VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

TRANSFORMACE FRONTOVÝCH GRAMATIK

TRANSFORMATION OF QUEUE GRAMMARS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. DAVID HOLAS**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

140952

Ústav: Ústav informačních systémů (UIFS)
Student: **Holas David, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Matematické metody

Název: **Transformace frontových gramatik**
Kategorie: Teoretická informatika
Akademický rok: 2022/23

Zadání:

1. Seznamte se s pokročilými řízenými gramatikami a dalšími pojmy z teorie formálních jazyků. Prozkoumejte existující výsledky se zaměřením na frontové gramatiky.
2. Nastudujte existující transformace frontových gramatik na jiné modely. Prozkoumejte vlastnosti těchto transformací včetně studia potřebných normálních forem frontových gramatik. Případně navrhněte nové vhodné normální formy.
3. Dle pokynů vedoucího navrhněte pokročilou transformaci frontových gramatik s důrazem na minimalizaci kontextových závislostí ve výsledném modelu, což souvisí se studiem popisné složitosti výsledného modelu (např. substituční-selektivní gramatiky s omezením selekčních vzorů nebo gramatiky s rozptýleným kontextem s významně omezenou popisnou složitostí).
4. Správnost navržené transformace dokažte nebo experimentálně ověřte s využitím efektivní implementace takovéto transformace.
5. Zhodnoťte danou transformaci včetně různých druhů složitosti (popisná, časová, prostorová).

Literatura:

- KŘIVKA Z., MEDUNA A. Scattered Context Grammars with One Non-Context-Free Production are Computationally Complete. *Fundamenta Informaticae*, roč. 179, č. 4, 2021, s. 361-384. ISSN 0169-2968.
- KLEIJN H.C.M., ROZENBERG G. Multi grammars. Intern. J. Computer Maths., roč. 12, 1983, s. 177-201.
- MEDUNA A., ZEMEK P. Regulated Grammars and Automata. New York: Springer US, 2014. ISBN 978-1-4939-0368-9.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a část bodu 3.

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

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 17.5.2023
Datum schválení: 26.10.2022

Abstrakt

Diplomová práce je rozdělena do dvou částí. První část spočívá v opravě chyby v algoritmickém převodu frontové gramatiky do 1. normální formy navrženém jiném odborném článku. Algoritmus byl analyzován a úspěšně opraven. Druhá část se zabývá návrhem nové transformace na gramatiky s rozptýleným kontextem s omezenou popisnou složitostí. Práce obsahuje důkaz její korektnosti a studium její složitosti. Byla vytvořena konzolová aplikace, která pomáhá automaticky analyzovat příslušné transformace.

Abstract

The master thesis is divided into two parts. First part focuses on fixing incorrect transformation algorithm of queue grammar into a first normal form proposed in other paper. The algorithm was analysed and successfully corrected. Second part focuses on proposing a new tranformation to scattered context grammars with reduced descriptional complexity. The thesis contains a proof of its correctness and contains its complexity analysis. Console application was created to help analyze the respective transformations.

Klíčová slova

Frontové gramatiky, 1. normální forma, 2. normální forma, normalizace, transformace, gramatiky s rozptýleným kontextem, jedno kontextové pravidlo, omezení popisné složitosti, jazyk Swift.

Keywords

Queue grammars, first normal form, second normal form, normalization, normalzation, tranformation, scattered context grammars, single context-sensitive rule, reduced descriptional complexity, Swift language.

Citace

HOLAS, David. *Transformace frontových gramatik*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Transformace frontových gramatik

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Křivky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Holas
17. května 2023

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Zbyňku Křivkovi, Ph.D. za rady, připomínky a pomoc při řešení této práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá frontovými gramatikami a jejich převodem do normálních forem nebo na jiné gramatiky. Frontové gramatiky jsou modely výpočtu se silou Turingova stroje a popisují tak třídu rekurzivně vyčíslitelných jazyků. Normální formy frontových gramatik jsou významné zvláště pro zavádění dalších komplikovanějších transformací. To díky tomu, že do frontové gramatiky zavádějí nové vlastnosti, ale současně zachovávají její výpočetní sílu a jazyk, který gramatika popisuje.

Práce bude rozdělena do dvou částí. První část spočívá v opravě chyby v algoritmické transformaci frontové gramatiky do 1. normální formy. Tento algoritmus byl představen v článku [6] a jeho neúplná korektnost byla odhalena v bakalářské práci [1]. Cílem této části práce je algoritmus analyzovat, identifikovat chyby a opravit je tak, aby byl algoritmus korektní. Protože frontové gramatiky po algoritmické transformaci do 1. normální formy jsou velmi komplexní, je nutné vytvořit aplikaci, která umožní automatickou analýzu složitějších gramatik. Tato aplikace následně může být použita i k hromadnému testování gramatik, které značně napoví, zda je algoritmus korektní nebo ne.

Cílem druhé části práce je společně s vedoucím navrhnout novou transformaci frontové gramatiky na gramatiku s rozptýleným kontextem a s jedním kontextovým pravidlem. Konkrétně je cílem co nejvíce omezit popisnou složitost výsledné gramatiky tak, že bude v gramatice zaručen konstantní počet neterminálů. Jinými slovy by počet neterminálů výsledné gramatiky měl být co nejnižší a současně by měla být zachována nízká kontextová závislost díky omezení na jedno kontextové pravidlo.

Dále je nutné dokázat korektnost takto navržené transformace a analyzovat ji. Do toho spadá analýza složitostí, ale i zkoumání následků dalšího snižování počtu neterminálů. Protože gramatiky s rozptýleným kontextem, které jsou výsledkem transformace z frontové gramatiky, jsou opět velmi komplexní, je opět výhodné vytvořit aplikaci. Tato aplikace by měla implementovat transformaci a umožnit snadnější analýzu výsledných gramatik, čímž se zjednoduší úvahy o korektnosti transformace. Transformované gramatiky s rozptýleným kontextem jsou výpočetně náročnější než frontové gramatiky a korektnost transformace tedy bude ověřena formou důkazu namísto hromadného testování gramatik.

Kapitola 2

Studované gramatiky

Tato kapitola definuje jednotlivé gramatiky, kterými se práce zabývá. Jedná se o gramatiky o síle Turingova stroje, které popisují třídu rekurzivně vyčíslitelných jazyků.

2.1 Frontové gramatiky

Frontová gramatika je šestice $Q = (V, T, W, F, s, R)$, kde

- V je abeceda symbolů,
- W je abeceda stavů,
- $T \subseteq V$ je abeceda terminálů,
- $F \subseteq W$ je abeceda koncových stavů,
- $s \in (V - T)(W - F)$ je počáteční větná forma,
- $R \subseteq (V \times (W - F)) \times (V^* \times W)$ je konečná relace reprezentující množinu pravidel.

Pokud existují 2 řetězce $u = arp$ a $v = rzq$, kde $a \in V$, $r, z \in V^*$ a $p, q \in W$, a pravidlo $(a, p, z, q) \in R$, potom v gramatice Q lze provést derivační krok $u \Rightarrow v\ [(a, p, z, q)]$. Derivační krok lze zapsat i zkráceným zápisem bez specifikace pravidla, $u \Rightarrow v$. Jako v ostatních gramatikách, $\Rightarrow^n$ značí sekvenci derivačních kroků, na jejímž základě jsou definovány sekvence $\Rightarrow^*$ a $\Rightarrow^+$ standardním způsobem.

Jazyk frontové gramatiky je definován jako

$$L(Q) = \{w \in T^* \mid s \Rightarrow^* wf, f \in F\}.$$

Definice frontové gramatiky byla převzata z článků [4] a [6]. Článek [3] dokázal, že třída jazyků frontových gramatik je shodná s třídou rekurzivně vyčíslitelných jazyků $\mathcal{L}_{\mathrm{RE}}$. Příklad 2.1 ukazuje derivaci řetězce ve frontové gramatice.

V této práci je zavedena nová notace frontových gramatik, kde Q_n značí gramatiku definovanou v souboru `example`n`.json` na přiloženém médiu a N_n značí výslednou gramatiku po algoritmické normalizaci gramatiky Q_n. Derivační krok může být navíc označen římskou číslicí, $\Rightarrow_n$, kde n je pouze identifikátor kroku, díky kterému je později možné se v práci na tento derivační krok odkázat.

Příklad 2.1. Mějme gramatiku $Q_{12} = (V, T, W, F, s, R)$, kde $V = \{S, a, b\}$, $T = \{a, b\}$, $W = \{e, f\}$, $F = \{f\}$ a $s = Se$. Množina pravidel R obsahuje pravidla:

1. (S, e, bSa, e);
2. (S, e, ε, f);
3. (a, e, a, e);
4. (b, e, b, e).

Jazyk gramatiky je $L(Q_{12}) = \{a^n b^n \mid n \geq 0\}$. Jednotlivá pravidla gramatiky jsou očíslována tak, aby se na ně bylo možné v sekvenci derivačních kroků odkazovat a zvýšila se tak přehlednost zápisu.

V této gramatice budeme derivovat řetězec $abba \in L(Q_{12})$. Sekvence derivačních kroků bude vypadat následovně:

$$Se \Rightarrow bSae[1] \Rightarrow Sabe[4] \Rightarrow abbSae[1] \Rightarrow bbSaae[3] \Rightarrow bSaabe[4] \Rightarrow Saabbe[4] \Rightarrow aabbf[2].$$

2.1.1 Levé rozšíření

Levě rozšířená frontová gramatika $Q = (V, T, W, F, s, R)$, kde V, T, W, F, s a R mají stejný význam jako v běžné frontové gramatice. Navíc ale existuje symbol $\# \notin V \cup W$.

Pokud existují dva řetězce $u = w\#arp$ a $v = wa\#rzq$, kde $a \in V$, $w, r, z \in V^*$ a $p, q \in W$, a pravidlo $(a, p, z, q) \in R$, pak lze v gramatice Q provést derivační krok $u \Rightarrow v\ [(a, p, z, q)]$. Derivační krok lze opět zapsat i bez specifikace pravidla pomocí $u \Rightarrow v$ a standardním způsobem rozšířit na $\Rightarrow^n$, $\Rightarrow^+$ a $\Rightarrow^*$.

Jazyk levě rozšířené frontové gramatiky je definován jako

$$L(Q) = \{w \in T^* \mid \#S \Rightarrow^* v\#wf, v \in V^*, f \in F\}.$$

Tato definice byla převzata z článku [4].

2.2 Gramatiky s rozptýleným kontextem

Gramatika s rozptýleným kontextem (jinak také zkráceně *rozptýlená gramatika*) je čtveřice $G = (N, T, P, S)$, kde

- N je abeceda neterminálů,
- T je abeceda terminálů,
- $P \subseteq \bigcup_{n=1}^{\infty} N^n \times ((N \cup T)^*)^n$ je konečná množina pravidel,
- $S \in N$ je počáteční symbol.

Pravidla gramatiky s rozptýleným kontextem z množiny P mají tvar $(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n)$, kde $n \geq 1$. Pravidla se dělí na bezkontextová, kde $n = 1$, a kontextová, kde $n \geq 2$. Bezkontextové pravidlo lze zkráceně zapsat jako $A_1 \to x_1$.

Pokud existují dva řetězce

$$\begin{aligned} u &= u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1}, \\ v &= u_1 x_1 u_2 x_2 \ldots u_n x_n u_{n+1}, \end{aligned}$$

kde $u_1, u_2, \ldots, u_{n+1} \in (N \cup T)^*$ a pravidlo $p \in P$, kde

$$p = (A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n),$$

pak lze provést v gramatice G derivační krok $u \Rightarrow v$ $[p]$. Derivační krok lze zapsat i zkráceným zápisem bez specifikace pravidla $u \Rightarrow v$. Jako v ostatních gramatikách $\Rightarrow^n$ značí sekvenci n derivačních kroků, na jejímž základě jsou standardním způsobem definovány sekvence $\Rightarrow^*$ a $\Rightarrow^+$. Derivační krok může být opět navíc označen římskou číslicí, $\Rightarrow_n$, kde n je pouze identifikátor kroku, díky kterému je později možné se v práci na tento derivační krok odkázat.

Jazyk rozptýlené gramatiky je definován jako

$$L(G) = \{w \in T \mid S \Rightarrow w\}.$$

Tato definice byla převzata z knihy [2], z článku [6] a z práce [1].

Kapitola 3

Transformace do 1. normální formy

Každou frontovou gramatiku lze transformovat do 1. či 2. normální formy tak, že se nezmění jazyk gramatiky [6]. Tato kapitola se věnuje převodu do 1. normální formy a jeho algoritmu, nicméně 2. normální forma je pro úplnost zavedena také. Algoritmus transformace byl publikován v článku [6], nicméně v bakalářské práci [1] byla naznačena jeho neúplná korektnost. V této kapitole tedy bude algoritmus analyzován a opraven.

Frontová gramatika je v 1. normální formě, pokud pravá strana každého pravidla obsahuje buď pouze terminály, nebo pouze neterminály. Současně lze v gramatice přepisovat pouze neterminály a aplikovat pravidla pouze ve stavech, které nejsou koncové. Formálně 1. normální formu zavádí definice 3.1.

Definice 3.1. Frontová gramatika $Q = (V, T, W, F, s, R)$ je v 1. normální formě právě tehdy, když platí:

$$\forall (a, p, x, q) \in R : a \in V - T \wedge p \in W - F \wedge (x \in T^* \vee x \in (V - T)^*) \text{ [6].}$$

Pokud frontová gramatika splňuje 1. normální formu, pak je i v 2. normální formě pokud pro každý stav existuje nanejvýš jeden neterminál, který v něm lze přepsat. Formálně 2. normální formu zavádí definice 3.2.

Definice 3.2. Frontová gramatika $Q = (V, T, W, F, s, R)$ je v 2. normální formě právě tehdy, když je v 1. normální formě a současně platí:

$$\forall p \in W - F : (a_1, p, y_1, q_1) \in R \wedge (a_2, p, y_2, q_2) \in R \implies a_1 = a_2 \text{ [6].}$$

Algoritmicky by měla být transfoce do 1. normální formy (dále také zkráceně jako *normalizace*) možná pomocí následujícího postupu. Korektnost tohoto algoritmu byla vyvrácena v bakalářské práci [1], nicméně je nutné se nejprve seznámit s nekorektní verzí algoritmu.

3.1 Algoritmus transformace do 1. normální formy

Mějme frontovou gramatiku $Q = (V_Q, T, W_Q, F_Q, s_Q, R_Q)$, kde $s_Q = a_0 q_0$. Dále definujme množiny $W'_Q = \{q' : q \in W_Q\}$, $W''_Q = \{q'' : q \in W_Q\}$, $V'_Q = \{a' : a \in V_Q\}$ a množinu *složených stavů* $U = \{\langle y, q \rangle : y \in T^* \wedge (a, p, xy, q) \in R_Q\}$. Na závěr definujeme bijekci α z W_Q do W'_Q jako $\alpha(q) = q'$, bijekci β z W_Q do W''_Q jako $\beta(q) = q''$. Analogicky definujeme bijekci δ z V_Q do V'_Q jako $\delta(a) = a'$, kterou navíc standardně rozšíříme na řetězce z V_Q^* do $(V'_Q)^*$. Je zřejmé, že bijekce α je na řetězcích rovněž úplná.

