VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

DEMONSTRACE ROZPTÝLENÝCH GRAMATIK S JEDINÝM KONTEXTOVÝM PRAVIDLEM

DEMONSTRATION OF SCATTERED CONTEXT GRAMMARS WITH SINGLE CONTEXT-SENSITIVE RULE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DAVID HOLAS**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Holas David**
Program: Informační technologie
Název: **Demonstrace rozptýlených gramatik s jediným kontextovým pravidlem**
Demonstration of Scattered Context Grammars with Single Context-Sensitive Rule
Kategorie: Překladače

Zadání:

1. Seznamte se s rozptýlenými gramatikami z oblasti formálních jazyků.
2. Prostudujte formální konstrukci ekvivalentních rozptýlených gramatik, které obsahují pouze jedno kontextové pravidlo.
3. Dle konzultací s vedoucím navrhněte vhodné datové struktury a způsob demonstrace tohoto převodu včetně jeho implementace.
4. Implementovaný nástroj testujte alespoň na 10 různých gramatikách a demonstrujte řadu přijatých i odmítnutých vět v zadané i zkonstruované gramatice.
5. Zhodnoťte implementovaný nástroj a pokuste se odhadnout či experimentálně vyhodnotit časovou složitost implementovaného algoritmu.

Literatura:

- MEDUNA Alexander a TECHET Jiří. Scattered Context Grammars and their Applications. WIT Press, UK. WIT Press, UK: WIT Press, 2010.
- MEDUNA Alexander a KŘIVKA Zbyněk. Scattered Context Grammars with One Non-Context-Free Production are Computationally Complete. Fundamenta Informaticae, submitted.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a návrh z 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. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Práce se zabývá tvorbou aplikace, která ze zadané frontové gramatiky konstruuje gramatiku s rozptýleným kontextem a jedním kontextovým pravidlem, derivuje v ní zadaný řetězec a celý proces prezentuje uživateli. K derivaci řetězců byla navržena heuristika, která snižuje počet aplikovatelných pravidel v každé větné formě. Uživatel může ovlivnit průběh derivace ručním výběrem pravidel. Průběh derivace je vypisován v podobě čistého textu nebo HTML dokumentu.

Byly odhaleny dvě chyby v teoretickém podkladu práce, jmenovitě v převodu frontové gramatiky do 1. normální formy a v konstrukci gramatik s rozptýleným kontextem a jedním kontextovým pravidlem. V práci se podařilo nalézt částečné řešení druhého problému, které je implementováno v aplikaci.

Abstract

The goal is to create an application that constructs scattered context grammars with single context-sensitive rule from queue grammars, and derives strings using them. The application presents the whole process to user in form of plain text or HTML. In order to derive strings, the heuristic is used to reduce the number of applicable rules in each sentential form. Applicable rules can be selected manually by the user in order to alter derivation process.

Two errors were discovered in underlying theory of thesis. Specifically in transformation of queue grammars into first normal form and the construction of scattered context grammars with single context-sensitive rule. A partial solution to the second error was found and is implemented in application.

Klíčová slova

Gramatiky s rozptýleným kontextem, kontextové pravidlo, frontové gramatiky, normální formy, derivace řetězců, demonstrace, jazyk C, prohledávání stavového prostoru.

Keywords

Scattered context grammars, non-context-free production, queue grammars, normal forms, string derivation, demonstration, C language, state space search.

Citace

HOLAS, David. *Demonstrace rozptýlených gramatik s jediným kontextovým pravidlem.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zbyněk Křivka, Ing., Ph.D.

Demonstrace rozptýlených gramatik s jediným kontextovým pravidlem

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
10. května 2021

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Zbyňku Křivkovi, Ph.D. za rady, připomínky a pomoc při řešení této práce.

Obsah

Kapitola 1

Úvod

Cílem práce je vytvořit aplikaci, která bude uživateli prezentovat způsob, jakým rozptýlené gramatiky s jedním kontextovým pravidlem zpracovávají řetězce. Zpracování řetězců v takové gramatice je složitý problém, který spočívá ve snaze najít správnou posloupnost pravidel, pomocí kterých gramatika řetězec generuje. Současně je tedy cílem práce i vymyslet algoritmus, kterým aplikace nalezne správnou posloupnost pravidel.

Práce studuje algoritmy z teorie formálních jazyků. Zaměřuje se na řízené gramatiky, a to konkrétně frontové gramatiky a gramatiky s rozptýleným kontextem. Obě gramatiky mají výpočetní sílu Turingova stroje.

Celá práce je založená na článku [6] o gramatikách s rozptýleným kontextem s jedním kontextovým pravidlem. Tento článek dokazuje, že každá frontová gramatika může být převedena na ekvivalentní gramatiku s rozptýleným kontextem a pouze jedním kontextovým pravidlem. Cílem tedy je vytvořit nástroj, který bude možné použít k demonstraci těchto gramatik a ověřit závěry článku na konkrétních gramatikách.

Vytvořená aplikace bude v gramatice s rozptýleným kontextem derivovat zadaný řetězec a celý proces prezentovat uživateli. Při derivaci řetězce v takovéto gramatice musí aplikace nedeterministicky vybírat pravidla. Součástí práce je proto i návrh a implementace heuristiky, která bude výběr pravidel provádět a tím nedeterminismus alespoň částečně řešit.

Protože aplikace slouží hlavně k demonstraci gramatik, předpokládá se, že bude pracovat s jednoduššími gramatikami. Není tedy nutné, aby aplikace zvládala derivaci libovolně dlouhých řetězců v libovolně složitých gramatikách. Jak složité mohou být zadávané gramatiky a jak dlouhé mohou být řetězce bude také záviset na navržené heuristice.

Implementovaná aplikace bude následně testována na nejméně deseti různých frontových gramatikách. V každé gramatice bude pomocí aplikace derivováno větší množství řetězců a výsledky budou porovnány se základním algoritmem s výběrem pravidel dle pořadí v seznamu pravidel. Cílem testování je ověřit správnost implementace všech algoritmů pro převod jednotlivých gramatik a ověřit kvalitu navržené heuristiky.

Práce začíná kapitolou 2 o studovaných gramatikách, kde jsou vysvětleny principy fungování jak gramatik s rozptýleným kontextem, tak frontových gramatik. Následuje kapitola 3, která popisuje, jakým způsobem bude aplikace tyto gramatiky demonstrovat. Tato kapitola rovněž obsahuje podrobný návrh aplikace a jejích funkcí. Kapitola 4 podrobněji popisuje jednotlivé algoritmy použité k demonstraci a za ní následuje kapitola o navržené heuristice, kde je popsáno, jakým způsobem heuristika pracuje a jakých principů využívá. Práce je zakončena kapitolou o testování, která se zabývá ověřením, zda implementovaná aplikace funguje tak, jak má, a současně hodnocením kvality navržené heuristiky.

Kapitola 2

Studované gramatiky

Gramatiky s rozptýleným kontextem s jedním kontextovým pravidlem (dále jen SCG_1) jsou podmnožinou gramatik s rozptýleným kontextem. $\mathrm{SCG_1}$ vznikají odvozením od frontových gramatik ve dvou krocích. Nejprve je frontová gramatika převedena na ekvivalentní gramatiku s rozptýleným kontextem s omezeným počtem kontextových pravidel (dále jen jako SCG_n). $\mathrm{SCG_n}$ je následně zakódováním terminálů převedena na ekvivalentní $\mathrm{SCG_1}$. Způsob derivace řetězců v $\mathrm{SCG_n}$ a $\mathrm{SCG_1}$ zůstává stejný a i všechny vlastnosti zůstanou zachovány.

Pokud se tedy v práci mluví o $\mathrm{SCG_n}$, text se vztahuje i na $\mathrm{SCG_1}$. Zkratka SCG_1 je v práci používána k popisu vlastností, které jsou specifické pouze pro gramatiky s jedním kontextovým pravidlem

2.1 Frontové gramatiky

Frontové gramatiky přepisují symboly s využitím fronty tak, že každé pravidlo odstraní symbol na začátku fronty a může na konec fronty umístit řetězec nových symbolů, nebo změnit stav, ve kterém se gramatika nachází. Derivační krok lze pomocí daného pravidla provést pouze tehdy, pokud pravidlu odpovídá jak symbol na začátku fronty, tak současný stav. Řetězec je přijat, pokud se gramatika nachází v koncovém stavu a ve frontě je pouze odpovídající řetězec [5][6].

Formálně je frontová gramatika šestice, $Q = (V, T, W, F, s, R)$, kde V představuje abecedu symbolů, W abecedu stavů, T abecedu terminálů a F abecedu koncových stavů. s je výchozí řetězec s počátečním symbolem a stavem a R je konečná množina pravidel. Tedy platí $V \cap W = \emptyset$, $T \subseteq V$, $F \subseteq W$ a $s \in (V - T)(W - F)$.

