek zapíše na standardní výstup (*stdout*)

python 3 jfa.py --automata2grammar --input=soubor1 --output=soubor2 -a #převede $_{k\text{-}left}$ *GJFA* ze souboru *soubor1* na ekvivalentní $_{k\text{-}left}$ *LG,* výsledek spolu s vyhodnocením zpracováním řetězce uvedeného ve vstupním souboru *soubor1* uvnitř znaků „!" zapíše do výstupního souboru *soubor2*

Chybové kódy

V programu jsou použity tyto chybové kódy:
1 – špatně zadané parametry programu

2 – nepodařil se otevřít vstupní soubor nebo je chybný
3 – nepodařil se otevřít výstupní soubor nebo je chybný
60 - syntaktická chyba v zápisu automatu, popř. gramatiky
61 – chyba v interpunkci, např. konstrukce ukončená posloupností znaků ,)
62 – chyba v sémantice skákajícího automatu, zadaný automat je v rozporu s definicí $_{k\text{-}left}$ *GJFA*, např. stav použitý v pravidlech není přítomen v množině stavů, pravidla nesplňují předpisy skákajících automatů, např. $|x| = k$. Definice $_{k\text{-}left}$ *GJFA* je popsaná v kapitole 2
63 – chyba v sémantice $_{k\text{-}left}$ *LG*. Definice se nachází v kapitole 2

Formát vstupního souboru

Pro vstupní soubor obsahující lineární gramatiku příp. skákající automat platí tato pravidla:

- stavy automatu zapisujeme v hranatých závorkách, jejich názvy mohou obsahovat velká a malá písmena + číslice, např. *<Stav1>*
- neterminály u lineární gramatiky značíme velkým písmenem ohraničené znaky $, např. *$Stav1$*
- komentáře do konce řádku začínají znakem #.
- bílé znaky jako konec řádku (\n i \r), mezera či tabulátor jsou kromě případu užití těchto znaků jako vstupních symbolů abecedy při zpracování vstupního souboru ignorovány
- vstupní symbol je reprezentovaný libovolnou číslicí nebo písmenem uzavřeném v apostrofu (ascii 39), případně lze využít i metaznaky a všechny bílé znaky, tj. ’(’, ‘)’, ‘{‘, ‘}’, ‘-’, ‘>‘, ‘,’, ‘.’, ‘#’, ‘ ‘. Apostrof sám být symbolem nesmí! (nejsou povolena epsilon pravidla)
- u stavů i vstupních symbolů **záleží** na velikosti písmen.
- program nepovoluje zadávání epsilon pravidel. (zdůvodnění viz kapitola 5)

Přesná specifikace vstupního souboru v případě parametru -- automata2grammar

(	#levou kulatou závorkou začíná formální popis automatu
k = číslo,	# parametr *k* značí stupeň skákajícího automatu (viz definice)
{<stav1>, <stav2>,…},	#množina stavů *Q*
{‘x’, ‘y’,…},	#množina vstupních symbolů a
{<stav1>‘xy’ -> <stav2>,…},	#konečná množina pravidel ve tvaru *Qa -> Q*
<stav1> ,	#startující stav
{<stav2>,…}	#množina koncových stavů
)	# pravá závorka ukončující zápis automatu

Poznámka: Mezi jednotlivými oddělujícími znaky, jako např. středník a závorka, může být libovolný počet bílých znaků.

Příklad vstupního souboru s $_{k\text{-}left}$ GJFA

(k = 1, {<s1>, <f1>,# nějaký komentář
<k18b>}, # nějaký komentář
{‘a’, ‘)’, ‘b’ }, {
<s1> ‘)a’-><k18b>,<k18b> ‘b’ -><f1>}, # následuje komponenta definující počáteční stav
<s1> ,

{<f1>}) # koncové stavy a ukončení definice automatu
 # zde může následovat libovolný počet bílých znaků nebo komentářů

Přesná specifikace vstupního souboru v případě --grammar2automata

(	#levou kulatou závorkou začíná formální popis gramatiky
k = číslo,	# parametr *k* značí stupeň gramatiky (viz definice)
{A, B,…},	#množina neterminálů *N*
{‘x’, ‘y’,…},	#množina terminálů T
{A ->‘x’B ‘yyyxy’,…},	#konečná množina pravidel ve tvaru $N \rightarrow T^k N\, T^*$
A	#startující neterminál
)	#pravá závorka ukončující zápis lineární gramatiky

Příklad lineární gramatiky ve vstupním souboru

(k = 1, { A,# nějaký komentář
B, C}, # nějaký komentář
{‘u’, ‘v’, ‘w’ }, {
A -> ‘u’ B ‘v‘ ,B -> ’v’ C ’w’ , C -> ‘uuuu’}, # definice počáteční neterminál
A
)

Ukázka kroků k-GJFA

Řetězec, na kterém má program oddemonstrovat jeho jednotlivé kroky, doplníme na konec vstupního souboru mezi znaky „!“, např. *!retezec!*.

Příloha B: Ukázka práce programu:

Vstupní soubor gramatika2:

(k = 1, {A,

B, C}, # nějaký komentář
{'u', 'v', 'w' }, {
A ->'u' B 'v' ,B -> 'v' C 'w' , C -> 'uuuu' }, # definice počáteční neterminál A
)

!uvuuuuwv!

Příkaz:

python3 jfa.py -g2a --automaton --input=gramatika2 --output=vysledek

Výstupní soubor vysledek:

(
{<A>, <B>, <C>, <S>},
{'u', 'v', 'w'},
{
<B> 'uv' -> <A>,
<C> 'vw' -> <B>,
<S> 'uuuu' -> <C> },
<S>,
{<A>}
)

****
Step by step na zadanem retezci:uvuuuuwv
uv<S>wv
u<C>vwv
<B>uv
<A>

Zadany retezec byl prijat danym k-left skakajicim automatem!

Vstupní soubor automat:

(k =2, {<s>,<ss>, <ddd>, <eee>} ,

{'a','b', 'n'}, {<s>'n' -> <eee>,<eee>'ba' -> <ss>,<ss> 'abaa'-> <ddd>},####
<s>,
{<ddd>}
)

!ababbanaaaafrika !

Příkaz:

python3 jfa.py -a2g --automaton --input=automat --output=vysledek2

Výstupní soubor vysledek2:

{ss, ddd, eee},
{'a', 'b', 'n'},
{
eee -> 'n' ,
ss -> 'ba' eee '',
ddd -> 'ab' ss 'aa'},
ddd
)

****
Step by step na zadanem retezci:ababbanaaaafrika
ababba<S>aaaafrika
abab<eee>baaaaafrika
ab<ss>abaaaafrika
<ddd>abaafrika

Zadany retezec nebyl prijat danym k-left skakajicim automatem!

Příloha C: Obsah CD

CD přiložené k bakalářské práci obsahuje tyto soubory a složky:

- **src** – složka obsahující skript *jfa.py*
- **examples** – složka obsahují ukázkové vstupní soubory *automat*, *gramatika* a *gramatika2* soubor *automat* obsahuje příklad $_{k\text{-}left}$ *GJFA* skákajícího automatu spolu s příkladem řetězce, který není přijímán daným skákajícím automatem. $_{k\text{-}left}$ *LG* se nachází v souboru *gramatika*. Poslední soubor *gramatika2* obsahuje $_{k\text{-}left}$ *LG* gramatiku spolu s příkladem řetězce, který je přijímán ekvivalentním automatem k této gramatice.
- **doc** - složka obsahující text práce ve formátu PDF a docx.