Zavedeme nový *dělicí symbol* 1, kde $1 \notin V'_Q \cup T$ a nový koncový stav f_N, kde $f_N \notin W'_Q \cup W''_Q \cup U$. S jejich pomocí sestrojíme frontovou gramatiku v 1. normální formě $N = (V_N, T, W_N, F_N, s_N, R_N)$, kde $V_N = V'_Q \cup \{1\} \cup T$, $W_N = W'_Q \cup W''_Q \cup \{f_N\} \cup U$, $F_N = \{f_N\}$ a $s_N = \delta(s_0)\alpha(q_0)$. Množina pravidel R_N je sestrojena v následujících krocích aplikovaných na každé pravidlo z R_Q:

1. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x, y \in V_Q^*$ a $q \in W_Q$, pak $(\delta(a), \alpha(p), \delta(x)\delta(y), \alpha(q))$ a $(\delta(a), \alpha(p), \delta(x)1\delta(y), \alpha(q))$ patří do R_N;
2. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in V_Q^*$, $y \in T^*$, $q \in W_Q$ a $\langle y, q\rangle \in U$, pak $(\delta(a), \alpha(p), \delta(x), \langle y, q\rangle)$ a $(1, \langle y, q\rangle, y, \beta(q))$ patří do R_N;
3. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in W_Q$, pak $(\delta(a), \beta(p), x, \beta(q))$ patří do R_N;
4. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in F_Q$, pak $(\delta(a), \beta(p), x, f)$ patří do R_N.

Tento algoritmus byl převzat z článku [6]. Normalizace konkrétní gramatiky Q_{12} je popsána v příkladu 3.1.

Příklad 3.1. Mějme gramatiku $Q_{12} = (V_Q, T, W_Q, F_Q, Se, R_Q)$, kde $V_Q = \{S, a, b\}$, $T = \{a, b\}$, $W_Q = \{e, f\}$, $F_Q = \{f\}$ a množina pravidel $R_Q = \{(S, e, bSa, e), (S, e, \varepsilon, f), (a, e, a, e), (b, e, b, e)\}$.

Při transformaci budeme postupovat podle algoritmu normalizace v podsekci 3.1. Speciální symboly necháme pro jednoduchost 1 a f_N a sestrojme pomocné množiny:

$$\begin{aligned} V' &= \{S', a', b'\}, \\ W' &= \{e', f'\}, \\ W'' &= \{e'', f''\} \text{ a} \\ U &= \{\langle\varepsilon, e\rangle, \langle a, e\rangle, \langle b, e\rangle, \langle\varepsilon, f\rangle\}. \end{aligned}$$

Z nich následně sestrojíme množiny potřebné k definici frontové gramatiky v 1. normální formě:

$$\begin{aligned} V_N &= V'_Q \cup \{1\} \cup T = \{S', a', b', 1, a, b\} \text{ a} \\ W_N &= W'_Q \cup W''_Q \cup \{f_N\} \cup U = \{e', f', e'', f'', f_N, \langle\varepsilon, e\rangle, \langle a, e\rangle, \langle b, e\rangle, \langle\varepsilon, f\rangle\}. \end{aligned}$$

Množinu pravidel R_N zkonstruujeme z jednotlivých pravidel gramatiky Q_{12} následovně:

Pravidlo $(S, e, bSa, e) \in R_Q$

V kroku 1 přidáme do R_N pravidla:

1. $(S', e', b'S'a', e')$;
2. $(S', e', 1b'S'a', e')$;
3. $(S', e', b'1S'a', e')$;
4. $(S', e', b'S'1a', e')$;
5. $(S', e', b'S'a'1, e')$.

V kroku 2 přidáme do R_N pravidla:

6. $(S', e', b'S'a', \langle \varepsilon, e \rangle)$;
7. $(1, \langle \varepsilon, e \rangle, \varepsilon, e'')$;
8. $(S', e', b'S', \langle a, e \rangle)$;
9. $(1, \langle a, e \rangle, a, e'')$.

Pravidlo $(S, e, \varepsilon, f) \in R_Q$

V kroku 1 přidáme do R_N pravidla:

10. $(S', e', \varepsilon, f')$;
11. $(S', e', 1, f')$.

V kroku 2 přidáme do R_N pravidla:

12. $(S', e', \varepsilon, \langle \varepsilon, f \rangle)$;
13. $(1, \langle \varepsilon, f \rangle, \varepsilon, f'')$.

V kroku 3 přidáme do R_N pravidlo:

14. $(S', e'', \varepsilon, f'')$.

V kroku 4 přidáme do R_N pravidlo:

15. $(S', e'', \varepsilon, f_N)$.

Pravidlo $(a, e, a, e) \in R_Q$

V kroku 1 přidáme do R_N pravidla:

16. (a', e', a', e');
17. $(a', e', 1a', e')$;
18. $(a', e', a'1, e')$.

V kroku 2 přidáme do R_N jen pravidla, která tam již nebyla přidána v krocích výše:

19. $(a', e', a', \langle \varepsilon, e \rangle)$;
20. $(a', e', \varepsilon, \langle a, e \rangle)$.

V kroku 3 přidáme do R_N pravidlo:

21. (a', e'', a, e'').

Pravidlo $(b, e, b, e) \in R_Q$

V kroku 1 přidáme do R_N pravidla:

22. (b', e', b', e');
23. $(b', e', 1b', e')$;
24. $(b', e', b'1, e')$.

V kroku 2 přidáme do R_N pravidla (opět jedno pravidlo již bylo přidáno dříve, takže ho nepřidáváme znovu):

25. $(b', e', b', \langle \varepsilon, e \rangle)$;
26. $(b', e', \varepsilon, \langle b, e \rangle)$;
27. $(1, \langle b, e \rangle, b, e'')$.

V kroku 3 přidáme do R_N pravidlo:

28. (b', e'', b, e'').

To znamená, že R_N nyní obsahuje všechna výše zmíněná pravidla 1 až 28 a můžeme definovat novou frontovou gramatiku v 1. normální formě $N_{12} = (V_N, T, W_N, \{f_N\}, S'e', R_N)$, kde má platit $L(Q_{12}) = L(N_{12})$.

3.2 Aplikace

Pro testování korektnosti algoritmu normalizace byla vytvořena jednoduchá konzolová aplikace. Tato aplikace umožňuje automaticky normalizovat frontové gramatiky a derivovat v nich řetězce.

3.2.1 Návrh

Tato aplikace asistuje s analýzou frontových gramatik. K tomu musí aplikace zvládat tři základní úkony:

1. Najít sekvenci derivačních konkrétního řetězce, pokud existuje, v zadané gramatice nebo případně v gramatice vzniklé po převodu zadané gramatiky do 1. normální formy;
2. Otestovat, zda daný řetězec patří do jazyka zadané gramatiky před i po převodu do 1. normální formy;
3. Hromadně testovat bod 2 pro všechny možné řetězce do určité délky pro všechny zadané gramatiky.

Díky tomu je snazší zkoumat, proč gramatika nepřijímá nějaký řetězec nebo testovat, zda gramatika opravdu přijímá předpokládaný jazyk.

Aplikace nepotřebuje grafické uživatelské rozhraní, protože se jedná primárně o nástroj sloužící k testování správnosti gramatik a jejich transformovaných protějšků. Není nutné, aby aplikace své procesy a výsledky vizuálně demonstrovala. Proto se jedná o konzolovou aplikaci.

K tomu musí aplikace zvládat algoritmicky převádět gramatiky do 1. normální formy podle algoritmu v podsekci 3.1 a současně algoritmicky derivovat řetězec. Jednotlivé úpravy algoritmu, které zavádí tato práce, jsou do aplikace přidávány jako nepovinné. To umožňuje snadno zkoumat rozdíly mezi různými verzemi algoritmu bez nutnosti úpravy zdrojového kódu.

3.2.2 Implementace

Aplikace je implementována v jazyce Swift[1]. Je to silně typovaný kompilovaný jazyk, vyvíjený společností Apple, který je nyní dostupný i na ostatních platformách. Oproti jazyku C++ nabízí mnohem jednodušší a přehlednější syntaxi, kontrolu dereference prázdného ukazatele během překladu a další prvky moderních programovacích jazyků. Oproti jazyku Python je výhodou integrovaná práce s datovými typy a jejich kontrola během překladu, protože se tak snižuje množství potenciálních chyb ve výsledném programu. Nutnost kompilace v tomto případě není problémem, protože aplikace slouží primárně jako pomůcka ke zkoumání frontových gramatik v rámci této práce. Navíc se jedná o konzolovou aplikaci, kde je kompilace pomocí příkazu `make` poměrně běžnou praxí.

Algoritmus derivace řetězců

Derivace řetězce ve frontové gramatice je implementována jednoduchým prohledáváním do hloubky. To se snaží pomocí postupné aplikace pravidel gramatiky na její konfigurace najít korektní sekvenci derivačních kroků vedoucích k derivaci daného řetězce. Protože všechny korektní konfigurace frontové gramatiky tvoří nekonečný stavový prostor, musí být v každém případě omezena maximální hloubka prohledávání, aby bylo možné rozhodnout, že gramatika nepřijímá daný řetězec. Pro aplikaci navíc není důležité najít optimální řešení, protože výstupem je pouze informace jestli gramatika daný řetězec přijímá nebo nepřijímá. Prohledávání do šířky by proto v tomto případě nebylo efektivní, protože jeho optimalita zde není výhodou, úplnost není relevantní kvůli nekonečnému stavovému prostoru a prohledávání do šířky vyžaduje výrazně větší množství paměti, což v konečném důsledku způsobuje znatelně delší dobu trvání testování více řetězců.

Aplikace nevyužívá žádných heuristik, pro zvýšení efektivity algoritmu, ale během prohledávání stavového prostoru si udržuje kolekci všech navštívených konfigurací, aby nemohlo dojít k zacyklení přes pravidla, která vkládají do fronty prázdný řetězec. Návrh vhodných heuristik a testování jejich správnosti a účinnosti sahá již nad rámec této práce.

Zpracování argumentů příkazové řádky

Pro zpracování argumentů příkazové řádky je používán balíček ArgumentParser[2]. Jedná se o balíček zaštiťovaný společností Apple, která se také stará o vývoj jazyka Swift. Balíček tak umožňuje zpracování argumentů, které je velmi dobře integrováno do syntaxe použitého jazyka.

Formát vstupní gramatiky

Pro zadávání gramatik je použit stejný formát jako v bakalářské práci [1], aby byla zachována konzistence. Jedná se tedy o formát JSON, který je snadno čitelný pro člověka

[1] https://developer.apple.com/swift/
[2] https://github.com/apple/swift-argument-parser.git

a umožňuje tak intuitivní zadávání gramatik z příkazové řádky. Současně tento formát nabízí poměrně jednoduché zpracování dat v samotné aplikaci, v jazyce Swift je tato funkcionalita dokonce součástí základní knihovny `Foundation`.

Jak je vidět na příkladu A.1, gramatika je reprezentována jedním objektem, kde jednotlivé abecedy a množiny stavů jsou reprezentovány pomocí polí. Počáteční symbol je reprezenován jako pouhý řetězec tak, aby se reprezentace více podobala formálnímu zápisu gramatiky. Současně není nutné, aby uživatel musel pokaždé psát dvě jména položek, jedno pro počáteční symbol a jedno pro počáteční stav. Množina pravidel je opět reprezentována polem. Pravidlo sice představuje z hlediska typu dat, která reprezentuje, objekt, ale ve formátu JSON je reprezentováno jako pole o pevné délce čtyř prvků. Je to opět z důvodu imitace formálního zápisu. Navíc pokud by byla pravidla reprezentována objekty, musel by uživatel napsat název každé položky v každém pravidle, což je při potřebě zápisu většího množství pravidel nežádoucí.

Aplikace nepodporuje celé rozšíření JSON5[3], které uvolňuje pravidla syntaxe formátu JSON a usnadňuje tak jeho ruční psaní. Toto rozšíření nemůže být v aplikaci podporováno, protože jeho implementace se nachází v SDK pro `macOS 12.0+`, který není kompatibilní s distribucí pro ostatní operační systémy.[4] Nicméně oproti standardu JSON aplikace navíc podporuje jednořádkové komentáře, které mohou sloužit například k popisu jazyka přijímaného gramatikou, jako je tomu v příkladu A.1. V aplikaci je tato funkce implementována pomocí jednoduchého konečného automatu tak, že je obsah řádků za značkou komentáře odstraněn před předáním dat ke zpracování.

Příklad 3.2. Mějme gramatiku $Q_{11} = (V, T, W, F, s, R)$, kde $V = \{A, a\}$, $T = \{a\}$, $W = \{s_0, f\}$, $F = \{f\}$, $s = As_0$ a $R = \{(A, s_0, aA, s_0), (A, s_0, \varepsilon, f), (a, s_0, a, s_0)\}$. Reprezentace této gramatiky ve formátu JSON je ve výpisu A.2.

```
{
    "symbols": ["A", "a"],
    "terminals": ["a"],
    "states": ["s0", "f"],
    "final_states": ["f"],
    "start": "As0",
    "productions": [
        ["A", "s0", "aA", "s0"],
        ["A", "s0", "", "f"],
        ["a", "s0", "a", "s0"]
    ]
}

// { a^n, n >= 0 }
```

Výpis 3.1: Vstupní soubor ve formátu JSON pro gramatiku Q_{11}

Vstup a výstup

Aplikace používá standardní výstupní formát pro konzolové aplikace. Návratová hodnota značí úspěch nebo neúspěch, kde `0` značí úspěch a číslo chyby značí neúspěch. Jako chybová

[3] https://json5.org

[4] https://developer.apple.com/documentation/foundation/jsondecoder/3766916-allowsjson5

hlášení jsou použity přímo výjimky, které mohou být způsobeny operacemi v kódu. Přímý výpis výjimek na standardní chybový výstup sice není tak přehledný a estetický jako ručně vytvořená chybová hlášení, ale předává uživateli informace o chybě dostatečně dobře. Navíc byla aplikace sestrojena primárně pro její použití při tvorbě této práce, takže není nutné, aby byla hlášení o chybách více přehledná.

Informace o průběhu a výsledcích jsou vypisovány na standardní výstup, což umožňuje jejich přesměrování dle potřeby uživatele. Vstupní gramatika může být zadána přes argument obsahující cestu souboru nebo přímo přes standardní vstup. To umožňuje použití aplikace v rámci unixové roury. Výjimkou je hromadné testování gramatik, kde musí být zadány cesty ke všem souborům s testovanými gramatikami a složka, do které budou uloženy textové soubory obsahující výsledky ve formátu CSV.

Formát CSV se hodí na reprezentaci výsledků testování. Pro aplikaci je snadné jej vytvořit, protože se jedná pouze o hodnoty s oddělovači v textovém souboru. Nicméně soubor je možné otevřít i v tabulkovém procesoru, což značně zvyšuje přehlednost dat a usnadňuje uživateli jejich další zpracování.

Paralelní testování

Testování většího počtu gramatik pro všechny možné řetězce je implementováno přímo v aplikaci. To skýtá několik výhod oproti tvorbě externího skriptu, který by opakovaně volal aplikaci nad různými vstupy. První výhodou je snazší přístup k abecedě terminálů dané gramatiky, protože vstupní gramatika musí být již převedena na vnitřní reprezentaci v aplikaci. Díky tomu je snazší vygenerovat všechny řetězce nad danou abecedou. Druhou výhodou je potom výrazně příjemnější rozhraní pro práci s paralelismem.

Aplikace nepoužívá paralelní algoritmus k derivaci jednoho řetězce, protože to by vyžadovalo režii pro řízení jednotlivých vláken a je otázkou, jestli by vůbec došlo ke zrychlení výpočtu. Místo toho aplikace využívá faktu, že derivace řetězce v gramatice je oddělená operace, která, mimo zápisu výsledků, nevyžaduje interakci se zbytkem aplikace. Proto je pro každou derivaci řetězce vytvořeno jedno vlákno. To nezpůsobí urychlení výpočtu z pohledu doby trvání derivace jednoho řetězce, ale umožňuje to aplikaci využít při testování celý dostupný výkon počítače. Současně při testování jednodušších gramatik je celkový čas výpočtu velmi krátký, takže zvýšená režie při zápisu není problém.

3.3 Identifikace chyb

Gramatiky na přiloženém médiu jsou rozděleny do skupin podle třídy jejich jazyků, toto rozdělení ukazují tabulky 3.1, 3.2 a 3.3.