Množina pravidel je formálně definována jako konečná relace $R \subseteq (V \times (W - F)) \times (V^* \times W)$ tak, že pro každé $a \in V$ existuje prvek (a, p, z, q). Pokud máme řetězce $u \in V^+W$ a $v \in V^*W$ takové, že $u = arp$ a $v = rzq$, kde $a \in V$, $r, z \in V^*$ a $p, q \in W$, pak lze v Q pomocí pravidla $(a, p, z, q) \in R$ provést derivační krok $u \Rightarrow v[(a, p, zq)]$, což lze jako v jiných gramatikách zjednodušit na tvar $u \Rightarrow v$ a běžným způsobem rozšířit na $\Rightarrow^n$, kde $n \geq 0$, $\Rightarrow^+$ a $\Rightarrow^*$. Jazyk, který gramatika Q popisuje je definován jako: $L(Q) = \{w \in T^* : s \Rightarrow^* wf, f \in F\}$. Tato definice je převzata z článků [5][6].

2.1.1 Levé rozšíření

Levě rozšířená frontová gramatika pracuje stejně jako běžná frontová gramatika. Jediným rozdílem je, že levě rozšířená gramatika uchovává ve větné formě historii přepisu neter-

minálů. Historie je zapisována nalevo od speciálního symbolu #. Řetězce jsou přijímány stejně jako v běžné frontové gramatice nezávisle na historii přepisu. Následující definice byla převzata z článků [5][6].

Levě rozšířená frontová gramatika je tedy šestice $Q = (V, T, W, Fs, R)$ stejně jako běžná frontová gramatika, ale navíc zavádí symbol $\# \notin V \cup W$. Pokud máme řetězce $u, v \in V^*\{\#\}V^*W$ takové, že $u = w\#arp$ a $v = wa\#rzc$, kde $a \in V$, $r, z, w \in V^*$, $b, c \in W$, pak lze v Q pomocí pravidla $(a, b, z, c) \in R$ provést derivační krok $u \Rightarrow v[(a, p, z, q)]$, což lze stejně jako u frontové gramatiky zjednodušit na $u \Rightarrow v$ a běžným způsobem rozšířit na $\Rightarrow^n$, $\Rightarrow^+$ a $\Rightarrow^*$. Jazyk, který levě rozšířená gramatika Q popisuje je definován jako: $L(Q) = \{v \in T^* : w\#vf, w \in V^*, f \in F\}$.

Levé rozšíření tedy nemá žádný vliv na přijímaný jazyk, na tvar pravidel, ani na sekvence derivačních kroků během přijímání řetězců. Článek [6] sice pracuje v definici $\mathrm{SCG_n}$ výhradně s levě rozšířenou frontovou gramatikou, nicméně levé rozšíření je nutné pouze k důkazu definice. Proto lze definici v následující sekci 2.2 zobecnit na frontové gramatiky bez levého rozšíření. Ze stejných důvodů jsou v celé práci používány frontové gramatiky bez levého rozšíření.

2.1.2 Normální formy

Každou frontovou gramatiku lze převést na ekvivalentní frontovou gramatiku v první normální formě a tu lze vždy převést do druhé normální formy. Pokud je gramatika v druhé normální formě, znamená to, že splňuje i první normální formu [6].

První normální forma

Gramatika je v první normální formě, pokud pravá strana každého pravidla obsahuje pouze neterminály nebo pouze terminály, nesmí obsahovat obojí naráz. Současně gramatika musí pracovat tak, že do určitého bodu zapisuje pouze neterminály a potom zapisuje pouze terminály [6]. Při ručním převodu lze této formy dosáhnout tvorbou nového neterminálu pro každý terminál a následně přidáním stavu na závěr derivace, ve kterém gramatika každý neterminál převede na odpovídající terminál.

Formálně převod probíhá následujícím způsobem, který byl celý převzat z článku [6]. Mějme levě rozšířenou frontovou gramatiku $H = (\bar{V}, T, \bar{W}, \bar{F}, \bar{s}, \bar{R})$, kde $\bar{s} = a_0q_0$, a množiny $\bar{W}' = \{q' : q \in \bar{W}\}$, $\bar{W}'' = \{q'' : q \in \bar{W}\}$, $\bar{V}' = \{a' : a \in V\}$ a $U = \{\langle y, q\rangle : y \in T^* \wedge (a, p, xy, q) \in \bar{R}$, kde $a \in \bar{V}, p \in \bar{W} - \bar{F}, x \in \bar{V}^*\}$. Dále je nutné definovat bijekce α z $\bar{W}$ do $\bar{W}'$ jako $\alpha(q) = q'$, β z $\bar{W}$ do $\bar{W}''$ jako $\beta(q) = q''$ a δ z $\bar{V}$ do $\bar{V}'$ jako $\delta(a) = a'$.

Poté lze definovat frontovou gramatiku v první normální formě $Q = (V, T, W, F, s, R)$, kde 1 a f jsou nové symboly nepatřící do $\bar{V}' \cup T \cup \bar{W}' \cup \bar{W}'' \cup U$ a kde $V = \bar{V}' \cup \{1\} \cup T$, $W = \bar{W}' \cup \bar{W}'' \cup \{f\} \cup U$, $F = \{f\}$ a $s = \delta(a_0)\alpha(q_0)$.

Konstrukce množiny pravidel R probíhá postupem popsaným v následujících krocích.

1. Pokud $(a, p, xy, q) \in \bar{R}$, kde $a \in \bar{V}$, $p \in \bar{W} - \bar{F}$, $x, y \in \bar{V}^*$ a $q \in \bar{W}$, 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;

2. Pokud $(a, p, xy, q) \in \bar{R}$, kde $a \in \bar{V}$, $p \in \bar{W} - \bar{F}$, $x \in \bar{V}^*$, $y \in T^*$, $q \in \bar{W}$ 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;

3. Pokud $(a, p, x, q) \in \bar{R}$, kde $a \in \bar{V}$, $p \in \bar{W} - \bar{F}$, $x \in T^*$ a $q \in \bar{W}$, pak $(\delta(a), \beta(p), x, \beta(q)$ patří do R;

4. Pokud $(a, p, x, q) \in \bar{R}$, kde $a \in \bar{V}$, $p \in \bar{W} - \bar{F}$, $x \in T^*$ a $q \in \bar{F}$, pak $(\delta(a), \beta(p), x, f)$ patří do R.

Druhá normální forma

Gramatika je v druhé normální formě, jestliže je v první normální formě a platí, že pokud libovolná dvě pravidla mají stejný výchozí stav, pak musí mít i stejný čtený symbol [6].

Následující definice byla převzata z článku [6]. Mějme frontovou gramatiku, $H = (V, T, \bar{W}, F, \bar{s}, \bar{R})$ v první normální formě, kde $\bar{s} = a_0 q_0$, a $W = \{\langle q, a\rangle : q \in \bar{W}, a \in V - T\} \cup F$ a $s = a_0\langle q_0, a_0\rangle$. Poté lze definovat frontovou gramatiku, $Q = (V, T, W, F, s, R)$ ve druhé normální formě, kde konstrukce množiny pravidel R probíhá následovně.

1. Pokud $(a, py, q) \in \bar{R}$, kde $a \in V - T$, $p, q \in \bar{W} - F$ a $y \in V^*$, pak $(a, \langle p, a\rangle, y, \langle q, b\rangle)$ patří do R pro všechna $b \in V - T$;
2. Pokud $(a, p, y, q) \in \bar{R}$, kde $a \in V - T$, $p \in \bar{W} - F$, $y \in T^*$ a $q \in F$, pak $(a, \langle p, a\rangle, y, q)$ patří do R.

2.2 Gramatiky s rozptýleným kontextem

Každé pravidlo gramatiky s rozptýleným kontextem nahrazuje několik výskytů obecně různých neterminálů ve větné formě za stejný počet odpovídajících řetězců. Pozice neterminálů ve větné formě není pravidlem gramatiky nijak specifikována, tudíž neterminály mohou být na libovolných pozicích a každý řetězec se vždy vloží na pozici odpovídajícího neterminálu [2][6].

Formálně je gramatika s rozptýleným kontextem čtveřice, $H = (N, T, P, S)$, kde N je abecedou neterminálů, T abecedou terminálů, P představuje konečnou množinu pravidel a S je počáteční symbol gramatiky.