Gramatika	Jazyk
Q_1	$\{\varepsilon\}$
Q_2	$\{\ aa\}$
Q_3	$\{\ aa\}$
Q_4	$\{\ aaa\}$
Q_5	$\{\ aa\}$
Q_6	$\{\ \varepsilon, a, aa\}$

Tabulka 3.1: Gramatiky přijímající konečné jazyky

Gramatika	Jazyk
Q_{11}	$\{ a^n \mid n \geq 0\}$
Q_{12}	$\{ a^n b^n \mid n \geq 0\}$
Q_{13}	$\{ w \mid \#_a(w) = \#_b(w)\}$
Q_{14}	$\{ ww^r\}$

Tabulka 3.2: Gramatiky přijímající regulární nebo bezkontextové jazyky. Kde w značí libovolný řetězec složený z terminálů příslušné gramatiky, $\#_a(w)$ značí počet symbolů a v řetězci w a w^r značí reverzaci řetězce w.

Gramatika	Jazyk
Q_{21}	$\{ a^{2^n} \mid n \geq 0\}$
Q_{22}	$\{ a^n b^n c^n \mid n \geq 0\}$
Q_{23}	$\{ ww\}$

Tabulka 3.3: Gramatiky přijímající kontextové jazyky, kde w značí libovolný řetězec složený z terminálů příslušné gramatiky.

Gramatiky Q_1, ..., Q_6, Q_{11}, Q_{12} a Q_{21} byly testovány pro všechny řetězce do délky 6 a maximální délky sekvence derivačních kroků 20. Všechny relevantní řetězce do délky n lze pro každou gramatiku vytvořit na základě množiny terminálů T následovně:

$$\bigcup_{i=0}^{n} T^i.$$

Ostatní gramatiky nebyly testovány hromadně, protože jsou příliš složité, pro hromadnou analýzu. Nicméně na nich byla provedena kontrola ručně, pouze pro některé řetězce.

Testování ukazuje, že některé gramatiky po algoritmické normalizaci přijímají méně řetězců než původní gramatiky, nikdy obráceně. Současně je vidět, že gramatiky, jako například N_2, N_3 nebo N_{12}, nepřijímají řetězce, které bylo možné před normalizací přijmout jediným derivačním krokem. Navíc ale některé gramatiky, jako například N_6 nebo N_{12}, nepřijímají i složitější řetězce. Nicméně z testování je zřejmé, že se problém týká výhradně kratších řetězců.

3.3.1 Derivace příliš krátkých řetězců

Mějme frontové gramatiky Q_{12} a N_{12} z příkladu 3.1. Je zřejmé, že $L(Q_{12}) = \{a^n b^n \mid n \geq 0\}$ a současně z algoritmu normalizace plyne $L(Q_{12}) = L(N_{12})$. V gramatice Q_{12} lze řetězec ab derivovat následovně:

$$Se \Rightarrow bSae \Rightarrow Sabe \Rightarrow abf.$$

Následně se pokusíme derivovat větu ab v gramatice N_{12}, která byla v příkladu 3.1 převedena podle alogritmu v podsekci 3.1. V následujícím textu je odkazováno na pravidla z tohoto příkladu pomocí jejich čísel.

K derivaci budeme přistupovat od konce. Víme, že na konci úspěšné derivace se musí gramatika nacházet v koncovém stavu f_N a současně musí ve frontě zbývat pouze symboly a a b. Začneme proto větnou formou:

$$abf_N.$$

Nyní se pokusíme najít všechny derivační kroky, které by mohly na danou větnou formu vést. Hledáme tedy všechna pravidla, jejichž cílový stav je f_N a zapisovaný řetězec je příponou řetězce ab, tedy ε, b nebo ab. Takovéto pravidlo existuje pouze jedno, a tak zpětně provedeme derivační krok:

$$S'abe'' \Rightarrow abf_N\,[15]\,.$$

Pro nově vzniklou větnou formu se analogicky snažíme najít příslušné derivační kroky. Existují dvě pravidla, jejichž cílový stav je e'' a zapisovaný řetězec je příponou řetězce $S'ab$, a sice pravidla 27 a 28.

Nejprve tedy zkusíme aplikovat pravidlo 27 a provedeme příslušný derivační krok:

$$1S'a\langle b, e\rangle \Rightarrow S'abe''\,[27] \Rightarrow abf_N.$$

Následně opět existuje pouze jediné pravidlo s cílovým stavem $\langle b, e\rangle$ a zapisovaným řetězcem, který je příponou řetězce $1S'a$. Provedeme tedy zpětně další derivační krok:

$$b'S'ae' \Rightarrow 1S'a\langle b, e\rangle\,[26] \Rightarrow S'abe'' \Rightarrow abf_N.$$

V gramatice N_{12} ale neexistuje žádné pravidlo s cílovým stavem e' a zapisovaným řetězcem, který je příponou řetězce $b'S'a$. Z toho vyplývá, že neexistuje sekvence derivačních kroků $S'e' \Rightarrow^* b'S'ae'$ a tímto způsobem nelze řetězec ab derivovat.

Pokud provedeme stejný postup i pro pravidlo 28, dostaneme tyto tři sekvence derivačních kroků:

$$S'1e' \Rightarrow 1b'S'\langle a, e\rangle\,[8] \Rightarrow b'S'ae''\,[9] \Rightarrow S'abe''\,[28] \Rightarrow abf_N,$$

$$a'1b'S'e' \Rightarrow 1b'S'\langle a, e\rangle\,[20] \Rightarrow b'S'ae''\,[9] \Rightarrow S'abe''\,[28] \Rightarrow abf_N,$$

$$1a'b'S'\langle \varepsilon, e\rangle \Rightarrow a'b'S'e''\,[7] \Rightarrow b'S'ae''\,[21] \Rightarrow S'abe''\,[28] \Rightarrow abf_N.$$

Žádná z větných forem $S'1e'$, $a'1b'S'e'$ a $1a'b'S'\langle \varepsilon, e\rangle$ ale opět není dosažitelnou sekvencí derivačních kroků z počáteční větné formy $S'e'$, protože v gramatice N_{12} neexistuje pravidlo s příslušnou kombinací cílového stavu a zapisovaného řetězce.

Z toho vyplývá, že neexistuje sekvence derivačních kroků $S'e' \Rightarrow^* abf_N$ v gramatice Q a tato gramatika tedy nepřijímá řetězec ab. Tím pádem $L\,(Q_{12}) \neq L\,(N_{12})$ což znamená, že algoritmus normalizace v podsekci 3.1 je chybný.

Návrh korekce

Gramatika N_{12} přijímá řetězec $aabb$, ale nepřijímá řetězec ab, přestože $ab \in L\,(Q_{12})$. Nejprve se podíváme na sekvenci derivačních kroků pro řetězec $aabb$:

$$S'e' \Rightarrow_{\mathrm{I}} b'S'1a'e'\,[4] \Rightarrow_{\mathrm{II}} S'1a'b'e'\,[22] \Rightarrow_{\mathrm{III}} 1a'b'b'S'\langle a, e\rangle\,[8] \Rightarrow_{\mathrm{IV}} a'b'b'S'ae''\,[9]$$
$$\Rightarrow_{\mathrm{V}} b'b'S'aae''\,[21] \Rightarrow_{\mathrm{VI}} b'S'aabe''\,[28] \Rightarrow_{\mathrm{VII}} S'aabbe''\,[28] \Rightarrow_{\mathrm{VIII}} aabbf_N\,[15].$$

Povšimněme si derivačního kroku I, kde gramatika vloží dělicí symbol 1 do fronty. Za ním potom následuje krok III, kde dochází k přechodu do složeného stavu $\langle a, e\rangle$. V obou těchto krocích je použito pravidlo, které vzniklo převodem z pravidla $(S, e, bSa, e) \in R_Q$ (viz příklad 3.1), a tím pádem kroky I a III v N_{12} odpovídají dvojité aplikaci pravidla $(S, e, bSa, e) \in R_Q$ v původní gramatice Q_{12}.

Problém však nastává, pokud by gramatika měla derivovat řetězec ab, protože v gramatice N_{12} existují pouze 3 tvary pravidel, které mohou do fronty vkládat terminály:

1. $(1, \langle y, p\rangle, y, q) \in R_N$, kde $y \in T^*$, $\langle y, p\rangle \in U$ a $q \in W''_Q$;
2. $(a, p, y, q) \in R_N$, kde $a \in V'_Q$, $y \in T^*$ a $p, q \in W''_Q$;
3. $(a, p, y, f_N) \in R_N$, kde $a \in V'_Q$, $y \in T^*$ a $p \in W''_Q$.

To znamená, že pro vkládání terminálů je nutné, aby gramatika byla ve stavu z množiny W''_Q, do kterého lze přejít pouze, pokud je dělicí symbol 1 na začátku fronty a současně je gramatika ve stavu z množiny U. V algoritmicky normalizované gramatice k tomu slouží pravidla ve tvaru:

1. $(a, p, x1y, q) \in R_N$, kde $a \in V'_Q$, $x, y \in (V'_Q)^*$ a $p, q \in W'_Q$ a
2. $(a, p, x, \langle y, q\rangle) \in R_N$, kde $a \in V'_Q$, $x \in (V'_Q)^*$, $p \in W'_Q$ a $\langle y, q\rangle \in U$.

Z toho plyne, že gramatika potřebuje dva derivační kroky k tomu, aby mohla začít vkládat terminály na konec fronty, a současně tyto dva derivační kroky musí proběhnout v pořadí: přidání dělicího symbolu 1 a až potom přechod do složeného stavu z množiny U. Ze složeného stavu se již nelze dostat do stavu z množiny W'_Q, aby bylo možné vložit dělicí symbol.

Současně z pravidel 14 a 15 plyne, že pokud se gramatika nachází ve stavu z množiny W''_Q a na začátku fronty se ocitne symbol S', gramatika přejde do koncového stavu. V gramatice N_{12} totiž neexistuje pravidlo s počátečním stavem f''.

Během derivace řetězce ab v gramatice N_{12} tedy musí dojít k aplikaci pravidla 8, protože jinak ve frontě zůstane symbol a' za symbolem S' a to, na základě tvrzení v předchozím odstavci, nemůže vést na korektní derivaci. Před aplikací pravidla 8 je ale nutné vložit do fronty dělicí symbol 1. To však není možné, protože jestli před pravidlem 8 mají být aplikována nějaká pravidla, musí mezi nimi být nejméně jedno pravidlo aplikované na počáteční konfiguraci $S'e'$. Všechna taková pravidla[5] byla ale odvozena od pravidla $(S, e, bSa, e) \in R_Q$ stejně jako pravidlo 8 a během derivace nemůže dojít k aplikaci více než jednoho pravidla odvozeného od $(S, e, bSa, e) \in R_Q$, protože by došlo k vložení příliš mnoha terminálů do fronty.

Aby bylo možné derivovat řetězec ab v gramatice N_{12}, musela by množina pravidel R_N navíc obsahovat pravidlo, které by umožňovalo vložení dělicího symbolu 1 a přechod do složeného stavu z množiny U současně během jednoho kroku. Derivace řetězce ab by potom vypadala takto:

$$S'e' \Rightarrow 1b'S'\langle a, e\rangle \left[(S', e', 1b'S', \langle a, e\rangle)\right] \Rightarrow b'S'ae'' \,[9] \Rightarrow S'abe'' \,[28] \Rightarrow abf_N \,[15]\,.$$

3.3.2 Sekvence derivačních kroků o délce jedna

I po úpravě algoritmu naznačené v předchozí podsekci 3.3.1, stále neplatí $L(Q_{12}) = L(N_{12})$ pro gramatiky z příkladu 3.1. Rozdíl je v řetězci ε, gramatika Q_{12} triviálně přijímá prázdný řetězec pomocí pravidla $(S, e, \varepsilon, f) \in R_Q$ v jediném derivačním kroku:

$$Se \Rightarrow f\,[(S, e, \varepsilon, f)]\,.$$

Nicméně gramatika N_{12} není schopná derivovat prázdný řetězec. K úspěšné derivaci by měla vést pravidla 10 až 15, protože byla vygenerována právě na základě pravidla $(S, e, \varepsilon, f) \in R_Q$. K nim v upraveném algoritmu přibude na základě předchozí podsekce

[5] Mimo pravidla vedoucí do stavů f' a f'', ve kterých nelze aplikovat žádné pravidlo.

3.3.1 pravidlo $(S', e', 1, \langle \varepsilon, f \rangle) \in R_N$. Z těchto pravidel je jasné, že k úspěšné derivaci nemůže dojít, protože pravidlo 15, které jediné umožňuje přechod do koncového stavu vyžaduje, aby se před jeho aplikací gramatika nacházela v konfigutaci $S'e''$. V gramatice ale neexistují pravidla, která by umožnila přechod do takové konfigurace.

K řešení tohoto problému se nabízejí dva přístupy:

1. Vytvoření zcela nového pravidla v normalizované gramatice, které umožní přímý přechod v jednom derivačním kroku z počátečního stavu normalizované gramatiky do koncového stavu normalizované gramatiky:

   $$S'e' \Rightarrow f_N \left[(S', e', \varepsilon, f_N) \right].$$

   Toto pravidlo by bylo vytvořeno pro všechna pravidla původní gramatiky, která také umožňují přechod z počáteční konfigurace původní gramatiky do kteréhokoliv z koncových stavů a zapisují pouze terminály;

2. Vytvoření nového pravidla v normalizované gramatice, které naváže na nové pravidlo z podsekce 3.3.1 a umožní následný přechod do koncového stavu normalizované gramatiky v jediném derivačním kroku:

   $$S'e' \Rightarrow 1\langle \varepsilon, f \rangle \left[(S', e', 1, \langle \varepsilon, f \rangle) \right] \Rightarrow f_N \left[(1, \langle \varepsilon, f \rangle, \varepsilon, f_N) \right].$$

   Toto pravidlo by bylo vytvořeno pouze, pokud v původní gramatice existuje pravidlo ve tvaru $(N, e, y, f) \in R_Q$, kde $N \in V_Q$, $e \in (W_Q - F_Q)$, $y \in T^*$ a $f \in F_Q$. Zde není nutné omezovat tvorbu pravidla pouze na případy, kdy Ne tvoří počáteční konfiguraci.

Obě tyto možnosti jsou ekvivalentní z hlediska jazyka přijímaného normalizovanou gramatikou, protože z koncového stavu normalizované gramatiky f_N již nelze přejít do žádného jiného stavu (viz podsekci 3.1 a příklad 3.1), takže nelze vložit do fronty žádné další symboly. To znamená, že z definice jazyka přijímaného frontovou gramatikou v podsekci 2.1, musí v obecném derivačním kroku $1x\langle y, f \rangle \Rightarrow xyf_N$ platit, že $xy \in T^*$, kde T je množina terminálů normalizované gramatiky. Tento krok tedy lze provést pouze z konfigurací ve tvaru $1x\langle y, f \rangle$, kde $x = \varepsilon$, protože před přechodem do složeného stavu nelze generovat žádné terminály.

Během kroku 1 v algoritmu normalizace v podsekci 3.1 jsou v normalizované gramatice vytvořena taková pravidla, že pro každou posloupnost derivačních kroků v původní gramatice Q:

$$Se \Rightarrow^n wf,$$

kde Se je počáteční konfigurace, $w \in L(Q)$ a f je koncový stav, platí, že v normalizované gramatice N je možné provést stejnou sekvenci derivačních kroků:

$$S'e' \Rightarrow^n y1xf',$$

kde $yx = \delta(w)$ (pro význam zobrazení δ viz podsekci 3.1). To znamená, že pokud je normalizovaná gramatika N schopná dosáhnout konfigurace $1\langle y, f \rangle$, ve které je možné aplikovat nové pravidlo vytvořené v přístupu 2, pak je i původní gramatika Q schopná derivovat řetězec y.

Oprava pomocí možnosti 2 se hodí více, protože lépe zapadá do algoritmu normalizace, který je postavený na myšlence, že normalizovaná gramatika nejprve vygeneruje dělicí symbol a přejde do složeného stavu a až poté generuje terminály. Naproti tomu možnost 1 celý tento postup obchází a umožňuje generování terminálů rovnou, i když jen ve speciálním případě.

3.4 Oprava chyb v algoritmu

Na základě sekce 3.3 byl upraven algoritmus normalizace. V kroku 2 jsou navíc do R_N přidávána pravidla ve tvaru $(a, p, 1x, \langle y, q\rangle)$, kde $a \in V'_Q$, $p \in W'_Q$, $x \in (V'_Q)^*$ a $\langle y, q\rangle \in U$. Dělicí symbol 1 je vždy vložen na začátek řetězce vkládaného do fronty, protože všechna pravidla začínající ve složeném stavu z množiny U vyžadují na začátku fronty neterminál 1 a tudíž by jiná pozice dělicího symbolu nemohla vést na korektní derivaci. Dále byl upraven krok 4, kde jsou přidána pravidla ve tvaru $(1, \langle y, f\rangle, y, f_N)$, aby bylo možné přejít do koncového stavu gramatiky i v případě, kdy je ve frontě pouze dělicí symbol 1.

3.4.1 Korektní algoritmus normalizace

Mějme frontovou gramatiku $Q = (V_Q, T, W_Q, F_Q, s_Q, R_Q)$, kde $s_Q = a_0 q_0$. Dále definujme množiny $W'_Q = \{q' : q \in W_Q\}$, $W''_Q = \{q'' : q \in W_Q\}$, $V'_Q = \{a' : a \in V_Q\}$ a množinu složených stavů $U = \{\langle y, q\rangle : y \in T^* \wedge (a, p, xy, q) \in R_Q\}$. Na závěr definujeme bijekci α z W_Q do W'_Q jako $\alpha(q) = q'$, bijekci β z W_Q do W''_Q jako $\beta(q) = q''$. Analogicky definujeme bijekci δ z V_Q do V'_Q jako $\delta(a) = a'$, kterou navíc standardně rozšíříme na řetězce z V_Q^* do $(V'_Q)^*$. Je zřejmé, že bijekce α je na řetězcích rovněž úplná.

Zavedeme nový dělicí symbol 1, kde $1 \notin V'_Q \cup T$ a nový koncový stav f_N, kde $f_N \notin W'_Q \cup W''_Q \cup U$. S jejich pomocí sestrojíme frontovou gramatiku v 1. normální formě $N = (V_N, T, W_N, F_N, s_N, R_N)$, kde $V_N = V'_Q \cup \{1\} \cup T$, $W_N = W'_Q \cup W''_Q \cup \{f_N\} \cup U$, $F_N = \{f_N\}$ a $s_N = \delta(s_0)\alpha(q_0)$. Množina pravidel R_N je sestrojena v následujících krocích aplikovaných na každé pravidlo z R_Q:

1. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x, y \in V_Q^*$ a $q \in W_Q$, pak $(\delta(a), \alpha(p), \delta(x)\delta(y), \alpha(q))$ a $(\delta(a), \alpha(p), \delta(x)1\delta(y), \alpha(q))$ patří do R_N;

2. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in V_Q^*$, $y \in T^*$, $q \in W_Q$ a $\langle y, q\rangle \in U$, pak $(\delta(a), \alpha(p), \delta(x), \langle y, q\rangle)$, $(\delta(a), \alpha(p), 1\delta(x), \langle y, q\rangle)$ a $(1, \langle y, q\rangle, y, \beta(q))$ patří do R_N;

3. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in W_Q$, pak $(\delta(a), \beta(p), x, \beta(q))$ patří do R_N;

4. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in F_Q$, pak $(\delta(a), \beta(p), x, f_N)$ a $(1, \langle y, q\rangle, y, f_N)$ patří do R_N.

3.4.2 Redukce počtu pravidel

Počet pravidel v algoritmicky normalizované gramatice lze snížit výměnou za navýšení počtu kroků ve formálním zápisu algoritmu. Redukce je založena na faktu z definice frontové gramatiky (viz sekci 2.1). Platí totiž, že pravidla nemohou být aplikována v koncových stavech. Současně s tím se v algoritmu normalizace (viz podsekci 3.4.1) nevytvářejí žádná pravidla, která by měla výchozí stav odvozený od koncového stavu pravidla v původní gramatice. Tedy platí:

$$((a, p, w, q) \in R_Q \wedge (b, q, x, r) \notin R_Q) \implies ((c, \alpha(q), y, s) \notin R_N \wedge (d, \beta(q), z, t) \notin R_N),$$

kde $a, b \in V_Q$, $c, d \in V_N$, $w, x \in V_Q^*$, $y, z \in V_N^*$, $p, q, r \in W_Q$, $s, t \in W_N$ a zobrazení α a β odpovídají definici v algoritmu normalizace.

Z toho plyne, že v algoritmicky normalizované gramatice neexistují pravidla ve tvaru $(a, \alpha(f), x, q) \in R_N$ ani $(a, \beta(f), x, q) \in R_N$, kde $f \in F_Q$. A tedy je v algoritmu zbytečné vytvářet pravidla ve tvaru $(b, p, y, \alpha(f))$ a $(b, p, y, \beta(f))$, kde $f \in R_Q$, protože $\alpha(f), \beta(f) \notin F_N$ a nebude vytvořeno žádné pravidlo, které by bylo možné ve vzniklé konfiguraci aplikovat. Pro tvary pravidel v tomto odstavci platí, že $a, b \in V_N$, $x, y \in V_N^*$ a $p, q \in W_N$.

Současně s tím je výhodné pravidla, přidaná na základě podsekce 3.3.2, omezit pouze na případy, kdy je pravidlo v původní gramatice Q aplikovatelné v počáteční konfiguraci gramatiky Q. Jazyk normalizované gramatiky N zůstane nezměněn, jak je naznačeno právě ve zmíněné podsekci.

Po aplikaci výše zmíněných úprav pro snížení počtu pravidel pravidel, zůstane první část algoritmu stejná jako v podsekci 3.4.1, ale změní se kroky pro tvorbu množiny pravidel R_N na následující:

1. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x, y \in V_Q^*$ a $q \in W_Q - F_Q$, pak $(\delta(a), \alpha(p), \delta(x)\delta(y), \alpha(q))$ a $(\delta(a), \alpha(p), \delta(x)1\delta(y), \alpha(q))$ patří do R_N;

2. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in V_Q^*$, $y \in T^*$, $q \in W_Q$ a $\langle y, q\rangle \in U$, pak $(\delta(a), \alpha(p), \delta(x), \langle y, q\rangle)$ a $(\delta(a), \alpha(p), 1\delta(x), \langle y, q\rangle)$ patří do R_N;

3. Pokud $(a, p, xy, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in V_Q^*$, $y \in T^*$, $q \in W_Q - F_Q$ a $\langle y, q\rangle \in U$, pak $(1, \langle y, q\rangle, y, \beta(q))$ patří do R_N;

4. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in W_Q - FQ$, pak $(\delta(a), \beta(p), x, \beta(q))$ patří do R_N;

5. Pokud $(a, p, x, q) \in R_Q$, kde $a \in V_Q$, $p \in W_Q - F_Q$, $x \in T^*$ a $q \in F_Q$, pak $(\delta(a), \beta(p), x, f_N)$ patří do R_N;

6. Pokud $(a, p, x, q) \in R_Q$, kde $ap = s_Q$, $x \in T^*$ a $q \in F_Q$, pak $(1, \langle y, q\rangle, y, f_N)$ patří do R_N.

Výsledný rozdíl v počtu pravidel oproti algoritmu v podsekci 3.4.1 záleží na množství pravidel v původní gramatice, která vedou do koncového stavu, a jejich složitosti. Nicméně vzhledem k celkovému dramatickému nárůstu počtu pravidel v algoritmicky normalizované gramatice, a faktu, že simulace derivace řetězce ve frontové gramatice vede na exponenciální algoritmus, je i sebemenší snížení počtu pravidel, a tím pádem stavů, výhodné.

3.4.3 Testování

Závěrečné testování korektnosti algoritmu bylo prováděno stejnou metodikou jako před implementací opraveného algoritmu. To ukázalo, že navržené změny v podsekci 3.4.1 opravdu vedou na korektní algoritmus a nezpůsobují žádné vedlejší účinky nebo nové nesrovnalosti mezi přijímanými jazyky. Rovněž byla zvlášť testována redukce pravidel z podsekce 3.4.2, která rovněž nezpůsobuje žádné vedlejší účinky.

Výsledky testování původního algoritmu jsou na přiloženém médiu označeny prefixem `original`, výsledky testování opraveného algoritmu jsou označeny prefixem `fixed` a výsledky testování redukce pravidel jsou označeny prefixem `reduced`. Sufix `m`n značí, že bylo prováděno testování pro maximální hloubku prohledávání n.

Kapitola 4

Transformace na rozptýlené gramatiky

Tato kapitola se zaobírá návrhem nové transformace frontových gramatik. Navržená transformace převádí frontové gramatiky na gramatiky s rozptýleným kontextem a jedním kontextovým pravidlem. Taková transformace již existuje v článku [6], nicméně tato práce se navíc soustředí na minimalizaci popisné složitosti výsledného modelu. Tedy díky jednomu kontextovému pravidlu jsou minimalizovány kontextové závislosti, ale navržená transformace by měla současně minimalizovat i počet neterminálů ve výsledné gramatice.

V době odevzdání práce rozpracovaný článek [5] pracuje s hypotézou, že každou frontovou gramatiku v 2. normální formě lze beze ztráty na obecnosti transformovat na gramatiku s rozptýleným kontextem a jedním kontextovým pravidlem, která obsahuje pouze 6 neterminálů. Transformace navržená v rozpracovaném článku byla upravena do podoby v následující sekci.

4.1 Navržená transformace

Pro potřeby transformace je nutné zadefinovat kódování neterminálů a stavů frontové gramatiky na množinu neterminálů rozptýlené gramatiky $D = \{0, 1, 2, 3\}$. Toto kódování je shodné s kódováním ve článku [6]. Samotné kódování popisují definice 4.1 a 4.2, zatímco definice 4.3, 4.4 a 4.5 zavádí pomocné konstrukce pro potřeby algoritmu transformace.

4.1.1 Kódování neterminálů a stavů frontové gramatiky

Mějme dvě obecné množiny A a B, na základě kterých zavedeme konstantu $c \geq |A| \cdot |B|$. Dále definujme injekci $o : A \times B \to \{1, \ldots, c\}$, která každé dvojici prvků z množin A a B přiřadí unikátní index.

Definice 4.1. Na základě konstanty c lze definovat množiny kódů $X, Y \subseteq D^*$:

$$X = \bigcup_{i=1}^{c} \{(103)^i 1 (10)^{(c+1)-i}\},$$
$$Y = \bigcup_{i=1}^{c} \{(301)^{(c+1)-i} 0301 (01)^i\}.$$

Definice 4.2. Definujme injekce ι a κ, které kódují množiny A a B do množiny D^*:

$$\iota : A \times B \to X$$
$$\iota(a,b) = (103)^k 1 (10)^{(c+1)-k}, \text{ kde } k = o(a,b)$$

$$\kappa : A \times B \to Y$$
$$\kappa(a,b) = (301)^{(c+1)-k} 0301 (01)^k, \text{ kde } k = o(a,b)$$

Konstrukce jednoduchého kódování je popsána v příkladu 4.1.

Příklad 4.1. Mějme $A = \{C, D\}$, $B = \{s\}$ a tedy $c = 2$. Potom injekci o lze definovat jako $o(C, s) = 1$ a $o(D, s) = 2$. Na jejím základě potom definujme injekce ι a κ:

$$\iota(C,s) = 10311010,$$
$$\iota(D,s) = 103103110,$$
$$\kappa(C,s) = 301301030101,$$
$$\kappa(D,s) = 30103010101.$$

Definice 4.3. Dva kódy si *odpovídají*, pokud byly vytvořeny na základě stejné dvojice prvků $(a, b) \in A \times B$. To znamená, že pro množiny A a B z příkladu 4.1 si například kódy $\iota(C, s)$ a $\kappa(C, s)$ odpovídají.

Definice 4.4. Na základě množin A a B a bijekcí ι a κ definujme substituce $\nu : A \to 2^X$ a $\mu : B \to 2^Y$:

$$\nu(a) = \{\iota(a,b) \mid b \in B\}$$
$$\mu(b) = \{\kappa(a,b) \mid a \in A\}$$

Obě substituce mohou být standardním způsobem rozšířeny na řetězce jako $A^* \to (2^X)^*$, popřípadě $B^* \to (2^Y)^*$.

Definice 4.5. Definujme funkci δ na množině $\{y \in \nu(a) \mid a \in A\}^*$, která umožní duplikaci kódů v řetězci:

1. $\delta(\varepsilon) = \varepsilon$;
2. $\delta(x_1 x_2 x_3 \dots x_n) = x_1^2 x_2^2 x_3^2 \dots x_n^2$, kde $n \geq 1$, $x_i \in \nu(a_i)$ a $a_i \in A$.

4.1.2 Algoritmus transformace

Mějme levě rozšířenou frontovou gramatiku $Q = (V, T, W, F, s, R)$ v 2. normální formě, pro kterou platí $\forall (a, p, y, q) \in R : y \neq \varepsilon$.[1] Tuto gramatiku transformujeme v následujících krocích:

1. Mějme množinu neterminálů $N = \{4, 5\} \cup D$, kde $D = \{0, 1, 2, 3\}$;

[1] Toto omezení plyne z následného důkazu korektnosti a je podrobně popsáno později v sekci 4.4.1.

2. Na základě množin $A = V - T$ a $B = W - F$ zaveďme ι, κ, ν, μ a δ podle sekce 4.1.1;

3. Inicializujme množinu pravidel M na $\emptyset$ a sestrojme M pomocí následujících kroků:

   (i) Pokud $s = a_0 q_0$, kde $a_0 \in V - T$ a $q_0 \in W - F$, pak $4 \to 1\delta(u)4v1$, kde $u = \iota(a_0, q_0)$ a $v = \kappa(a_0, q_0)$, patří do M;

   (ii) Pokud $(a, p, y, q) \in R$, kde $a \in V - T$, $p, q \in W - F$ a $y \in (V - T)^+$, pak $4 \to \delta(u)4vw$, kde $w = \kappa(a, p)$, pro všechna $u \in \nu(y)$ a $v \in \mu(q)$ patří do M;

   (iii) Pravidlo $4 \to 205$ patří do M;

   (iv) Pokud $(a, p, y, q) \in R$, kde $a \in V - T$, $p, q \in W - F$ a $y \in T^+$, pak $5 \to y5vw$, kde $w = \kappa(a, p)$, pro všechna $v \in \mu(q)$ patří do M;

   (v) Pokud $(a, p, y, q) \in R$, kde $a \in V - T$, $p \in W - F$, $y \in T^*$ a $q \in F$, pak $5 \to y302w$, kde $w = \kappa(a, p)$, patří do M.

4. Nastavme $O = \{(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2), 2 \to \varepsilon\}$;

5. Výsledná gramatika $G = (N, T, M \cup O, 4)$.

4.2 Aplikace

Pro efektivní analýzu algoritmu transformace ze sekce 4.1.2 je výhodné vytvořit nástroj. Tento nástroj je součástí aplikace ze sekce 3.2, protože takto je možné sdílet části kódu mezi jednotlivými případy užití.

4.2.1 Návrh

Aplikace je navržena jako pomocný nástroj k derivaci řetězců v rozptýlené gramatice. Je tedy nutné aby dokázala:

1. Algoritmicky transformovat zadanou frontovou gramatiku v 2. normální formě na gramatiku s rozptýleným kontextem;

2. Asistovat uživateli při derivaci řetězců.

Aplikace musí umět přepínat mezi jednotlivými verzemi algoritmu, které byly v během procesu návrhu uvažovány. Tak bude možné se případně k některé z předchozích verzí pohodlně vrátit, pokud by se ukázalo, že ta současná není korektní.