Množina pravidel ve tvaru $(A_1, A_2, \ldots, A_n) \to (x_1, x_2, \ldots, x_n)$ je formálně popsána jako $P \subseteq \bigcup_{m=1}^{\infty} N^m \times ((N \cup T)^*)^m$, kde $A_i \in N$ a $x_i \in (N \cup T)^*$, pro $0 \leq i \leq n$ a $n \geq 1$. Pokud máme řetězce $u = u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1}$ a $v = u_1 x_1 u_2 x_2 \ldots u_n x_n u_{n+1}$, kde $u_i \in (N \cup T)^*$, pro $0 \leq i \leq n + 1$, pak lze v Q pomocí pravidla $p = (A_1, \ldots, A_n) \to (x_1, \ldots, x_n)$, kde $p \in P$, provést derivační krok $u \Rightarrow v[p]$, což lze opět zjednodušit na tvar $u \Rightarrow v$ a běžným způsobem rozšířit na $\Rightarrow^n$, $\Rightarrow^+$ a $\Rightarrow^*$. Pravidla gramatiky se dělí na bezkontextová, kde $n = 1$, a kontextová, kde $n \leq 2$. Jazyk, který gramatika H popisuje je definován jako: $L(H) = \{w \in T^* : S \Rightarrow^* w\}$. Tato definice byla převzata z knihy [2].

2.2.1 Odvození z frontové gramatiky

Každou frontovou gramatiku je možné převést na ekvivalentní gramatiku s rozptýleným kontextem. Frontovou gramatiku je nejdříve nutné převést do druhé normální formy, jak bylo popsáno výše. Vzniklá $\mathrm{SCG_n}$ má omezenou popisnou složitost a neterminály gramatiky mají velmi specifický tvar odvozený ze symbolů a stavů frontové gramatiky [6].

Následující definice i algoritmus byly převzaty z článku [6]. Mějme frontovou gramatiku, $Q = (V, T, W, F, s, R)$ ve druhé normální formě. Před samotnou konverzí je nutné definovat nové tři symboly, $S, \bar{L}, \bar{R}$ mimo abecedu $V \cup W \cup N \cup M$ a abecedy: $U = \{\langle p, i\rangle : p \in W - F, i \in \{1, 2\}\} \cup \{S, \bar{L}, \bar{R}\}$, $N = \{\lceil \underline{a}, q\rceil : a \in V - T, q \in W - F\}$, $M = \{\lfloor a, \underline{q}\rfloor : a \in V - T, q \in W - F\}$. Poté lze definovat gramatiku s rozptýleným kontextem

$H = (U \cup N \cup M, T, P, S)$, kde množina pravidel $P = P_1 \cup P_2 \cup P_3$ je sestrojena následujícím způsobem převzatým z článku [6].

Konstrukce pravidel P_1 probíhá procesem popsaným v následujících pěti krocích.

1. Pokud $a_0 q_0 = s$, kde $a_0 \in V - T$ a $q_0 \in W - F$, pak $S \to \lceil \underline{a_0}, q \rceil \langle q_0, 1 \rangle \lfloor a, \underline{q_0} \rfloor$ patří do P_1 pro všechna $q \in W - F$ a všechna $a \in V - T$;

2. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $p, q \in W - F$, $y \in (V - T)^*$ a $y = a_1 a_2 \cdots a_{|y|}$, pak $\langle q, 1 \rangle \to \lceil \underline{a_1}, q_1 \rceil \lceil \underline{a_2}, q_2 \rceil \cdots \lceil \underline{a_{|y|}}, q_{|y|} \rceil \langle p, 1 \rangle \lfloor b, \underline{p} \rfloor$, kde $q_1 = p$, patří do P_1 pro všechna $q_2 q_3, \ldots, q_{|y|} \in W - F$ a všechna $b \in V - T$;

3. Pro každé $q \in W - F$ patří $\langle q, 1 \rangle \to \bar{L} \langle q, 2 \rangle$ do P_1;

4. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $p, q \in W - F$ a $y \in T^*$, pak $\langle q, 2 \rangle \to y \langle p, 2 \rangle \lfloor b, \underline{p} \rfloor$ patří do P_1 pro všechna $b \in V - T$;

5. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $q \in W - F$, $y \in T^*$ a $p \in F$, pak $\langle q, 2 \rangle \to y \bar{R}$ patří do P_1.

Na závěr je vytvořena množina kontextových pravidel

$$P_2 = \left\{ \left(\lceil \underline{a}, q \rceil, \bar{L}, \bar{R}, \lfloor a, \underline{q} \rfloor \right) \to \left(\bar{L}, \varepsilon, \varepsilon, \bar{R} \right) : a \in V - T, q \in W - T \right\}$$

a pravidla pro smazání speciálních symbolů

$$P_3 = \{ \bar{L} \to \varepsilon, \bar{R} \to \varepsilon \}.$$

Z procesu konstrukce můžeme vyvodit obecný tvar bezkontextového pravidla:

$$A \to r_1 F_1 t C F_2 r_2,$$

kde $r_1 \in N^*$, $r_2 \in M^*$, $F_1 \in \{\varepsilon, \bar{L}\}$, $F_2 \in \{\varepsilon, \bar{R}\}$, $t \in T^*$, $A \in U - \{\bar{L}, \bar{R}\}$ a $C = \varepsilon \ \vee \ C \in U - \{\bar{L}, \bar{R}\}$.

Symboly ve tvaru $\lceil a, q \rceil$ budou dále označovány jako *L-symboly*, symboly ve tvaru $\lfloor a, q \rfloor$ budou dále označovány jako *R-symboly*, kde $a \in V - T$ a $q \in W - F$.

2.2.2 Derivace řetězce v $\mathrm{SCG_n}$

Derivaci řetězce v $\mathrm{SCG_n}$ je prováděna ve dvou fázích, jako je vidět na příkladu 2.1. Na příkladu lze také vidět, že L-symboly značí všechny neterminály frontové gramatiky zapsané do fronty během derivace řetězce ve frontové gramatiky. Oproti tomu R-symboly značí všechny neterminály frontové gramatiky, které byly v korespondujícím stavu čteny z fronty. Odpovídá i pořadí derivačních kroků, $\mathrm{SCG_n}$ pouze provádí první krok navíc, aby simulovala počáteční symbol a počáteční stav.

Například v prvním derivačním kroku frontové gramatiky, v příkladu 2.1, bylo aplikováno pravidlo (S, s, XS, p), které zapsalo do fronty dva neterminály X a S. Po aplikaci tohoto pravidla se frontová gramatika nachází ve stavu p. Proto bylo ve druhém derivačním kroku $\mathrm{SCG_n}$ aplikováno pravidlo, které přidává L-symboly $\lceil \underline{X}, p \rceil$ a $\lceil \underline{S}, q \rceil$. Protože se frontová gramatika nachází ve stavu p a na začátku fronty je neterminál X, čtení toho symbolu rovnou simulováno vložením R-symbolu $\lfloor X, p \rfloor$.

Příklad 2.1. Mějme frontovou gramatiku v 2. normální formě $Q = (V, T, W, \{f\}, Ss, R)$, kde $V = \{S, X, a, b\}$, $T = \{a, b\}$, $W = \{s, p, q, f\}$ a $R = \{(S, s, XS, p), (X, p, aa, q), (S, q, bb, f)\}$.

Podle konstrukce v podsekci 2.2.1 vytvoříme gramatiku $H = (K, T, P, S_H)$. Z množiny pravidel P jsou dále vypsána pouze pravidla nutná k derivaci řetězce $aabb$. Pravidla jsou rozdělena podle kroků, ve kterých byla během konstrukce vytvořena:

1. $S_H \to \lceil \underline{S}, s \rceil \langle s, 1 \rangle \lfloor S, \underline{s} \rfloor$, vytvořené z počátečního symbolu a počátečního stavu Ss;
2. $\langle s, 1 \rangle \to \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \langle p, 1 \rangle \lfloor X, \underline{p} \rfloor$, vytvořené z pravidla $(S, s, XS, p) \in R$;
3. $\langle p, 1 \rangle \to \bar{L} \langle p, 2 \rangle$, vytvořené na základě stavu p;
4. $\langle p, 2 \rangle \to aa \langle q, 2 \rangle \lfloor S, \underline{q} \rfloor$, vytvořené z pravidla $(X, p, aa, q) \in R$;
5. $\langle q, 2 \rangle \to bb\bar{R}$ vytvořené z pravidla $(S, q, bb, f) \in R$.

Derivace řetězce $aabb$ v gramatice Q vypadá následovně:

$$Ss \Rightarrow XSp \Rightarrow Saaq \Rightarrow aabbf.$$

Derivace řetězce $aabb$ v gramatice H potom simuluje derivaci v gramatice Q ve dvou fázích.