Asistovaná derivace uživateli nabízí pravidla, která lze aplikovat v dané větné formě. Takto se uživatel může přesouvat mezi jednotlivými větnými formami a derivovat tak libovolný řetězec. Současně s tím má každý dostupný derivační krok přiřazený očekávaný počet kroků potřebný k úspěšné derivaci. Pravidla, která nemohou vést k úspěšné derivaci mají počet kroků potřebný k úspěšné derivaci nastaven jako ∞.

Pro přehlednost aplikace neukazuje celé kódy, ale pouze zástupné značky. Až v případě, že je z kódu během derivace odstraněn nějaký neterminál, je celý kód vypsán do větné formy.

4.2.2 Implementace

Tato část aplikace je tedy součástí nástroje popsaného výše v sekci 3.2. Proto platí, že formát vstupní gramatiky a chybové výstupy jsou shodné s tím, co již bylo popsáno výše.

Volba mezi jednotlivými verzemi algoritmu je implementováno pomocí přepínačů příkazové řádky, protože není žádoucí měnit verzi algoritmu za běhu. Asistované derivace je implementována jako prohledávání do hloubky se zpětným navracením, kde uživatel řídí jednotlivé kroky prohledávání z příkazové řádky.

Heuristika, která odhaduje počet kroků potřebných k úspěšné derivaci, se řídí počty odpovídajících si kódů nalevo ve větné formě. Pokud jsou ve větné formě naproti sobě dva neodpovídající si kódy, pak je počet kroků ∞. Jinak je počet kroků vypočítán následujícím vzorcem:

$$\text{počet kroků} = \frac{\text{počet kódů vlevo} - \text{počet kódů vpravo}}{2} + \text{počet zbývajících fází.}$$

Fáze v tomto vzorci znamená nutný derivační krok. Tedy například aplikace pravidla $4 \to 205$ snižuje počet zbývajících fázi o 1, protože aby byla derivace úspěšná, musí být proveden krok z neterminálu 4 do neterminálu 5. Rozdíl v počtu kódů je dělen dvěma, protože běžné pravidlo vkládá na pravou stranu kódy po dvojicích.

4.3 Důkaz korektnosti

Dále následuje důkaz korektnosti algoritmu transformace výše v sekci 4.1.2. Nejdříve je ale nutné popsat základní princip fungování algoritmu.

Rozptýlená gramatika po transformaci nejprve v 1. fázi derivace pomocí bezkontextových pravidel z množiny M nedeterministicky simuluje derivaci řetězce ve frontové gramatice. To je prováděno vkládáním kódů na obě strany větné formy okolo neterminálu 4, později okolo neterminálu 5. Následně ve 2. fázi pomocí kontextového pravidla z množiny O ověří, zda byla 1. fáze korektní. Ověření je založeno na porovnáváni vložených kódů. A sice musí platit, že vložené kódy na levé a pravé straně větné formy si odpovídají (viz definice 4.3). Úspěšná derivace řetězce je demonstrována na příkladu 4.2.

Pro přehlednost je dále zavedena konvence:

$$\iota(a, p) = \lceil a, p \rceil$$
$$\kappa(a, p) = \lfloor a, p \rfloor$$

Příklad 4.2. Mějme levě rozšířenou frontovou gramatiku $Q = (V, T, W, F, s, R)$, kde $V = \{S, X, a, b\}$, $T = \{a, b\}$, $W = \{s, p, q, f\}$, $F = \{f\}$ a $s = Ss$. Množina pravidel R obsahuje pravidla:

1. (S, s, XS, p);
2. (X, p, aa, q);
3. (S, q, bb, f).

Jazyk gramatiky je zřejmě $L(Q) = \{aabb\}$. Jednotlivá pravidla gramatiky jsou očíslována tak, aby se na ně bylo možné v sekvenci derivačních kroků odkazovat a zvýšila se tak přehlednost zápisu.

Gramatiku Q dále na základě algoritmu v sekci 4.1.2 transformujme na gramatiku $G = (N, T, P, 4)$, kde $N = \{0, 1, 2, 3, 4, 5\}$. Množina pravidel P bude mimo jiné obsahovat pravidla:

1. Pravidlo $4 \to 1\lceil S, s\rceil\lceil S, s\rceil 4\lfloor S, s\rfloor 1$ vytvořené v kroku (i) na základě počáteční větné formy Ss;
2. Pravidlo $4 \to \lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil 4\lfloor X, p\rfloor\lfloor S, s\rfloor$ vytvořené v kroku (ii) na základě pravidla $(S, s, XS, p) \in R$;
3. Pravidlo $4 \to 205$;
4. Pravidlo $5 \to aa5\lfloor S, q\rfloor\lfloor X, p\rfloor$ vytvořené v kroku (iv) na základě pravidla $(X, p, aa, q) \in R$;
5. Pravidlo $5 \to bb302\lfloor S, q\rfloor$ vytvořené v kroku (v) na základě pravidla $(S, q, bb, f) \in R$;

Potom úspěšná derivace řetězce $aabb$ v gramatice G bude probíhat následovně ve dvou fázích. Nejprve je v 1. fázi pomocí bezkontextových pravidel simulována derivace řetězce ve frontové gramatice Q:

$$
\begin{array}{lcr}
 & 4 & \\
\Rightarrow 1\lceil S, s\rceil\lceil S, s\rceil & 4 & \lfloor S, s\rfloor 1 \\
\Rightarrow 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil & 4 & \lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil 20 & 5 & \lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil 20 & aa5 & \lfloor S, q\rfloor\lfloor X, p\rfloor\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil 20 & aabb3 & 02\lfloor S, q\rfloor\lfloor S, q\rfloor\lfloor X, p\rfloor\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1
\end{array}
$$

Následně, pokud byla simulace korektní, lze ve 2. fázi pomocí kontextového pravidla eliminovat odpovídající si kódy:

$$
\begin{array}{lll}
 & 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil\lceil S, q\rceil 20 \; aabb3 & 02\lfloor S, q\rfloor\lfloor S, q\rfloor\lfloor X, p\rfloor\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow & 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil\lceil S, q\rceil 20 \quad aabb3 & 02\lfloor S, q\rfloor\lfloor X, p\rfloor\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow & 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil\lceil X, p\rceil 20 \quad aabb3 & 02\lfloor X, p\rfloor\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow & 1\lceil S, s\rceil\lceil S, s\rceil\lceil X, p\rceil 20 \quad aabb3 & 02\lfloor X, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow & 1\lceil S, s\rceil\lceil S, s\rceil 20 \quad aabb3 & 02\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow & 1\lceil S, s\rceil 20 \quad aabb3 & 02\lfloor S, s\rfloor 1 \\
\Rightarrow & 120 \quad aabb3 & 021 \\
\Rightarrow & 2 \quad aabb & 2 \\
\Rightarrow^2 & aabb &
\end{array}
$$

4.3.1 Kódování

Tato sekce obsahuje pouze intuitivní vysvětlení jak zavedené kódování funguje a proč platí lemma 4.2. Pro kompletní důkaz viz článek [6].

Lemma 4.1. *Každá sekvence derivačních kroků má tvar:*

$$
\begin{aligned}
& 4 \\
\Rightarrow\ & w_1\ [p_1] \Rightarrow \ldots \Rightarrow w_n\ [p_n] \\
\Rightarrow\ & w_{n+1}\ [\pi] \Rightarrow \ldots \Rightarrow w_{n+m}\ [\pi] \\
\Rightarrow\ & w_{n+m+1}\ [2 \to \varepsilon] \Rightarrow \ldots \Rightarrow w_{n+m+k}\ [2 \to \varepsilon],
\end{aligned}
$$

kde $n \geq 1$, $m \geq 0$, $0 \leq k \leq 2$, $p_1, \ldots, p_n \in M$ *a* $\pi = (1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2)$.

Pravdivost lemmatu 4.1 zaručuje tvar kontextového pravila π a tvar pravidel vygenerovaných v kroku (v). Všechna bezkontextová pravidla z množiny M přepisují pouze neterminál 4, což znamená, že pokud se neterminál 4 ve větné formě nenachází, nelze aplikovat žádné pravidlo z množiny M. Současně s tím žádné pravidlo z množiny O nezapisuje do větné formy neterminál 4. Tedy z větné formy neobsahující neterminál 4 se nelze pomocí žádné sekvence pravidel dostat do větné formy obsahující neterminál 4.

Aby bylo možné aplikovat kontextové pravidlo, musí se ve větné formě nacházet více než dva výskyty neterminálu 2. Jediná pravidla, která do větné formy vkládají navíc neterminál 2 jsou právě pravidla vygenerovaná v kroku (v). Tato pravidla ale současně z větné formy odstraňují neterminál 4 a tím pádem aplikace jednoho z těchto pravidel zaručuje striktní rozdělení sekvence derivačních kroků na dvě fáze.

Pravidlo $2 \to \varepsilon$ opět nelze aplikovat před vložením neterminálu 2 do větné formy a současně s tím aplikace tohoto pravidla znemožní další aplikaci kontextového pravidla. Tedy znamená, že aplikací tohoto pravidla každá sekvence derivačních kroků končí. Protože jsou ve větné formě maximálně dva výskyty neterminálu 2, lze toto pravidlo aplikovat nejvýše dvakrát.

Lemma 4.2. *V gramatice G existuje sekvence derivačních kroků*

$$1\lceil a_0, p_0\rceil\lceil a_1, p_1\rceil \dots \lceil a_n, p_n\rceil 20y302\lfloor b_n, q_n\rfloor\lfloor b_{n-1}, q_{n-1}\rfloor \dots \lfloor b_0, q_0\rfloor 1 \Rightarrow^* y,$$

kde $n \geq 0$, $a_0, \dots, a_n \in V - T$, $b_0, \dots, b_n \in V - T$, $p_0, \dots, p_n \in W - F$ a $q_0, \dots, q_n \in W - F$, právě tehdy, když platí

$$\forall i \in \{0, 1, \dots, n\} : a_i = b_i \wedge p_i = q_i$$

Způsob jakým kódování funguje, lze ilustrovat na eliminaci obecného kódu z množin X a Y zavedených v definici 4.1. Platí, že $u' \in X^+$ a $v' \in Y^+$, c je nějaká konstanta založená na původní frontové gramatice a $1 \leq k \leq c$ je konstanta unikátní pro každý kód. Dále platí, že pro odpovídající si kódy je k shodné. Proto lze provést následující derivaci:

	1		u'	$20y302$	v'		1
$\equiv$	1	u	$(103)^k1(10)^{(c+1)-k}$	$20y302$	$(301)^{(c+1)-k}0301(01)^k$	v	1
$\equiv$	1	u	$(103)^k1(10)^{(c+1)-k-1}\underline{10}$	$\underline{20y302}$	$30\underline{1}(301)^{(c+1)-k-1}0301(01)^k$	v	1
$\Rightarrow$	1	u	$(103)^k1(10)^{(c+1)-k-1}20$	y	$302(301)^{(c+1)-k-1}0301(01)^k$	v	1
$\dots$							
$\Rightarrow$	1	u	$(103)^k\underline{120}$	y	$\underline{3020}30\underline{1}(01)^k$	v	1
$\Rightarrow$	1	u	$(103)^k2$	y	$0302(01)^k$	v	1
$\equiv$	1	u	$(103)^{k-1}\underline{103}\underline{2}$	y	$\underline{0302}0\underline{1}(01)^{k-1}$	v	1
$\Rightarrow$	1	u	$(103)^{k-1}203$	y	$02(01)^{k-1}$	v	1
$\equiv$	1	u	$(103)^{k-2}\underline{103}\underline{203}$	y	$\underline{02}0\underline{1}(01)^{k-2}$	v	1
$\Rightarrow$	1	u	$(103)^{k-2}203$	y	$02(01)^{k-2}$	v	1
$\dots$							
$\Rightarrow$	1	u	203	y	02	v	1
$\dots$							
$\Rightarrow$	$\underline{1}$	$\underline{203}$		y		$\underline{02}$	$\underline{1}$
$\Rightarrow$	2			y			2
$\Rightarrow^2$				y			

V tomto zápisu symboly $\cdots$ značí bod, kdy se derivace dostala do větné formy, která je z pohledu kontextového pravidla shodná s větnou formou výše, a tudíž lze několik ná-

sledujících derivačních kroků přeskočit. Podtržení značí symboly přepisované kontextovým pravidlem v následujícím derivačním kroku.

Dále je vidět, že pokud by v sekvenci derivačních kroků nebylo aplikováno pravidlo vytvořené v kroku (i), a tím pádem se ve větné formě nenacházely symboly 1 na obou koncích, nebylo by možné kódy úspěšně eliminovat.

4.3.2 Korektnost množiny bezkontextových pravidel

Následuje důkaz, že první fáze derivace v gramatice G pomocí bezkontextových pravidel korektně simuluje derivaci v gramatice Q, a tím pádem i důkaz rovnosti jazyků obou gramatik jak říká tvrzení 4.1.

V rozptýlené gramatice G lze aplikovat libovolné pravidlo téměř kdykoliv, protože všechna přepisují neterminál 4 nebo 5. Nicméně k úspěšné derivaci mohou vést pouze kroky, které zachovávají správné pořadí odpovídajících si kódů na jednotlivých stranách větné formy (viz příklad 4.3). Takový derivační krok bude dále označován jako *smysluplný* derivační krok.

Příklad 4.3. Uvažujme $C, D, E \in A$, $s \in B$ a na jejich základě definované injekce ι a κ. Potom uvažujme dva derivační kroky:

$$\begin{array}{rl} & 1\lceil C,s\rceil \qquad\qquad 4 \qquad\qquad 1 \\ \Rightarrow_{\mathrm{I}} & 1\lceil C,s\rceil\lceil D,s\rceil\lceil E,s\rceil\, 4 \qquad \lfloor D,s\rfloor\lfloor C,s\rfloor 1 \\ \Rightarrow_{\mathrm{II}} & 1\lceil C,s\rceil\lceil D,s\rceil\lceil E,s\rceil\, 4\, \lfloor D,s\rfloor\lfloor D,s\rfloor\lfloor C,s\rfloor 1\,. \end{array}$$

Derivační krok I je smysluplný, protože $\lceil C,s\rceil$ odpovídá $\lfloor C,s\rfloor$ a $\lceil D,s\rceil$ odpovídá $\lfloor D,s\rfloor$. Naproti tomu derivační krok II není smysluplný, protože $\lceil E,s\rceil$ neodpovídá $\lfloor D,s\rfloor$.

Lemma 4.3. *Pravidlo vytvořené v kroku (i) ve tvaru* $4 \to 1\lceil a_0,p_0\rceil\lceil a_0,p_0\rceil 4\lfloor a_0,p_0\rfloor 1$ *musí být vždy aplikováno jako první, pokud má být derivace úspěšná.*

Předpokládejme, že pravidlo z lemmatu 4.3 (dále jen *startovní pravidlo*) nebude aplikováno jako první.

$$\begin{array}{rll} & & 4 \\ \Rightarrow^* & \lceil c_0,r_0\rceil \dots \lceil c_k,r_k\rceil & 4 \qquad\qquad \lfloor d_l,s_l\rfloor \dots \lfloor d_0,s_0\rfloor \\ \Rightarrow & \lceil c_0,r_0\rceil \dots \lceil c_k,r_k\rceil 1\lceil a_0,p_0\rceil & 4 \qquad\qquad 1\lfloor d_l,s_l\rfloor \dots \lfloor d_0,s_0\rfloor \\ \Rightarrow^* & \lceil c_0,r_0\rceil \dots \lceil c_k,r_k\rceil 1\lceil a_0,p_0\rceil \dots \lceil a_n,p_n\rceil & 20y302\, \lfloor b_m,q_m\rfloor \dots \lfloor b_0,q_0\rfloor 1\lfloor d_l,s_l\rfloor \dots \lfloor d_0,s_0\rfloor \end{array}$$

Derivace potom může probíhat dvěma způsoby podle toho, jestli platí

$$n = m \wedge (\forall i \in \{0 \dots n\} : a_i = b_i \wedge p_i = q_i)\,:$$

1. Pokud toto tvrzení platí, pak je možné dle lemmatu 4.2 odstranit všechny neterminály, které byly vloženy po aplikaci startovního pravidla a derivace bude pokračovat následovně.

   $$\begin{array}{rll} \Rightarrow^* & \dots \lceil c_k,r_k\rceil 1\lceil a_0,p_0\rceil \dots \lceil a_n,p_n\rceil 20\, y\, 302\lfloor b_m,q_m\rfloor \dots \lfloor b_0,q_0\rfloor 1\lfloor d_l,s_l\rfloor \dots \\ \Rightarrow^* & \dots \lceil c_k,r_k\rceil 120 & y \qquad\qquad 3021\lfloor d_l,s_l\rfloor \dots \\ \Rightarrow^* & \dots \lceil c_k,r_k\rceil 2 & y \qquad\qquad 2\lfloor d_l,s_l\rfloor \dots \\ \not\Rightarrow & & \end{array}$$

   Je zřejmé, že dále nelze v derivaci pokračovat, protože se mezi neterminály 2 nenachází neterminál 3 a tím pádem nelze aplikovat kontextové pravidlo.

2. V opačném případě z lemmatu 4.2 plyne, že není možné z větné formy odstranit symboly $\lceil a_0, p_0 \rceil \dots \lceil a_n, p_n \rceil$ a $\lfloor b_m, q_m \rfloor \dots \lfloor b_0, q_0 \rfloor$. To znamená, že v takovém případě není možné provést úspěšnou derivaci.

Opakovaná aplikace pravidla povede ke stejným výsledkům. Z lemmatu 4.2 navíc plyne, že první pravidlo musí být během derivace aplikováno a tedy lemma 4.3 platí.

Lemma 4.4. *Pokud byl do fronty během derivace v gramatice Q v 1. normální formě vložen alespoň jeden terminál, lze již dále aplikovat pouze pravidla vkládající do fronty terminály, jinak nebude derivace úspěšná.*

Z 1. normální formy (viz definice 3.1) plyne, že jakmile se dostane terminál na začátek fronty, tak nelze aplikovat žádné další pravidlo. Z toho plyne, že pokud se za terminálem ve frontě nachází nějaké neterminály, bude derivace neúspěšná. Aplikace pravidla vkládajícího neterminály do fronty na větnou formu, kde již nějaké terminály ve frontě jsou způsobí vložení neterminálů na konec fronty, tedy za terminály, které se ve frontě již nacházejí.

Lemma 4.5. *Větná forma*

$$b_1 \dots b_k \# a_1 \dots a_n p_1,$$

kde $a_1 \dots a_n \in (V - T)^+$, $b_1 \dots b_k \in (V - T)^*$ a $p_1 \in W - F$, v původní gramatice Q v 2. normální formě a větná forma

$$1u\lceil a_1, p_1 \rceil \lceil a_1, p_1 \rceil \lceil a_2, p_2 \rceil \lceil a_2, p_2 \rceil \dots \lceil a_n, p_n \rceil \lceil a_n, p_n \rceil 4 \lfloor a_1, p_1 \rfloor v 1,$$

kde

$$\begin{aligned} u &= \delta(\lceil b_1, q_1 \rceil \dots \lceil b_k, q_k \rceil), \\ v &= \delta(\lfloor b_k, q_k \rfloor \dots \lfloor b_1, q_1 \rfloor), \end{aligned}$$

pro nějaká $p_2, \dots, p_n \in W - F$ a $q_1, \dots, q_k \in W - F$, v gramatice G si navzájem odpovídají.

Z lemmatu 4.3 plyne, že každá úspěšná derivace řetězce v rozptýlené gramatice musí začínat derivačním krokem $4 \Rightarrow 1\lceil a_0, p_0 \rceil \lceil a_0, p_0 \rceil 4 \lfloor a_0, p_0 \rfloor 1$. To znamená, že vzniklá větná forma $1\lceil a_0, p_0 \rceil \lceil a_0, p_0 \rceil 4 \lfloor a_0, p_0 \rfloor 1$ by měla odpovídat počáteční větné formě frontové gramatiky $\# a_0 p_0$. To platí, stačí dosadit: $b_1 \dots b_k = \varepsilon$, $a_1 \dots a_n = a_0$ a $p_1 = p_0$.

Nyní uvažujme obecný přechod ve frontové gramatice

$$b_1 \dots b_k \# a_1 \dots a_n p_1 \Rightarrow b_1 \dots b_k a_1 \# a_2 \dots a_n c_1 \dots c_l r \ [(a_1, p_1, c_1 \dots c_l, r)],$$

kde $a_1 \dots a_n \in (V - T)^+$, $b_1 \dots b_k \in (V - T)^*$, $p_1, r \in W - F$ a $c_1 \dots c_l \in (V - T)^+$.

Předpokládejme, že větné formě $b_1 \dots b_k \# a_1 \dots a_n p_1$ frontové gramatiky opravdu odpovídá větná forma

$$1u\lceil a_1, p_1 \rceil \lceil a_1, p_1 \rceil \dots \lceil a_n, p_n \rceil \lceil a_n, p_n \rceil 4 \lfloor a_1, p_1 \rfloor v 1,$$

kde

$$\begin{aligned} u &= \delta(\lceil b_1, q_1 \rceil \dots \lceil b_k, q_k \rceil), \\ v &= \delta(\lfloor b_k, q_k \rfloor \dots \lfloor b_1, q_1 \rfloor). \end{aligned}$$

Pravidlo frontové gramatiky Q aplikované během derivačního kroku bude transformováno na množinu pravidel, kterou lze popsat obecným zápisem

$$4 \to \lceil c_1, s_1 \rceil \lceil c_1, s_1 \rceil \dots \lceil c_l, s_l \rceil \lceil c_l, s_l \rceil 4 \lfloor d, r \rfloor \lfloor a_1, p_1 \rfloor,$$

pro nějaká $s_1, \dots, s_l \in W - F$ a $d \in V - T$.

Pomocí tohoto pravidla proveďme derivační krok v rozptýlené gramatice G:

$$\begin{aligned} & 1u\lceil a_1,p_1\rceil\lceil a_1,p_1\rceil\lceil a_2,p_2\rceil\lceil a_2,p_2\rceil\delta(\lceil a_3,p_3\rceil\ldots) && 4 && \lfloor a_1,p_1\rfloor v1 \\ \Rightarrow\ & 1u\lceil a_1,p_1\rceil\lceil a_1,p_1\rceil\lceil a_2,p_2\rceil\lceil a_2,p_2\rceil\delta(\lceil a_3,p_3\rceil\ldots\lceil c_1,s_1\rceil\ldots) && 4\,\lfloor d,r\rfloor\lfloor a_1,p_1\rfloor\lfloor a_1,p_1\rfloor v1\,. \end{aligned}$$

Nově vzniklá větná forma by měla odpovídat větné formě $b_1\ldots b_k a_1\#a_2\ldots a_n c_1\ldots c_l p_2$ frontové gramatiky. To zřejmě platí pokud $d=a_2$ a $r=p_2$. Může nastat případ kdy $n=1$ a tím pádem $\lceil a_2,p_2\rceil\equiv\lceil c_1,s_1\rceil$, nicméně tvrzení stále platí. Případ $n=0$ nemůže nastat, protože z předpokladu platí, že $n\geq 1$ a současně pravidlo definuje, že $l\geq 1$. Pokud by jedna z těchto podmínek neplatila, je lemma 4.5 neplatná.

V množině M existuje verze každého pravidla pro každé $d\in V-T$ a tedy $d=a_2$ může platit vždy, pokud je během derivace zvolena správná verze pravidla. Stav p_2 je určen dříve aplikovaným pravidlem, které opět existuje ve verzích pro všechna $p_2\in W-F$. Lze tedy předpokládat, že byla verze pravidla zvolena nederministicky správně.

Lemma 4.6. *Větná forma*

$$b_1\ldots b_k\#a_1\ldots a_n y p_1,$$

kde $a_1\ldots a_n\in(V-T)^+$, $b_1\ldots b_k\in(V-T)^*$, $y\in T^+$ *a* $p_1\in W-F$, *v původní gramatice* Q *v 2. normální formě a větná forma*

$$1u\lceil a_1,p_1\rceil\lceil a_1,p_1\rceil\lceil a_2,p_2\rceil\lceil a_2,p_2\rceil\ldots\lceil a_n,p_n\rceil\lceil a_n,p_n\rceil 20y5\lfloor a_1,p_1\rfloor v1,$$

kde

$$\begin{aligned} u &= \delta(\lceil b_1,q_1\rceil\ldots\lceil b_k,q_k\rceil), \\ v &= \delta(\lfloor b_k,q_k\rfloor\ldots\lfloor b_1,q_1\rfloor), \end{aligned}$$

pro nějaká $p_2,\ldots,p_n\in W-F$ *a* $q_1,\ldots,q_k\in W-F$, *v gramatice* G *si navzájem odpovídají.*

Z lemmatu 4.5 víme, že větná forma $b'_1\ldots b'_k\#a'_1\ldots a'_{n'}p'_1$ v Q má odpovídající větnou formu v gramatice G. Uvažujme tedy obecný derivační krok z takovéto větné formy

$$b'_1\ldots b'_{k'}\#a'_1\ldots a'_{n'}p'_1\Rightarrow b'_1\ldots b'_{k'}a'_1\#a'_2\ldots a'_{n'}z'r'\ [(a'_1,p'_1,z',r')],$$

kde $b'_1\ldots b'_{k'}\in(V-T)^*$, $a'_1\ldots a'_{n'}\in(V-T)^+$, $p'_1,r'\in W-F$ a $z'\in T^+$.

Pravidlo aplikované v derivačním kroku frontové gramatiky Q bude v kroku (iv) transformováno na množinu pravidel, kterou lze zapsat následujícím obecným tvarem.

$$5\to z'5\lfloor d',r'\rfloor\lfloor a'_1,p'_1\rfloor,$$

pro nějaké $d'\in V-T$. Aby bylo pravidlo možné aplikovat, musí být do větné formy nejprve vložen neterminál 5, potom pomocí tohoto pravidla proveďme derivační krok v rozptýlené gramatice G z odpovídající větné formy:

$$\begin{aligned} & 1u'\lceil a'_1,p'_1\rceil\lceil a'_1,p'_1\rceil\lceil a'_2,p'_2\rceil\lceil a'_2,p'_2\rceil\ldots\lceil a'_{n'},p'_{n'}\rceil\lceil a'_{n'},p'_{n'}\rceil && 4 && \lfloor a'_1,p'_1\rfloor v'1 \\ \Rightarrow\ & 1u'\lceil a'_1,p'_1\rceil\lceil a'_1,p'_1\rceil\lceil a'_2,p'_2\rceil\lceil a'_2,p'_2\rceil\ldots\lceil a'_{n'},p'_{n'}\rceil\lceil a'_{n'},p'_{n'}\rceil && 20\ 5 && \lfloor a'_1,p'_1\rfloor v'1 \\ \Rightarrow\ & 1u'\lceil a'_1,p'_1\rceil\lceil a'_1,p'_1\rceil\lceil a'_2,p'_2\rceil\lceil a'_2,p'_2\rceil\ldots\lceil a'_{n'},p'_{n'}\rceil\lceil a'_{n'},p'_{n'}\rceil && 20z'\,5\,\lfloor d',r'\rfloor\lfloor a'_1,p'_1\rfloor\lfloor a'_1,p'_1\rfloor v'1\,, \end{aligned}$$

kde

$$\begin{aligned} u' &= \delta(\lceil b'_1,q'_1\rceil\ldots\lceil b'_{k'},q'_{k'}\rceil), \\ v' &= \delta(\lfloor b'_{k'},q'_{k'}\rfloor\ldots\lfloor b'_1,q'_1\rfloor), \end{aligned}$$

pro nějaká $p'_2,\ldots,p'_{n'}\in W-F$ a $q'_1,\ldots,q'_{k'}\in W-F$.

Nově vzniklé větné formy si navzájem odpovídají, pokud platí $d' = a'_2$ a $r' = p'_2$. Toho lze nederministicky dosáhnout vždy, jak bylo popsáno výše (viz lemma 4.5). Současně však musí platit, že $n' \geq 2$, protože jinak ve větné formě symbol a'_2 neexistuje. Pokud by tato podmínka neplatila, tak aplikací pravidla v gramatice Q dojde k situaci, kdy se na vrcholu fronty nenachází neterminál. Z definice 1. normální formy (viz definice 3.1) plyne, že symbol na vrcholu fronty nelze přepsat, protože lze přepisovat pouze neterminály. Pokud taková situace nastane v gramatice G, tak se gramatika dostane do následující větné formy:

$$1u'\lceil a'_1, p'_1\rceil\lceil a'_1, p'_1\rceil 205\lfloor d', r'\rfloor\lfloor a'_1, p'_1\rfloor\lfloor a'_1, p'_1\rfloor v'1.$$

Do větné formy ale v takovém případě nelze nikdy vložit kód odpovídající $\lfloor d', r'\rfloor$, protože žádné pravidlo přepisující neterminál 5 nevkládá do větné formy kódy na levou stranu od neterminálu 5. Současně z algoritmu transformace plyne, že pokud se ve větné formě nachází neterminál 5, již tam nelze vložit neterminál 4.

Obdobným způsobem uvažujme derivační krok

$$b_1 \dots b_k \# a_1 a_2 \dots a_n y p_1 \Rightarrow b_1 \dots b_k a_1 \# a_2 \dots a_n y z r \ [(a_1, p_1, z, r)],$$

kde $a_1 \dots a_n \in (V - T)^+$, $b_1 \dots b_k \in (V - T)^*$, $p_1, r \in W - F$ a $y, z \in T^*$ a předpokládejme, že větné formě $b_1 \dots b_k \# a_1 a_2 \dots a_n y p_1$ opravdu odpovídá větná forma

$$1u\lceil a_1, p_1\rceil\lceil a_1, p_1\rceil\lceil a_2, p_2\rceil\lceil a_2, p_2\rceil \dots \lceil a_n, p_n\rceil\lceil a_n, p_n\rceil y 4\lfloor a_1, p_1\rfloor v1,$$

kde

$$\begin{aligned} u &= \delta(\lceil b_1, q_1\rceil \dots \lceil b_k, q_k\rceil), \\ v &= \delta(\lfloor b_k, q_k\rfloor \dots \lfloor b_1, q_1\rfloor). \end{aligned}$$

Pravidlo aplikované v derivačním kroku frontové gramatiky Q bude opět transformováno na množinu pravidel zapsanou jako:

$$5 \to z5\lfloor d, r\rfloor\lfloor a_1, p_1\rfloor,$$

kde $d \in V - T$. Pomocí tohoto pravidla proveďme derivační krok v rozptýlené gramatice G:

$$\begin{aligned} & 1u\lceil a_1, p_1\rceil\lceil a_1, p_1\rceil\lceil a_2, p_2\rceil\lceil a_2, p_2\rceil\delta(\lceil a_3, p_3\rceil \dots)\, 20y \quad 5 \qquad\qquad \lfloor a_1, p_1\rfloor v1 \\ \Rightarrow\ & 1u\lceil a_1, p_1\rceil\lceil a_1, p_1\rceil\lceil a_2, p_2\rceil\lceil a_2, p_2\rceil\delta(\lceil a_3, p_3\rceil \dots)\, 20yz\, 5\, \lfloor d, r\rfloor\lfloor a_1, p_1\rfloor\lfloor a_1, p_1\rfloor v1\,. \end{aligned}$$

Nově vzniklá větná forma by měla odpovídat větné formě $b_1 \dots b_k a_1 \# a_2 \dots a_n y z r$ v Q. To platí, pokud $d = a_2$, $r = p_2$. Toho lze opět nederministicky dosáhnout vždy, jak bylo popsáno výše (viz lemma 4.5). Současně musí opět platit $n \geq 2$, což opět platí ve všech krocích vedoucích k úspěšné derivaci jak bylo popsáno výše pro n'.