V první fázi jsou aplikována pouze bezkontextová pravidla:

$$\begin{array}{lrcl}
 & & S_H & \\
\Rightarrow & \lceil \underline{S}, s \rceil & \langle s, 1 \rangle & \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil & \langle p, 1 \rangle & \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \bar{L} & \langle p, 2 \rangle & \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \bar{L} & aa\langle q, 2 \rangle & \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \bar{L} & aabb & \bar{R} \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor
\end{array}$$

V druhé fázi jsou pomocí kontextového pravidla odstraňovány neterminály:

$$\begin{array}{lrcl}
 & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \lceil \underline{S}, q \rceil \bar{L} & aabb & \bar{R} \lfloor S, \underline{q} \rfloor \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \lceil \underline{X}, p \rceil \bar{L} & aabb & \bar{R} \lfloor X, \underline{p} \rfloor \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \lceil \underline{S}, s \rceil \bar{L} & aabb & \bar{R} \lfloor S, \underline{s} \rfloor \\
\Rightarrow & \bar{L} & aabb & \bar{R} \\
\Rightarrow^2 & & aabb &
\end{array}$$

Tento příklad byl převzat z článku [6].

První fáze

Během první fáze derivace jsou používána pouze bezkontextová pravidla gramatiky (z množiny P_1), která vždy přepisují jediný symbol reprezentující stav gramatiky (dále jen *stav*) ve tvaru $\langle q, i \rangle$, kde q je stav původní frontové gramatiky a $i \in \{1, 2\}$.

Jak je patrné z podsekce 2.2.1, pravidla frontové gramatiky zapisující neterminály jsou převedena na pravidla se stavy označenými číslicí 1, zatímco pravidla zapisující terminály jsou převedena na pravidla se stavy s číslicí 2. Z toho plyne, že během derivace řetězce v $\mathrm{SCG_n}$ jsou rovněž od určitého bodu aplikována výhradně pravidla zapisující terminály, popřípadě prázdné řetězce na místě terminálů. Tímto bodem je pak přidání speciálního symbolu $\bar{L}$, neboť pouze pravidla obsahující speciální symbol $\bar{L}$ mohou změnit stav označený číslicí 1 na ten označený číslicí 2.

Druhá fáze

Druhá fáze derivace (dále jen *eliminace*) probíhá opakovanou aplikací kontextových pravidel následovaných pravidly pro smazání speciálních symbolů $\bar{L}$ a $\bar{R}$ (z množin P_2 a P_3). Tato pravidla lze aplikovat pouze pokud se ve větné formě nachází oba symboly $\bar{L}$ a $\bar{R}$.

Kontextové pravidlo vždy odstraní odpovídající L-symbol a R-symbol, každý na jedné straně větné formy a na jejich místo posune symboly $\bar{L}$ a $\bar{R}$. Proto žádné kontextové pravidlo nemůže přepsat neterminály mezi symboly $\bar{L}$ a $\bar{R}$. Takto je zajištěno, že kontextové pravidlo vždy přepíše pouze neterminály sousedící se speciálními symboly, protože jinak by výsledkem kroku byla větná forma, kde se nejméně jeden neterminál nachází mezi symboly $\bar{L}$ a $\bar{R}$, a nemůže být přepsán. Eliminace tedy ověřuje, zda pořadí L-symbolů ve větné formě odpovídá pořadí R-symbolů ve větné formě.

2.2.3 Jedno kontextové pravidlo

Celá následující podsekce je převzata z článku [6]. Gramatiky s rozptýleným kontextem s jedním kontextovým pravidlem ukazují, že je možné zredukovat počet kontextových pravidel a stále zachovat výpočetní sílu Turingova stroje. SCG_1 pracují stejně jako SCG_n a jsou tedy ekvivalentní. Rozdílem je kódování neterminálů gramatiky tak, aby jedno kontextové pravidlo bylo schopné provádět redukci všech neterminálů. Tato změna však nemá vliv na fungování gramatiky a postup, kterým jsou derivovány řetězce, zůstává taktéž stejný.

Kódování neterminálů

V této kapitole je popsán pouze jeden konkrétní způsob kódování neterminálů v SCG_1, který bude používán v této práci, definice obecného kódování je uvedena v článku [6].

Mějme původní frontovou gramatiku $Q = (V, T, W, F, s, R)$, která byla konvertována na SCG_n. Pro každý neterminál ve tvaru $\lceil \underline{a}, q \rceil$, kde $a \in V - T$ a $q \in W - F$, musí existovat unikátní kód v podobě řetězce z množiny $\{103\}^m\{1\}\{10\}^n$, kde $m, n \geq 1$ tak, aby měly kódy pro všechny možné takovéto neterminály stejný počet znaků 1.

Aby toto bylo zaručeno, musí platit, že

$$m + n = |(V - T) \times (W - F)| + 1.$$

Tím pádem pro každou možnou kombinaci neterminálu a nekoncového stavu původní frontové gramatiky, a tím i pro každý neterminál ve tvaru $\lceil \underline{a}, q \rceil$, existuje unikátní kombinace m a n, kde $m, n \geq 1$, pomocí které lze jednoznačně vytvořit kód ve výše zmíněném tvaru.

Tento způsob kódování je označen jako bijekce ι, kde

$$\lceil \underline{a}, q \rceil \sim \iota(aq) = (103)^m 1 (10)^n,$$

kde každá unikátní kombinace a, q jednoznačně určuje unikátní kombinaci m, n, která splňuje výše zmíněné podmínky.

Bijekce κ, která kóduje neterminály ve tvaru $\lfloor a, \underline{q} \rfloor$, je potom odvozena od bijekce ι následujícím způsobem. Mějme $\iota(aq) = u11v$, kde $a \in V - T$ a $q \in W - F$, potom platí $\kappa(aq) = z10301w$. Řetězec z je řetězec v pozpátku, ve kterém je následně každý symbol 0 nahrazen řetězcem 30. Řetězec w je řetězec u, z kterého jsou nejprve odstraněny všechny symboly 3 a následně je obráceno jeho pořadí. Platí tedy následující implikace

$$\iota(aq) = (103)^m 1 (10)^n \Rightarrow \kappa(aq) = (301)^n 0301 (01)^m.$$

Definice gramatiky

Mějme $\mathrm{SCG_n}$ $H = (U \cup N \cup M, T, P_1 \cup P_2 \cup P_3, S)$ sestrojenou způsobem popsaným v podsekci 2.2.1. Potom lze definovat $\mathrm{SCG_1}$ $G = (U \cup D, T, P)$, kde $P = P' \cup P''$.

Konstrukce množiny pravidel P' probíhá procesem popsaným v následujících pěti krocích.

1. Pro každé pravidlo z P_1 ve tvaru $S \to \lceil \underline{a_0}, q \rceil \langle q_0, 1 \rangle \lfloor a, \underline{q_0} \rfloor$,
   kde $a_0, a \in V - T$ a $q_0, q \in W - F$,
   patří $S \to 1\iota(a_0 q)\langle q_0, 1\rangle\kappa(aq_0)1$ do P';

2. Pro každé pravidlo z P_1 ve tvaru $\langle q, 1\rangle \to \lceil \underline{a_1}, q_1 \rceil \lceil \underline{a_2}, q_2 \rceil \cdots \lceil \underline{a_m}, q_m \rceil \langle p_1, 1\rangle \lfloor b, \underline{p} \rfloor$,
   kde $a_1, a_2, \ldots, a_m, b \in V - T$ a $q_1, q_2, \ldots \in W - F$, $m \geq 0$,
   patří $\langle q, 1\rangle \to \iota(a_1 q_1)\iota(a_2 q_2) \cdots \iota(a_m q_m)\langle p, 1\rangle\kappa(bp)$ do P';

3. Pro každé pravidlo z P_1 ve tvaru $\langle q, 1\rangle \to \bar{L}\langle q, 2\rangle$, kde $q \in W - F$,
   patří $\langle q, 1\rangle \to 20\langle q, 2\rangle$ do P';

4. Pro každé pravidlo z P_1 ve tvaru $\langle q, 2\rangle \to y\langle p, 2\rangle \lfloor b, \underline{p} \rfloor$,
   kde $b \in V - T$, $q, p \in W - F$ a $y \in T^*$,
   patří $\langle q, 2\rangle \to y\langle p, 2\rangle\kappa(bp)$ do P';

5. Pro každé pravidlo z P_1 ve tvaru $\langle q, 2\rangle \to y\bar{R}$, kde $q \in W - F$ a $y \in T^*$,
   patří $\langle q, 2\rangle \to y302$ do P'.

Množina pravidel P'' pak vypadá následovně.

$$P'' = \{(1, 2, 0, 3, 0, 2, 1) \to (2, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, 2), 2 \to \varepsilon\}$$

Eliminace pomocí kontextového pravidla

Eliminace dvou odpovídajících zakódovaných symbolů ve větné formě je ukázána na příkladu 2.2. Stejným způsobem gramatika provede i eliminaci větné formy o libovolném počtu neterminálů.