Lemma 4.7. *Větná forma*

$$b_1 \dots b_k \# y f,$$

kde $b_1 \dots b_k \in (V - T)^*$, $y \in T^+$ a $f \in W - F$, v původní gramatice Q v 2. normální formě a větná forma

$$1\lceil b_1, q_1\rceil\lceil b_1, q_1\rceil \dots \lceil b_k, q_k\rceil\lceil b_k, q_k\rceil 20y302\lfloor b_k, q_k\rfloor\lfloor b_k, q_k\rfloor \dots \lfloor b_1, q_1\rfloor\lfloor b_1, q_1\rfloor 1,$$

pro nějaká $q_1, \dots, q_k \in W - F$, v gramatice G si navzájem odpovídají.

Z lemmat 4.5 a 4.6 víme, že větná forma $b_1 \dots b_k \# a_1 \dots a_n y p_1$ v Q má odpovídající větnou formu v gramatice G. Uvažujme tedy derivační krok z takovéto větné formy

$$b_1 \dots b_{k-1} \# b_k y q_k \Rightarrow b_1 \dots b_{k-1} b_k \# y z f \ [(b_k, q_k, z, f)],$$

kde $b_1 \dots b_{k-1} \in (V - T)^*$, $b_k \in V - T$, $q_k \in W - F$, $y \in T^*$, $z \in T^+$ a $f \in F$.

Pravidlo aplikované v derivačním kroku frontové gramatiky Q bude v kroku (v) transformováno na množinu pravidel, kterou lze zapsat následujícím obecným tvarem.

$$5 \to z302\lfloor b'_k, q_k \rfloor.$$

Pomocí tohoto pravidla proveďme derivační krok v rozptýlené gramatice G z odpovídající větné formy:

$$\begin{aligned} &1\lceil b_1, q_1\rceil\lceil b_1, q_1\rceil \dots \lceil b_k, q_k\rceil\lceil b_k, q_k\rceil 20\, y \quad 5 \qquad\qquad \lfloor b_k, q_k\rfloor \delta(\lfloor b_{k-1}, q_{k-1}\rfloor \dots \lfloor b_1, q_1\rfloor)1 \\ &1\lceil b_1, q_1\rceil\lceil b_1, q_1\rceil \dots \lceil b_k, q_k\rceil\lceil b_k, q_k\rceil 20\, yz\, 3\, 02\lfloor b_k, q_k\rfloor\lfloor b_k, q_k\rfloor \delta(\lfloor b_{k-1}, q_{k-1}\rfloor \dots \lfloor b_1, q_1\rfloor)1\,, \end{aligned}$$

pro nějaká $q_1, \dots, q_{k-1} \in W - F$. Je zřejmé, že vzniklé větné formy si navzájem opravdu odpovídají.

Lemma 4.8. *Smysluplný derivační krok*[2] *mezi dvěma větnými formami v gramatice G lze provést právě tehdy, když lze provést derivační krok mezi odpovídajícími větnými formami v původní gramatice Q.*

Z lemmat 4.5 a 4.6 plyne, že každé větné formě gramatiky Q

$$b_1 \dots b_k \# a_1 \dots a_n y p_1,$$

kde $b_1 \dots b_k \in (V - T)^*$, $a_1 \dots a_n \in (V - T)^+$, $y \in T^*$ a $p_1 \in W - F$, odpovídá následující větná forma v gramatice G

$$1u\lceil a_1, p_1\rceil\lceil a_1, p_1\rceil\lceil a_2, p_2\rceil\lceil a_2, p_2\rceil \dots \lceil a_n, p_n\rceil\lceil a_n, p_n\rceil wyc\lfloor a_1, p_1\rfloor v1,$$

kde $w \in \{20, \varepsilon\}$, $c \in \{4, 5\}$ a kde

$$\begin{aligned} u &= \delta(\lceil b_1, q_1\rceil \dots \lceil b_k, q_k\rceil), \\ v &= \delta(\lfloor b_k, q_k\rfloor \dots \lfloor b_1, q_1\rfloor), \end{aligned}$$

pro nějaké $q_1, \dots, q_k \in W - F$.

Neterminál, který pravidlo v gramatice G přepisuje, nemá vliv na to, jestli jeho aplikací lze provést smysluplný derivační krok. Protože pokud je ve větné formě neterminál 4 a mělo by být aplikováno pravidlo přepisující neterminál 5, lze vždy aplikovat pravidlo $4 \to 205$ a neterminál 5 tak do větné formy nejprve vložit. Opačná situace, kdy se ve větné formě nachází neterminál 5 a mělo by být aplikováno pravidlo přepisující 4, nemůže během úspěšné derivace nikdy nastat. To platí, protože pravidla přepisující neterminál 4 jsou transformovány v kroku (ii) a tedy z pravidel gramatiky Q, která vkládají alespoň jeden neterminál do fronty. Nicméně z lemmatu 4.6 plyne, že se ve frontě v odpovídající větné formě gramatiky Q nachází nejméně jeden terminál. Z lemmatu 4.4 potom plyne, že během úspěšné derivace nemůže nikdy dojít k aplikaci takového pravidla na tuto větnou formu.

Platnost lemmatu 4.8 potom plyne z následujících dvou tvrzení:

[2] Smysluplný derivační krok v gramatice G je takový krok, který může vést na úspěšnou derivaci (viz příklad 4.3).

1. *Pokud lze provést derivační krok mezi dvěma větnými formami v gramatice Q, pak lze provést smysluplný derivační krok mezi odpovídajícími větnými formami i v gramatice G.*

   Na větnou formu ve frontové gramatice Q lze pravidlo $(a, p, z, q) \in R$ aplikovat právě tehdy, když $a = a_1$ a $p = p_1$. Z algoritmu transformace plyne, že každé pravidlo vytvořené v kroku (ii), (iv) nebo (v) má tvar $c \to w\lfloor a, p \rfloor$, kde $c \in \{4, 5\}$ a $w \in \{0, 1, 2, 3, 4\}^*$. Aplikací takového pravidla lze provést smysluplný derivační krok právě tehdy, když $a = a_1$ a $p = p_1$.

2. *Pokud lze provést smysluplný derivační krok mezi dvěma větnými formami v gramatice G, pak lze provést derivační krok mezi odpovídajícími větnými formami i v gramatice Q.*

   Smysluplný derivační krok z větné formy v gramatice G musí na pravou stranu větné formy vkládat kód $\lfloor a_1, p_1 \rceil$, tedy aplikované pravidlo musí mít tvar $c \to w\lfloor a_1, p_1 \rfloor$, kde $c \in \{4, 5\}$ a $w \in \{0, 1, 2, 3, 4\}^*$. Z transformace opět plyne, že takové pravidlo může být vytvořeno v kroku (ii), (iv) nebo (v) pouze z pravidla $(a, p, z, q) \in R$, kde $a = a_1$ a $p = p_1$. Takové pravidlo lze zřejmě aplikovat na větnou formu v Q.

Tvrzení 4.1. $L(Q) = L(G)$

Pro každý řetězec $y \in L(Q)$ existuje sekvence derivačních kroků $\#a_0 s_0 \Rightarrow^* w\#yf$, kde $w \in (V - T)^*$ a $f \in F$. V gramatice G existuje vždy odpovídající sekvence smysluplných derivačních kroků. Dle lemmatu 4.3 je jako první proveden startovní krok. Následně lemma 4.8 říká, že pro každý derivační krok ve frontové gramatice Q lze v rozptýlené gramatice G provést odpovídající smysluplný derivační krok. Protože všechny provedené derivační kroky byly smysluplné, bylo zachováno správné pořadí kódů a ty tak mohou být eliminovány pomocí kontextového pravidla dle lemmatu 4.2.

Lemma 4.8 rovněž zaručuje, že pokud $y \notin L(Q)$ a tedy neexistuje sekvence derivačních kroků vedoucí na derivaci řetězce y v Q, pak taková sekvence neexistuje ani v rozptýlené gramatice G.

4.4 Analýza transformace

Tato kapitola se zabývá analýzou navržené transformace v sekci 4.1.2. To se týká jak jejích omezení a složitostí, tak složitostí výsledného modelu.

4.4.1 Limitace transformace

Navržená transformace v sekci 4.1.2 není korektní pokud frontová gramatika obsahuje pravidla, která vkládají do fronty prázdný řetězec. Tato limitace vyplývá z důkazu lemmatu 4.5, kde je vyžadováno aby vkládaný řetězec nebyl prázdný.

Problém spočívá v tom, že frontová gramatika vkládající prázdné řetězce se může dostat do situace, kdy je fronta prázdná a tím pádem nelze aplikovat žádné pravidlo. V transformované rozptýlené gramatice ale není nijak zakódována délka fronty a tím pádem může dojít k přepsání neterminálu, který bude do fronty vložen až v dalším kroku. V rozptýlené gramatice se tato situace projeví tím, že je ve větné formě více kódů napravo od symbolu 4 nebo 5 než nalevo. Celá situace je prezentována v protipříkladu 4.4.

Příklad 4.4. Mějme levě rozšířenou frontovou gramatiku $Q = (V, T, W, F, s, R)$, kde $V = \{S, X, a\}$, $T = \{a\}$, $W = \{s, p, q, f\}$, $F = \{f\}$ a $s = Ss$. Množina pravidel R obsahuje pravidla:

1. (S, s, ε, p);
2. (S, p, SX, q);
3. (X, q, aa, f).

Jazyk gramatiky je zřejmě $L(Q) = \emptyset$. Jednotlivá pravidla gramatiky jsou očíslována tak, aby se na ně bylo možné v sekvenci derivačních kroků odkazovat a zvýšila se tak přehlednost zápisu.

Gramatiku Q dále na základě algoritmu v sekci 4.1.2 transformujme na gramatiku $G = (N, T, P, 4)$, kde $N = \{0, 1, 2, 3, 4, 5\}$. Množina pravidel P bude mimo jiné obsahovat pravidla:

1. Pravidlo $4 \to 1\lceil S, s\rceil\lceil S, s\rceil 4\lfloor S, s\rfloor 1$ vytvořené v kroku (i) na základě počáteční větné formy Ss;
2. Pravidlo $4 \to 4\lfloor S, p\rfloor\lfloor S, s\rfloor$ vytvořené v kroku (ii) na základě pravidla $(S, s, \varepsilon, p) \in R$;
3. Pravidlo $4 \to \lceil S, p\rceil\lceil S, p\rceil\lceil X, q\rceil\lceil X, q\rceil 4\lfloor X, q\rfloor\lfloor S, p\rfloor$ vytvořené v kroku (ii) na základě pravidla $(S, p, SX, q) \in R$;
4. Pravidlo $4 \to 205$;
5. Pravidlo $5 \to aa302\lfloor X, q\rfloor$ vytvořené v kroku (v) na základě pravidla $(X, q, aa, f) \in R$;

V této gramatice lze následujícím způsobem derivovat řetězec aa a tím pádem $L(Q) \neq L(G)$.

$$
\begin{aligned}
& & 4 & \\
\Rightarrow\ & 1\lceil S, s\rceil\lceil S, s\rceil & 4 & \lfloor S, s\rfloor 1 \\
\Rightarrow\ & 1\lceil S, s\rceil\lceil S, s\rceil & 4 & \lfloor S, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow\ & 1\lceil S, s\rceil\lceil S, s\rceil\lceil S, p\rceil\lceil S, p\rceil\lceil X, q\rceil\lceil X, q\rceil & 4 & \lfloor X, q\rfloor\lfloor S, p\rfloor\lfloor S, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow\ & 1\lceil S, s\rceil\lceil S, s\rceil\lceil S, p\rceil\lceil S, p\rceil\lceil X, q\rceil\lceil X, q\rceil 20 & 5 & \lfloor X, q\rfloor\lfloor S, p\rfloor\lfloor S, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow\ & 1\lceil S, s\rceil\lceil S, s\rceil\lceil S, p\rceil\lceil S, p\rceil\lceil X, q\rceil\lceil X, q\rceil 20\, aa3\, 02 & \lfloor X, q\rfloor\lfloor X, q\rfloor\lfloor S, p\rfloor\lfloor S, p\rfloor\lfloor S, s\rfloor\lfloor S, s\rfloor 1 \\
\Rightarrow^* & & aa &
\end{aligned}
$$

4.4.2 Analýza složitostí

Nová transformace byla navržena s cílem snížit popisnou složitost výsledné gramatiky G z pohledu počtu neterminálů. Transformace byla v tomto ohledu úspěšná a pro novou gramatiku platí:

- Počet neterminálů je konstantní, konkrétně je jich 6;
- Počet kontextových pravidel je rovněž konstantní, konkrétně je právě jedno;

- Počet bezkontextových pravidel závisí na frontové gramatice Q a složitost tedy je:

$$\mathcal{O}\left(|R| \cdot |W-F|^k \cdot |V-T|\right),$$

  kde k je takové číslo, že platí $\forall(a,p,y,q) \in R : |y| \leq k$.

Popisná složitost počtu bezkontextových pravidel byla odvozena z kroku (ii), protože v něm bude zřejmě vždy vytvořeno nejvíce pravidel. Pro každé pravidlo v $(a,p,y,q) \in R$ bude pro každý neterminál vytvořeno několik pravidel, v algoritmu je to zapsáno pomocí μ. Pro každý neterminál bude na základě délky řetězce y vytvořeno nové pravidlo pro každou možnou kombinaci stavů této délky, v algoritmu je to zapsáno pomocí ν.

Časová složitost výsledného modelu je odvozena od počtu bezkontextových pravidel, protože kontextové pravidlo lze aplikovat vždy pouze jen na jednu danou pozici (viz důkaz lemmatu 4.2), tak celý výpočet probíhá pomocí bezkontextových pravidel. Ze vstupu bohužel nelze snadno odvodit kolik pravidel bude nutné aplikovat k derivaci řetězce, nicméně je jasné, že časová složitost bude exponenciální a sice:

$$\mathcal{O}\left(\left(|R| \cdot |W-F|^k \cdot |V-T|\right)^n\right),$$

kde k je takové číslo, že platí $\forall(a,p,y,q) \in R : |y| \leq k$, a n značí počet derivačních kroků vedoucích k derivaci řetězce.

Prostorová složitost výsledné gramatiky podle lemmat 4.5 a 4.6 závislá opět na počtu derivačních kroků, protože levě rozšířená frontová gramatika uchovává ve větné formě informaci o každém kroku. Konkrétně pro každý derivační krok budou vloženy dvě dvojice odpovídajících si kódů. Současně do prostorové složitosti zasahuje kódování, kde je délka jednoho kódu závislá na počtu neterminálů a stavů frontové gramatiky. Prostorová složitost výsledného modelu tedy je:

$$\mathcal{O}\left(n \cdot c\right),$$

kde n značí počet derivačních kroků vedoucích k derivaci řetězce a $c \geq |V-T| \cdot |W-F|$ z definice kódování v sekci 4.1.1.

Časová složitost transformace je přímo závislá na počtu vytvořených bezkontextových pravidel, protože všechny ostatní operace v algoritmu mají konstantní složitost. Tedy časová složitost je rovněž:

$$\mathcal{O}\left(|R| \cdot |W-F|^k \cdot |V-T|\right),$$

kde k je takové číslo, že platí $\forall(a,p,y,q) \in R : |y| \leq k$.

Samotná transformace nepracuje s pamětí. Každé pravidlo frontové gramatiky může být zpracováno individuálně, protože všechny informace potřebné k přidání nových pravidel do množiny M jsou samotné pravidlo a množiny V, T, W a F, které jsou součástí vstupu. Kódování jednotlivých symbolů rovněž není nutné si pamatovat, protože ho lze vždy vypočítat ze vstupu pomocí zobrazení ι a κ. Proto je prostorová složitost transformace konstantní.

4.4.3 Redukce na pět neterminálů

Snížení počtu neterminálů na pět v současném algoritmu transformace v sekci 4.1.2 by znamenalo nahrazení neterminálů 4 a 5 jediným neterminálem 4. To ale není možné, protože by neplatila lemma 4.6. Během důkazu tohoto lemmatu by nebylo možné zaručit, že rozptýlená gramatika z větné formy obsahující více kódů napravo od symbolu 5 než nalevo neprovede úspěšnou derivaci.

Frontová gramatika v 1. normální formě může přepisovat pouze neterminály. Z toho plyne, že pokud se ve frontě nachází terminál, žádné neterminály vložené za něj nelze přepsat. Pokud by tam nějaké byl, bude derivace vždy neúspěšná. Ale rozptýlená gramatika s 5 neterminály nekóduje do větné formy terminály (pouze je tam vkládá tak jak jsou), takže může dojít k situaci, kdy rozptýlená gramatika přepíše neterminál, který by se ve frontové gramatice nacházel ve frontě až za terminálem.

Kapitola 5

Závěr

Tato práce se zabývala opravou algoritmu převodu frontové gramatiky do 1. normální formy a návrhem nové transformace frontových gramatik na gramatiky s rozptýleným kontextem s omezenou popisnou složitostí.

Jako součást práce byla vytvořena aplikace sloužící k analýze a testování jednotlivých transformací. Aplikace byla navržena jako jeden univerzální nástroj, který pracuje jak s normálními formami frontových gramatik, tak s gramatikami s rozptýleným kontextem. To umožnilo sdílet společné části implementace mezi jednotlivými případy užití. Jedná se o multiplatformní konzolovou aplikaci napsanou v jazyce Swift.

V první části práce se podařilo úspěšně opravit chybný algoritmus transformace frontové gramatiky do 1. normální formy. Hromadné testování bylo prováděno pomocí vytvořené aplikace na 13 gramatikách různé složitosti a bylo tak ukázáno, že navržená oprava algoritmu je korektní. Navíc byla navržena upravená verze algoritmu, která mírně snižuje počet pravidel výsledné gramatiky. Tato verze byla rovněž hromadně testována na 13 gramatikách a bylo demonstrováno, že je korektní.

V druhé části práce byla navržena nová transformace frontových gramatik na gramatiky s rozptýleným kontextem a s jedním kontextovým pravidlem. Tato transformace zachovává kontextové závislosti na minimu díky pouze jednomu kontextovému pravidlu. Současně má výsledná gramatika významně omezenou popisnou složitost, protože vždy obsahuje pouze 6 neterminálů. Dále byla dokázána korektnost navržené transformace a byly analyzovány složitosti jak transformace tak výsledné gramatiky.

Bohužel však transformace zavádí nové požadavky na vstupní gramatiku, konkrétně nesmí gramatika pracovat s prázdnými řetězci, a tedy není obecně korektní pro všechny frontové gramatiky. Otázka, zda je možné tyto nedostatky opravit a zachovat ve výsledné gramatice 6 neterminálů, tedy zůstává otevřená. Dále byla zkoumána možnost transformace na gramatiku o 5 neterminálech. Možnost jednoduché úpravy transformace tak, aby výsledná gramatika obsahovala pouze 5 neterminálů, byla v této práci vyvrácena. Nicméně otázka existence takové transformace zůstává také stále otevřena.

Literatura

[1] Holas, D. *Demonstrace rozptýlených gramatik s jediným kontextovým pravidlem* [online]. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Vedoucí práce Křivka, Z. Dostupné z: http://hdl.handle.net/11012/199461.

[2] Horáček, P., Meduna, A. a Tomko, M. *Handbook of Mathematical Models for Languages and Computation*. The Institution of Engineering and Technology, 2020. 750 s. ISBN 978-1-78561-659-4. Dostupné z: https://www.fit.vut.cz/research/publication/12041.

[3] Kleijn, H. C. M. a Rozenberg, G. On the generative power of regular pattern grammars. *Acta Informatica*. prosinec 1983, sv. 20, č. 4, s. 391–411. DOI: 10.1007/BF00264281. ISSN 1432-0525. Dostupné z: https://doi.org/10.1007/BF00264281.

[4] Kolář, D. a Meduna, A. Regulated Pushdown Automata. *Acta Cybernetica*. 2000, sv. 2000, č. 4, s. 653–664. ISSN 0324-721X. Dostupné z: https://www.fit.vut.cz/research/publication/6020.

[5] Křivka, Z. a Meduna, A. *A Reduction of Scattered Context Grammars*.

[6] Křivka, Z. a Meduna, A. Scattered Context Grammars with One Non-Context-Free Production are Computationally Complete. *Fundamenta Informaticae*. 2021, sv. 2021, č. 1, s. 361–384. ISSN 0169-2968. Dostupné z: https://www.fit.vut.cz/research/publication/11559.

Příloha A

Manuál

Aplikace vytvořená jako součást této práce sloučí jako nástroj pro snadnější analýzu frontových gramatik a jejich transformací. Překlad probíhá standardním způsobem pomocí příkazu `make`, který používá překladač jazyka Swift[1].

Aplikace je rozdělena na dvě části. Jedna z nich slouží k analýze transformace do 1. normální formy a druhá souží k analýze transformace na gramatiky s rozptýleným kontextem.

A.1 1. normální forma

Část aplikace sloužící k analýze algoritmu normalizace obsahuje tři různé nástroje:

- Nástroj `compare` slouží k automatickému porovnání derivace řetězce ve frontové gramatice a jejímu normalizovanému protějšku;
- Nástroj `search` slouží k automatické derivaci řetězce v zadané frontové gramatice;
- Nástroj `test` slouží k hromadnému testování rovnosti jazyků frontových gramatik a jejich normalizovaných verzí.

Volání z příkazové řádky tedy může mít tři různé varianty:

- `qg-transformer 1nf compare [options] [--input-path <input-path>] [--max-depth <max-depth>] <sentence>`
- `qg-transformer 1nf search [options] [--input-path <input-path>] [--max-depth <max-depth>] <sentence>`
- `qg-transformer 1nf test [options] --output-dir <output-dir> [--inputs <input-path>] [--max-length <max-length>] [--max-depth <max-depth>] <sentence>`

Následuje popis jednotlivých argumentů:

- Argument `sentence` určuje řetězec, který má být ve vstupní gramatice nebo vstupních gramatikách derivován;
- Přepínač `--show-grammar(s)` vytiskne vstupní a transformované gramatiky do příkazové řádky;

[1] https://www.swift.org

- Přepínač `--show-result(s)` vytiskne, zda zadaný řetězec patří do jazyka příslušné gramatiky;
- Přepínač `--show-derivation` vytiskne sekvenci derivačních kroků vedoucí k derivaci řetězce `sentence`;
- Přepínač `--input-path <input-path>` určuje vstupní frontovou gramatiku;
- Přepínač `--max-depth <max-depth>` určuje maximální hloubku prohledávání;
- Přepínač `--single-step-fix` použije verzi algoritmu normalizace, kde je opraven problém, kdy nelze derivovat řetězce pomocí jednoho derivačního kroku;
- Přepínač `--too-short-fix` ppoužije verzi algoritmu normalizace, kde je opraven problém, kdy nelze derivovat řetězce příliš krátkou sekvencí derivačních kroků;
- Přepínač `--rules-reduction` použije navrženou verzi algoritmu, která snižuje počet vytvořených pravidel;
- Přepínač `--normalize` normalizuje zadanou gramatiku před derivací řetězce `sentence`;
- Přepínač `--silence-search` zakáže tisk průběhu prohledávání během derivace řetězce sentence;
- Přepínač `--inputs` určuje soubory s vstupními frontovými gramatikami;
- Přepínač `--output-dir` určuje adresář do kterého mají být uloženy výsledky hromadného testování;
- Přepínač `--max-length` určuje maximální délku řetězců testovaných během hromadného testování;
- Přepínač `--parallel` povoluje paralelní testování řetězců během hromadného testování;
- Přepínač `--language-only` zapíná pouze testování řetězců, které patří do jazyka vstupní gramatiky.

Speciální symboly byly převedeny do kódování ASCII tak, aby mohla být zajištěna jejich unikátnost, podle tabulky A.1.

Symbol	ASCII
1	|
f_N	§
ε	_

Tabulka A.1: Převod speciálních symbolů do kódování ASCII, aby mohly být zobrazeny v příkazové řádce.

A.2 Gramatiky s rozptýleným kontextem

Tato část aplikace slouží jako pomocný nástroj k derivaci řetězců. Interaktivním způsobem uživateli nabízí dostupné derivační kroky a pomáhá mu tak se posouvat mezi jednotlivými větnými formami.

Pro použití této části aplikace je třeba znát algoritmus transformace navržený v práci. Volání z příkazové řádky je následující:

```
qg-transformer scg (two-phase|simplified) [options] [--input-path
        <input-path>] [--max-depth <max-depth>]
```

Následuje popis jednotlivých argumentů:

- Příkaz `two-phase` použije verzi algoritmu, která používá pravidlo $4 \to 205$ a tím je derivace rozdělena na dvě fáze;
- Příkaz `simplified` použije verzi algoritmu, kde pravidlo $4 \to 205$ není vytvořeno a tím pádem lze libovolné pravidlo aplikovat kdykoliv;
- Přepínač `--simplified-start` použije verzi algoritmu, kdy je na základě počáteční větné formy vytvořeno pouze jedno pravidlo;
- Přepínač `--start-symbol-removed` nahradí neterminály S a 4 jedním společným neterminálem 4;
- Přepínač `--five-symbol-removed` nahradí neterminály 4 a 5 jedním společným neterminálem 4;
- Přepínač `--algorithm-fix` opravuje návrh algoritmu, všechna pravidla $(a, p, y, q) \in R$ již do vytvořených pravidel zapisují i symbol a;
- Přepínač `--test-in-qg` automaticky testuje, zda nalezený řetězec patří do jazyka původní frontové gramatiky Q;
- Přepínač `--hide-nonsense` skrývá větné formy, které podle heuristiky nemohou vést na úspěšnou derivaci, tedy očekávaný počet kroků potřebných k dokončení derivace je ∞;
- Přepínač `--max-depth <max-depth>` určuje maximální hloubku prohledávání při použití přepínače `--test-in-qg`;
- Přepínač `--input-path <input-path>` určuje vstupní frontovou gramatiku.

Navrženému algoritmu odpovídá volání:

```
qg-transformer scg two-phase --simplified-start --start-symbol-removed
                            --algorithm-fix
```

A.2.1 Formát výstupu

Během asistované derivace nástroj nejprve vypíše transformovanou gramatiku a následně vždy zobrazuje současnou větnou formu `w0` a z ní proveditelné derivační kroky ve formátu ve výpisu A.1. Uživatel volí další akci zadáním znaku v hranatých závorkách. Buď se jedná o číslo derivačního kroku, nebo o příkaz. Příkazy lze zadávat i celými jmény.

- Volba derivačního kroku `[i]: (si) wi [pi]`, kde `i` je číslo volby, `si` je očekávaný počet kroků potřebných k dokončení derivace, `pi` je pravidlo aplikované v tomto derivačním kroku a `wi` je výsledná větná forma, způsobí posun do větné formy `wi`;
- Volba příkazu `[s]kip` se zobrazí pouze pokud lze aplikovat kontextové pravilo a ukončí derivaci jako úspěšnou pokud jsou na obou stranách větné formy odpovídající kódy. Nepoužívá přímo kontextové pravidlo, pouze porovnává zástupné symboly pro jednotlivé kódy;
- Volba příkazu `[a]utorun` se zobrazí pouze pokud lze aplikovat kontextové pravilo a automaticky simuluje jeho aplikaci dokud nedojde k nejasnosti ohledně dalšího postupu.
- Volba příkazu `[r]utorun` označí současnou větnou formu jako nevedoucí k úspěšné derivaci a vrátí se o jeden derivační krok zpět. Zadáním řetězce $\texttt{r}^n$ se lze vrátit o n derivačních kroků zpět.

```
w0 =>
Available steps:
[1]: (s1) w1 [p1]
[2]: (s1) w2 [p2]
...
[s]kip: Skip context free productions, decide only via compound symbols
[a]utorun: Simultate context free productions until none or than one
  applications are available
[r]eturn: Go back
Select:
```

Výpis A.1: Formát interaktivní volby derivačního kroku.

Speciální znaky používané během derivace jsou převedeny do kódování ASCII na základě tabulky A.2.

Symbol	ASCII
⌈	[
⌉	]
⌊	<
⌋	>
∞	oo
ε	_

Tabulka A.2: Převod speciálních symbolů do kódování ASCII, aby mohly být zobrazeny v příkazové řádce.

A.3 Formát vstupní gramatiky

Vstupní gramatika je zadávána ve formátu JSON. Aplikace nepodporuje celé rozšíření JSON5[2], nicméně oproti standardu JSON aplikace navíc podporuje jednořádkové komentáře.

[2] https://json5.org

Jak je vidět na příkladu A.1, gramatika je reprezentována jedním objektem s následujícími položkami:

- Položka `symbols` typu pole představuje abecedu symbolů gramatiky;
- Položka `terminals` typu pole představuje abecedu terminálů gramatiky;
- Položka `states` typu pole představuje množinu stavů gramatiky;
- Položka `final_states` typu pole představuje množinu koncových stavů gramatiky;
- Položka `start` typu řetězec obsahuje počáteční symbol následovaný počátečním stavem;
- Položka `productions` typu pole představuje množinu pravidel gramatiky.

Symboly a stavy jsou zapisovány jako jeden znak za kterým může následovat nepovinná číslice. Pravidla jsou zapisována ve tvaru pole o čtyřech položkách v přesně daném pořadí odpovídajícím formálnímu zápisu pravidla. Pro zápis prázdného řetězce se používá `""` nebo `"_"`.

Příklad A.1. Mějme gramatiku $Q_{11} = (V, T, W, F, s, R)$, kde $V = \{A, a\}$, $T = \{a\}$, $W = \{s_0, f\}$, $F = \{f\}$, $s = As_0$ a $R = \{(A, s_0, aA, s_0), (A, s_0, \varepsilon, f), (a, s_0, a, s_0)\}$. Reprezentace této gramatiky ve formátu JSON je ve výpisu A.2.

```
{
    "symbols": ["A", "a"],
    "terminals": ["a"],
    "states": ["s0", "f"],
    "final_states": ["f"],
    "start": "As0",
    "productions": [
        ["A", "s0", "aA", "s0"],
        ["A", "s0", "", "f"],
        ["a", "s0", "a", "s0"]
    ]
}

// { a^n, n >= 0 }
```

Výpis A.2: Vstupní soubor ve formátu JSON pro gramatiku Q_{11}

A.4 Návratové kódy

Návratové kódy aplikace lze rozdělit do několika kategorií:

Standardní kódy

- Kód `0` značí úspěch;
- Kód `1` značí neúspěch;

Chyby zpracování vstupu

- Kód `10` značí nevalidní označení symbolu;
- Kód `11` značí nevalidní označení stavu;
- Kód `12` značí chybnou počáteční větnou formu;
- Kód `13` značí nevalidní zápis pravidla;
- Kód `14` značí, že nějaký symbol nebyl definován v množině symbolů;
- Kód `15` značí nevalidní symbol na nějaké pozici;
- Kód `16` značí, že bylo použito jiné kódování než UTF-8;

Chyby během hromadného testování

- Kód `20` značí chybu v zápisu do souboru s výsledky;

Chyby příkazové řádky

- Kód `30` značí, že byl nalezen konec souboru tam, kde byl očekáván vstup;

Ostatní chyby

- Kód `40` značí neočekávanou chybu, která nespadá do žádné jiné kategorie.