Příklad 2.2. Mějme větnou formu

$$\lceil \underline{S}, s \rceil \bar{L} y \bar{R} \lfloor S, \underline{s} \rfloor,$$

kde $y \in T^*$ a současně $\iota(Ss) = 103103110$ a tedy $\kappa(Ss) = 30103010101$. Potom zakódovaná větná forma bude mít následující tvar, kde mezery slouží pouze ke zvýšení přehlednosti a oddělují jednotlivé symboly:

$$1 \quad 103103110 \quad 20y302 \quad 30103010101 \quad 1.$$

Na takto zakódované větné formě potom bude pomocí kontextového pravidla eliminace provedena následovně. Podtržení označuje symboly, které budou přepsány kontextovým pravidlem.

$$\begin{array}{lccccc}
\Rightarrow & 1 & 103103\underline{1}\underline{0} & \underline{20y302} & 30\underline{1}03010101 & 1 \\
\Rightarrow & 1 & 103103\underline{120} & y & \underline{302}03\underline{1}0101 & 1 \\
\Rightarrow & 1 & 10\underline{3}1\underline{0}3\underline{2} & y & \underline{030}2\underline{0}1011 & 1 \\
\Rightarrow & 1 & \underline{1}03\underline{203} & y & \underline{02}011 & 1 \\
\Rightarrow & \underline{1} & \underline{203} & y & \underline{021} & \underline{1} \\
\Rightarrow & \underline{2} & & y & & \underline{2} \\
\Rightarrow^2 & & & y & &
\end{array}$$

Kapitola 3

Demonstrace gramatik

Gramatiky s rozptýleným kontextem s jedním kontextovým pravidlem budou demonstrovány pomocí aplikace, která bude v zadané gramatice derivovat zadaný řetězec a celý proces prezentovat uživateli. Kódování v těchto gramatikách nicméně způsobuje značnou nepřehlednost až nečitelnost celého procesu, a proto bude aplikace primárně demonstrovat derivaci v ekvivalentní SCG_n, která derivuje řetězce identickým způsobem (viz podsekci 2.2.3). Aplikace bude nabízet i možnost derivovat řetězec přímo v SCG_1 se zakódovanými neterminály.

Tato kapitola tedy popisuje návrh a implementaci této aplikace, zatímco konkrétní algoritmy, které aplikace používá pro derivaci řetězců a převody gramatik jsou popsány v kapitole 4.

3.1 Návrh aplikace

Aplikace demonstruje gramatiky prostřednictvím derivace řetězce v SCG_n a prezentací celého postupu při derivaci uživateli. Mimo využití pro demonstraci gramatik musí aplikace sloužit i jako nástroj k testování heuristiky.

Kvůli složitosti odvození SCG_n z frontové gramatiky může uživatel jen těžko zadat přímo validní gramatiku. Proto uživatel zadává frontovou gramatiku, která bude převedena na SCG_n a v té bude následně derivován zadaný řetězec. Aplikace umožňuje i algoritmický převod frontové gramatiky do druhé normální formy, díky čemuž může uživatel zadat libovolnou frontovou gramatiku.

Nedeterminismus při derivaci řetězce lze pro účely demonstrace řešit prohledáváním stavového prostoru s použitím heuristik pro usnadnění celého procesu. Při práci se složitějšími gramatikami a řetězci by tento přístup mohl být problémem kvůli příliš velkému stavovému prostoru. Tento problém částečně řeší heuristika v sekci 4.3, nicméně se neočekává, že by aplikace dokázala sama zpracovat libovolně složitý řetězec v libovolně složité SCG_n.

Demonstrace derivace řetězce v SCG_1 se od derivace v SCG_n liší pouze v jiné reprezentaci neterminálů. Tato funkce je tedy řešena až při samotném tisku a nijak nezasahuje do interního chování aplikace.

3.1.1 Uživatelské rozhraní

Základním uživatelským rozhraním aplikace je příkazová řádka. Její nespornou výhodou je možnost automatizace vstupů od uživatele, což se hodí pro testování kvality heuristiky navržené v sekci 4.3. Nevýhodou jsou však omezené možnosti prezentace výsledků a vizualizace

postupu. Toto lze částečně eliminovat využitím formátu HTML společně s kaskádovými styly a dynamickými prvky. Nicméně i přesto se tento způsob prezentace může snadno stát nepřehledným a aplikace by se proto dala rozumně rozšířit grafickým uživatelským rozhraním zobrazujícím data ve stromové struktuře.

Při práci s příkazovou řádkou je výhodou, pokud aplikace v základním nastavení čte ze standardního vstupu a tiskne na standardní výstup, hlavně kvůli využití v kombinaci s *unixovým*i rourami. Mimo to aplikace podporuje i čtení a zápis do souboru pro větší pohodlí uživatele. Nastavení aplikace bude prováděno při spuštění pomocí přepínačů způsobem běžným *unixové* operační systémy.

3.1.2 Vstupy

Aplikace pracuje se dvěma vstupy, a sice frontovou gramatikou a řetězcem, který má být v gramatice derivován. Řetězec je předáván argumentem příkazové řádky, zatímco vstupní gramatika je čte, buď ze standardního vstupu nebo ze specifikovaného souboru.

Vstupní gramatika je zadávána ve formátu JSON, protože je poměrně rozšířený, snadno čitelný člověkem a nevyžaduje tak velké množství znaků vyjadřujících strukturu dat, jako například formát XML, takže uživatel nemusí psát tolik znaků navíc [3]. Formát vstupní frontové gramatiky by se navíc měl co nejvíce podobat matematické definici této gramatiky, protože většina uživatelů bude pravděpodobně obeznámena právě s tímto způsobem zadání frontové gramatiky.

Zápis gramatiky ve formátu JSON

Jak je vidět na příkladu 3.1, gramatika je předávána jako jediný objekt, což umožní při čtení ignorovat zbytek souboru po konci prvního objektu. Tím se zbylý prostor uvolní případným poznámkám a komentářům, které v základním formátu jinak nejsou definovány, a způsobily by tak chybu při zpracování dat [3].

Množiny symbolů, terminálů, stavů a koncových stavů jsou zadávány v podobě pole s řetězci jako položkami. Jelikož je aplikace určena pro práci s jednoduchými gramatikami, je výhodné, aby symbol představovat pouze jediný znak anglické abecedy, což eliminuje jakoukoli nejednoznačnost v řetězci symbolů a usnadňuje reprezentaci uvnitř aplikace. Stavy se nevyskytují v řetězcích, a i jednoduché frontové gramatiky mohou mít po převodu do druhé normální formy poměrně velké množství stavů, proto mohou mít navíc oproti symbolům i tvar jednoho písmena anglické abecedy následovaného jednou číslicí.

Množina pravidel gramatiky je opět reprezentována ve tvaru pole. Pravidlo gramatiky je rovněž reprezentováno polem i přes to, že by se správně mělo jednat o objekt se čtyřmi pojmenovanými položkami. Důvodem je právě snaha, aby byl zápis ve formátu JSON podobný běžnému matematickému zápisu gramatiky. Navíc tak uživatel není nucen opakovaně psát v každém pravidle názvy položek. Tento přístup je umožněn faktem, že jak symboly, tak stavy jsou zadávány ve formě řetězců, nicméně všechny položky musí samozřejmě být validní a ve správném pořadí.

Příklad 3.1. Například gramatika $Q = \{V, T, W, \{f\}, As_0, R\}$, kde $V = \{A, a\}$, $T = \{a\}$, $W = \{s_0, f\}$ a $R = \{(A, s_0, aA, s_0), (A, s_0, \varepsilon, f), (a, s_0, a, s_0)\}$, bude vyjádřena zapsána ve tvaru popsaném ve výpisu 3.1.

```
{
    "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"]
    ]
}
```

Výpis 3.1: Příklad vstupu ve formátu JSON

3.1.3 Výstup

Výstupem aplikace je celý proces derivace řetězce, který je tištěn na standardní výstup nebo do specifikovaného souboru. Derivace řetězce je prezentována ve formátu podobném formálnímu zápisu, a sice v podobě derivačních kroků mezi větnými formami. U každého derivačního kroku by mělo být vypsáno i příslušné pravidlo.

Aplikace nabízí jak výstup ve formátu HTML pro větší přehlednost, tak výstup ve podobě čistého textu pro případnou práci s výsledky přímo v příkazové řádce. Současně je pro zápis gramatik vhodné využít speciálních symbolů v kódování unicode pro větší přehlednost. Kódování unicode však může při výpisu způsobovat problémy a aplikace tak musí nabízet i alternativní výstup pouze v symbolech tabulky ASCII.

3.1.4 Heuristiky

Jak již bylo zmíněno, aplikace využívá při derivaci řetězce heuristiky. Základní používanou heuristikou je heuristika navržená v sekci 4.3, nicméně aplikace musí umožňovat i derivaci s použitím alternativních heuristik.

Speciálním případem alternativní heuristiky je vstup od uživatele, kdy je možné vybrat, které pravidlo gramatiky se má v dalším kroku použít. Tento postup umožňuje uživateli kontrolovat prohledávání stavového prostoru a tím pádem teoreticky i zpracování libovolně složité gramatiky nebo řetězce.

Protože aplikace bude využívána i k testování kvality navržené heuristiky, je nutné, aby bylo výsledky testování s čím porovnat. Proto musí aplikace, mimo heuristiky navržené v sekci 4.3 a výše zmíněného vstupu od uživatele, tedy ještě umožňovat derivaci pomocí jednoduché heuristiky. Ta bude sloužit jako základ, se kterým bude navržená heuristika srovnávána. Díky tomu bude možné při testování porovnat kvalitu navržené heuristiky. Na závěr aplikace tiskne i jednoduché statistiky ohledně úspěšnosti prohledávání stavového prostoru.

3.1.5 Normalizace

Aplikace by měla ověřovat, jestli je vstupní gramatika ve druhé normální formě a současně nabízet možnost algoritmicky vstupní gramatiku normalizovat. Pokud však vstupní gramatika nesplňuje druhou normální formu, normalizace není provedena automaticky, ale uživatel je pouze varován ohledně této skutečnosti. Tento přístup ponechává uživateli větší volnost při práci s aplikací.

Algoritmická normalizace zavádí velké množství nových pravidel a stavů, a proto může být její použití problematické pro složitější gramatiky, které tak budou muset být normalizovány ručně uživatelem. Stavy vytvořené při algoritmické normalizaci navíc mají velmi složitý tvar, protože dojde k vnoření obou složených stavů popsaných v podsekci 2.1.2. Po následném převodu na $\mathrm{SCG_n}$ mohou mít symboly reprezentující stavy tvar $\langle\langle\langle a, p\rangle, A\rangle, 1\rangle$, což může vést k výraznému snížení přehlednosti. Proto by aplikace měla umožňovat přejmenování stavů po normalizaci na jednoduchá písmena anglické abecedy, případně číslice.

3.1.6 Ověření správnosti derivace řetězce

Po derivaci řetězce je nutné ověřit, že byl řetězec zpracován správně. Tedy, že řetězce přijaté frontovou gramatikou byly přijaty i $\mathrm{SCG_n}$ a řetězce odmítnuté frontovou gramatikou byly odmítnuty i $\mathrm{SCG_n}$. Jedná se o jednoduchý způsob jakým ověřit, zda je v pořádku implementace jednotlivých algoritmů a současně zda navržená heuristika funguje správně.

Je tedy nutné, aby aplikace zpracovávala řetězec i v příslušné frontové gramatice. Tuto funkcionalitu by sice mohl během testování zajišťovat externí nástroj, ale při implementaci přímo v aplikaci je možné využít již zavedených struktur.

3.1.7 Reprezentace dat

Pro implementaci aplikace je nutné vhodně navrhnout vnitřní reprezentaci gramatik, se kterými aplikace pracuje.

Symboly

Daty na nejnižší úrovni jsou pro aplikaci symboly, které, jak bylo zmíněno výše v podsekci 3.1.2, představují jediný znak. Pro potřeby algoritmu normalizace (viz podsekci 2.1.2) je nutné, aby všechny symboly mohly být označovány čarou nebo dvěma čarami.

Reprezentace symbolu je použita i při reprezentaci stavů, které však mohou být zadávány i ve tvaru písmena anglické abecedy následovaného jedinou číslicí. Stavy v tomto tvaru musí být opět možné označit čarou, a proto je výhodné použít i pro ně strukturu reprezentující symbol.

Symboly jsou tedy reprezentovány strukturou obsahující znak anglické abecedy následovaný volitelnou číslicí a volitelným označením jednou nebo dvěma čarami.

Stavy

Stavy v obou gramatikách mohou nabývat mnoha různých tvarů, které je nutné do sebe navzájem během normalizace a konverze obalovat. Například ze stavu p může po normalizaci vzniknout stav $\langle p, a\rangle$, kde a je nějaký neterminál, který může být při konverzi převeden na tvar $\langle\langle p, a\rangle, 1\rangle$. Normalizace je ale volitelný krok a během konverze tak může být převáděn pouze stav p na stav ve tvaru $\langle p, 1\rangle$. Z toho důvodu musí aplikace pracovat s každým stavem stejným způsobem nezávisle na jeho tvaru.

Stav tedy může nabývat jednoho z následujících tvarů:

1. Tvar řetězce znaků, který může být reprezentován výše zmíněnou strukturou představující symbol;
2. Tvar $\langle y, p\rangle$, kde y je řetězec symbolů a p je stav v libovolném tvaru;
3. Tvar $\langle p, a\rangle$, kde a je symbol a p je stav v libovolném tvaru;

4. Tvar $\langle p, i \rangle$, kde $i \in \{1, 2\}$ a p je stav v libovolném tvaru.

Obdobným rekurzivním způsobem lze reprezentovat libovolný stav i v aplikaci pomocí struktur.

Podobně jako symboly, i stavy musí být možné během transformace do první normální formy označovat čarou. Vstupní stavy však mohou být pouze v prvním tvaru, a proto stačí využít této možnosti ve struktuře reprezentující symbol.

Frontová gramatika

Vstupní frontová gramatika by měla být celá ukládána do jedné datové struktury, která bude součástí vstupního rozhraní jádra aplikace. Díky tomu je možné na základě vstupů od uživatele vyplnit tuto strukturu a předat data do programu v ucelené formě. Součástí reprezentace frontové gramatiky musí být i reprezentace pravidel. I přes jejich jednoduchost a možnost reprezentace pouhým řetězcem znaků je výrazně lepší použít opět datovou strukturu. Výhodami by měl být přehlednější kód a snazší následná práce s pravidly v programu.

Součástí reprezentace frontové gramatiky musí být i reprezentace abecedy a množiny stavů. Každá abeceda nebo množina stavů by měla být rovněž reprezentována jedinou strukturou tak, aby nad ní bylo možné snadno provádět běžné operace, jako například iteraci nebo dotaz, zda je v ní symbol obsažen.

Gramatika s rozptýleným kontextem

Základem celé aplikace je $\mathrm{SCG_n}$, takže je vhodné pro ni mít ucelenou reprezentaci pomocí jedné struktury podobně jako pro frontovou gramatiku tak, aby s ní bylo možné snadno pracovat. Abecedy terminálů, neterminálů a stavů frontové gramatiky budou v aplikaci nutné pouze pro její konverzi na $\mathrm{SCG_n}$ a tím pádem tvorbu pravidel. Na konci tohoto procesu tedy neexistuje žádný symbol nové gramatiky, který by nebyl použit v alespoň jednom z vytvořených pravidel (viz podsekci 2.2.1), dále již bude program pracovat pouze s pravidly $\mathrm{SCG_n}$. Neterminály této gramatiky jsou přímo odvozeny ze symbolů a pravidel frontové gramatiky a abeceda terminálů je shodná s tou patřící frontové gramatice. Z těchto důvodu není nutné mít ve struktuře redundantní data v podobě reprezentace přímo abecedy terminálů nebo abecedy neterminálů, stačí pouze množina pravidel.

Větná forma

Neterminály $\mathrm{SCG_n}$ mohou mít tvar $\langle p, i \rangle$, $\lceil \underline{a}, q \rceil$, $\lfloor a, \underline{q} \rfloor$ nebo tvar speciálního symbolu S, $\bar{L}$, $\bar{R}$, kde p a q jsou nekoncové stavy frontové gramatiky, a je neterminál frontové gramatiky a $i \in \{1, 2\}$, a lze je rozdělit do tří skupin:

1. Neterminály ve tvarech $\langle p, i \rangle$ a S se vyskytují vždy na levé straně pravidla, ve větné formě se vyskytují vždy nanejvýš jednou a představují stav frontové gramatiky. V aplikaci je tedy žádoucí mít jednotnou strukturu, kterou bude možné vyjádřit oba tyto tvary a pracovat s nimi jednotným způsobem;

2. Neterminály ve tvaru $\lceil \underline{a}, q \rceil$ nebo $\lfloor a, \underline{q} \rfloor$, jejichž řetězce se vyskytují na začátku a na konci větné formy nebo pravé strany pravidla. Slouží pro eventuální ověření správnosti postupu a výsledku derivace řetězce pomocí $\mathrm{SCG_n}$. Neterminály jsou rozděleny na pravé a levé, nicméně v aplikaci bude lepší mít jednu strukturu, která bude schopna obsáhnout jak levé, tak pravé symboly. Toto opět zjednoduší práci s touto skupinou neterminálů;

3. Neterminály $\bar{L}$ a $\bar{R}$ jsou velmi specifické svou pozicí ve větné formě, respektive pravé straně pravidla a tím, že se v ní každý z nich může vyskytovat nanejvýš jednou. Z toho důvodu pravděpodobně nebude nutné pro ně vytvářet strukturu, ale bude stačit označit příslušný řetězec příznakem, který říká, zda se v ní daný symbol nachází nebo ne. To samé platí i pro pravou stranu pravidla.

Větná forma SCG_n má, jak bylo naznačeno v podsekci 2.2.1, velmi striktní tvar, $r_1F_1tCF_2r_2$, kde $r_{1,2}$ jsou řetězce neterminálů z druhé skupiny, $F_{1,2}$ jsou neterminály z třetí skupiny, t je řetězec terminálů a C je neterminál z první skupiny. Pro reprezentaci větné formy je tedy možné použít příslušné reprezentace neterminálu a doplnit je o reprezentaci řetězce terminálů. To celé je samozřejmě třeba zabalit do jedné struktury pro následnou práci s větnými formami.

Pravidla

Běžné bezkontextové pravidlo SCG_n z množiny P_1 (viz sekci 2.2) se skládá z levé a pravé strany, obě se dají reprezentovat již výše zmíněnými strukturami. Levá strana je vždy symbol z první skupiny, zatímco pravá strana má stejný tvar jako větná forma. Pro reprezentaci pravidel tedy stačí pouze obalit reprezentace obou stran do nové struktury tak, aby se s pravidlem dalo pracovat jako s jedním celkem.

Kontextová pravidla nemají v aplikaci vlastní reprezentaci v podobě datové struktury. Tato pravidla jsou svým principem pro každou SCG_n stejná a sice, že vždy eliminují komplementární neterminály na obou stranách větné formy. Pro aplikaci je tedy rozumnější využít této shody a při zpracovávání řetězců odstranit neterminály přímo na to určeným algoritmem, než implementovat složitější algoritmus pro zpracování libovolného kontextového pravidla gramatiky. Kvůli tomu není aplikace ani teoreticky schopné zpracovat libovolnou gramatiku s rozptýleným kontextem, nicméně to ani není její ambicí. Poslední dvě bezkontextová pravidla pro odstranění speciálních symbolů $\bar{L}$, $\bar{R}$ na závěr derivace mohou být implementována obdobně jako pravidla kontextová.

3.1.8 Architektura

Aplikace je rozdělena do modulů. Každý hlavní modul se stará o jednu z následujících funkcí:

- zpracování argumentů z příkazové řádky,
- převod vstupní gramatiky ve formátu JSON na její vnitřní reprezentaci,
- normalizaci frontové gramatiky,
- validaci řetězce ve frontové gramatice,
- odvození SCG_n od frontové gramatiky,
- derivaci řetězce v SCG_n.

Hlavní moduly jsou volány v příslušném pořadí ze vstupního bodu programu a potom používají pomocné moduly.

Mezi pomocné moduly patří hlavně moduly obsahující abstraktní datové typy pro struktury popsané v podsekci 3.1.7. Dále pak čtyři pomocné moduly obsahující:

- funkce pro tisk chybových hlášení a správu návratového kódu,
- implementaci heuristik,
- funkce pro tisk složitějších struktury hlavně pro výstup ve formátu HTML,
- společné univerzální funkce a konstanty používané ve více ostatních modulech.

3.2 Implementace aplikace

Aplikace je implementována v jazyce C, protože tento jazyk nevyžaduje žádné externí závislosti a lze jej snadno kompilovat na téměř jakémkoliv stroji. Některé vyšší jazyky navíc umožňují použití kódu v jazyce C. Například jazyk C++ umožňuje volání kódu v jazyce C.

Celá aplikace je implementována v několika výše popsaných modulech, bez využití globálních proměnných, a to ani proměnných globálních pouze v jednom modulu. Výjimkami jsou proměnné používané při tisku kódovaných neterminálů a proměnná pro návratový kód, jejichž použití bude vysvětleno dále. Tento přístup sice zvyšuje počet parametrů předávaných do jednotlivých funkcí, ale výrazně snižuje pravděpodobnost chyby způsobené tím, že funkce pracuje se špatnými daty.

3.2.1 Proces demonstrace

Pro demonstraci gramatiky musí aplikace provést následující kroky:

1. zpracovat argumenty příkazové řádky,
2. zpracovat vstupní frontovou gramatiku,
3. normalizovat vstupní gramatiky, pokud o to uživatel žádá,
4. pro kontrolu derivovat řetězec v příslušné frontové gramatice,
5. vytvořit $\mathrm{SCG_n}$ odvozenou od příslušné frontové gramatiky,
6. derivovat řetězec v $\mathrm{SCG_n}$.

3.2.2 Abeceda symbolů a množina stavů

Při implementaci abecedy symbolů a množiny stavů je využíváno pole. Abeceda ani množina nemůže být po svém vytvoření editována, což zajišťuje konzistenci symbolů a stavů během práce s gramatikou. To společně s implementací pomocí pole umožňuje používat abecedu nebo množinu jako seřazenou sekvenci prvků. Z toho důvodu je v implementaci možné získat index, na kterém se nachází určitý prvek v dané abecedě nebo množině, přestože se jedná z formálního hlediska o nesmysl.

3.2.3 Chybová hlášení

Pro tisk chybových hlášení bylo vytvořeno několik funkcí odlišujících hlášení podle závažnosti. Všechny tyto funkce využívají seznamu argumentů s proměnnou délkou, tak aby se svým voláním tak napodobují standardní funkci `printf`. Všechna chybová hlášení jsou tištěna na standardní chybový výstup ve tvaru `ZÁVAŽNOST: zpráva`, kde je za každým hlášením automaticky odlomen řádek. Každá funkce implementuje výpis zprávy s jedním stupněm závažnosti, které jsou následující (seřazeny od nejméně závažného):

1. ladící výpis,
2. informace,
3. varování,
4. chyba,
5. fatální chyba.

Ladící výpisy by měly být tištěny pouze, pokud to programátor během ladění vyžaduje. To je implementováno pomocí makra `DEBUG_MODE`, které může být nastaveno při kompilaci kódu. Pokud toto makro není nastaveno, jsou všechny ladící funkce pomocí preprocesoru odstraněny ještě před samotnou kompilací programu.

Informace není nijak negativně ani pozitivně zabarvena. Jedná se o zprávu, která uživateli typicky sděluje, že proběhla nějaká událost. Naproti tomu varování je hlášení o negativní události, která ale nebrání v dalším běhu programu.

Chyby

Chyby jsou vyvolány v případě, že aplikace nemůže pokračovat v běhu programu. Funkce pro tisk chyb mají mimo zprávy navíc parametr s návratovým kódem aplikace. Návratový kód je doplněn do tisknuté zprávy, nyní ve tvaru `ZÁVAŽNOST KÓD: zpráva`, a současně je aplikace ukončena s tímto návratovým kódem.

Rozdíl mezi fatální chybou a chybou je v tom, že chyba ukončí program korektním způsobem. To znamená, že každá funkce postupně vrátí hodnotu reprezentující chybu, uvolní se všechna alokovaná paměť a běh je ukončen zavoláním příkazu `return` z funkce `main`. Předávání návratového kódu skrze návratové hodnoty funkcí by bylo velmi obtížné a nepřehledné, proto je tento problém řešen pomocí globální proměnné pro návratový kód aplikace. Funkce pro tisk chybového hlášení tuto proměnnou nastaví a na konci běhu je vrácena její hodnota.

Fatální chyba na rozdíl od toho ukončuje program okamžitě na místě výskytu chyby. To se provádí voláním funkce `exit` s příslušným návratovým kódem. Kvůli tomu nemusí dojít ke korektnímu uvolnění veškeré alokované paměti.

Fatální chyba v aplikaci typicky nastává při selhání alokace paměti. Aplikace pracuje s velkým množstvím dynamicky alokovaných struktur, které musí být vytvářeny na mnoha různých místech poměrně komplexního algoritmu, a proto by bylo velmi obtížné propagovat chybu při alokaci korektně až na nejvyšší úroveň. Z toho důvodu je ve zdrojovém kódu použita tato potenciálně nebezpečná cesta, která spoléhá na uvolnění paměti operačním systémem.

3.2.4 Kódování neterminálů

Kódování neterminálů SCG_1 je z hlediska implementace pouze jiný formát tisku symbolů SCG_n, které mají tvar $\lceil \underline{a}, q \rceil$ nebo $\lfloor a, \underline{q} \rfloor$, kde $a \in N$ a $q \in Q$. N jsou neterminály frontové gramatiky a Q jsou nekoncové stavy frontové gramatiky.

Jedinou další změnou je pak přidání symbolu 1 na okraje pravidel přepisujících počáteční symbol gramatiky. Protože takováto pravidla nemohou být aplikována během derivace opakovaně, lze tyto symboly implementovat jednoduchou proměnnou ve struktuře reprezentující větnou formu, která říká, zda větná forma tyto symboly již obsahuje či nikoliv.

Výpočet kódu

Při kódováni se využívá faktu, že abeceda neterminálů a množina nekoncových stavů jsou implementovány tak, že zachovávají pořadí svých prvků (viz podsekci 3.1.7). Proto lze jednoznačně určit index prvku v množině neterminálů nebo nekoncových stavů.

Určení kódu příslušného neterminálu je založeno na tom, že je každý kód jednoznačně určen dvojicí hodnot m a n (viz podsekci 2.2.3). Dále platí, že $m+n = d$, kde d je konstanta $d = |N| \cdot |Q| + 1$. Z toho vyplývá, že m, n lze vypočítat z dvojice aq jako

$$m = |Q| \cdot a_{\text{idx}} + q_{\text{idx}} + 1$$
$$n = d - m,$$

kde a_{idx} je index symbolu a v abecedě N a q_{idx} je index stavu q v množině Q.

Globální proměnné

Pro tvorbu správného kódu je tedy nutné znát všechny neterminály a všechny nekoncové stavy frontové gramatiky tak, aby mohla být určena délka kódu a zajištěna unikátnost. Zakódované neterminály se v programu vyskytují jako součásti větších celků a jejich tisk probíhá na mnoha různých místech hlavně během derivace řetězce. Z těchto důvodu by bylo velmi nepraktické předávat každé funkci pro tisk i všechny neterminály a nekoncové stavy frontové gramatiky. V době derivace řetězce je navíc žádoucí, aby byla frontová gramatika uvolněna z paměti, protože se s ní během samotné derivace přímo nepracuje.

Tento problém je vyřešen zavedením proměnných, které jsou globální pouze v modulu starajícím se o tisk. Po ukončení případné normalizace jsou potom tyto proměnné inicializovány na množinu neterminálů a nekoncových stavů příslušné frontové gramatiky. Dále pak může každá funkce pro tisk zakódovaného neterminálu využít tyto proměnné k výběru správného kódu způsobem naznačeným v pod.

3.2.5 Knihovny pro zpracování formátu JSON

Zpracování formátu JSON je poměrně komplexní problém, který ale již pro jazyk C řeší mnoho externích knihoven, a je proto rozumné použít v aplikaci některou z nich. Knihoven pro zpracování formátu JSON v jazyce C je mnoho a nejsou mezi nimi příliš velké rozdíly. Hlavními požadavky na knihovnu proto byly jednoduchá integrace do aplikace, rozumné aplikační rozhraní a kvalitní návod k použití nebo dokumentace. Gramatika zapsaná v tomto formátu obsahuje hlavně struktury typu *array* a *string*, proto je důležité, aby knihovna zvládala alespoň práci s těmito typy. Naopak knihovna nemusí umět formát JSON vytvářet.

json-c

Komplexní knihovna pro tvorbu a čtení formátu JSON[1]. Nabízí přehledné aplikační rozhraní pro práci s knihovnou a poměrně kvalitní dokumentaci. Nevýhodou však je nutnost složitější kompilace knihovny a komplikovanější integrace do aplikace. Vzhledem k tomu, že aplikace potřebuje číst pouze jednoduchá data, je tato knihovna zbytečně složitá.

YAJL

YAJL[2]je opět komplexní knihovna pro zpracování formátu JSON. Oproti json-c nabízí několik výhod jako například nezávislost na reprezentaci, nicméně pro potřeby aplikace

[1] https://github.com/jehiah/json-c

žádná z těchto výhod není důležitá. Způsob použití knihovny také není nikde moc kvalitně popsán.

JSMN

Velmi jednoduchá a kompaktní knihovna vyznačující se efektivní prací s pamětí[3]. Velkou výhodou je i fakt, že se celá knihovna skládá pouze z jediného hlavičkového souboru. Protože je knihovna určena i pro použití ve vestavěných systémech, nemá žádné externí závislosti. Nevýhodou je o něco komplikovanější proces zpracování formátu z pohledu aplikace používající knihovny. Nicméně navíc nabízí větší možnosti přizpůsobení tohoto procesu na míru a dobře funguje při zpracování typu *string* formátu JSON.

Tuto knihovnu jsem nakonec zvolil hlavně kvůli jednoduchosti integrace a kvalitní dokumentaci.

3.2.6 Unicode a HTML

Aplikace implementuje možnost tisku jak v podobě čistého textu, tak v podobě HTML kódu. Při tisku jsou požívány některé znaky v kódování *unicode*, nicméně při tisku čistého textu je možné nahradit znaky v kódování ASCII, protože se předpokládá, že čistý text bude často tištěn do konzole, která nemusí nutně kódování unicode podporovat. V HTML dokumentu je tento problém vyřešen příslušnou značkou, a proto aplikace vždy používá kódování *unicode* ve spojení s HTML.

O tisk komplexnějších konstrukcí se stará speciální modul, zatímco o tisk jednotlivých struktur zajišťují běžné funkce pro tisk v příslušném modulu. Každá funkce pro tisk dostává parametry podle kterých rozhoduje, zda na výstup vložit příslušné HTML značky, či naopak ne a zda použít ASCII nebo *unicode* verzi znaku. Ke generování HTML kódu nepoužívá aplikace žádnou externí knihovnu, ale využívá maker, která nahrazují značky jazyka tak, aby nedocházelo k při tvorbě aplikace k překlepům.

Celý HTML dokument včetně kaskádových stylů a skriptů je tisknut jako jeden celek, aby byl celý výsledek kompaktní. To nenabízí tak snadnou možnost změny kaskádových stylů uživatelem, ale pokud uživatel ovládá kaskádové styly, pak je pravděpodobně i schopen provést případné navázání externího souboru se styly.

Jednotlivé symboly gramatiky jsou v dokumentu barevně odlišeny pomocí kaskádových stylů pro vyšší přehlednost. Všechny symboly označující stavy jsou tedy zobrazeny v oranžové barvě, zatímco neterminály jsou modré. U neterminálů ve větné formě je důležité vědět, které části větné formy jsou na obou stranách komplementární. Z toho důvodu jsou komplementární symboly zobrazeny ve světlejším odstínu modré, zatímco symboly, které nemají na druhé straně větné formy odpovídající dvojici jsou zvýrazněny odstínem tmavším.

V dokumentu jsou skryté postupy jednotlivých eliminací, aby se zvýšila přehlednost a je možné si příslušnou eliminaci zobrazit až stiskem tlačítka. Jednotlivé větné formy jsou vypisovány v pořadí, v jakém je algoritmus prochází. Pokud se algoritmus vrátí do větné formy, na kterou již nějaké pravidlo jednou aplikoval, měla by tato forma být označena stejným indexem jako její předcházející výskyt tak, aby bylo lépe zřetelné do kterého bodu se algoritmus vrátil. Pro podrobný popis formátování výstupu ve formátu HTML viz přílohu A.

[2] https://github.com/lloyd/yajl/
[3] https://zserge.com/jsmn/

Dynamické prvky

Dokument obsahuje dvě funkce v jazyce *JavaScript*. První funkce je pouze obsluha tlačítka, které dynamicky skrývá příslušnou eliminaci, tak aby se zvýšila přehlednost celého procesu. Druhá funkce se volá pouze jednou při načtení stránky a zajišťuje přípravu indexů souhlasných větných, které jsou tvořeny následujícím způsobem:

1. Při tvorbě dokumentu se všechny indexy vytvoří prázdné označené unikátním identifikátorem každé větné formy (Stejné větné formy mají stejný identifikátor);
2. Při načtení stránky jsou postupně procházeny všechny identifikátory;
3. Pokud existuje více než jeden index označený daným identifikátorem, pak mu vyplněno příslušné pořadové číslo;
4. První z indexů je označen novým unikátním identifikátorem na základě svého pořadového čísla;
5. Do zbylých indexů je přidán odkaz na první index.

Kapitola 4

Algoritmy a Heuristika

Aplikace ke svému běhu potřebuje několik algoritmů, z nichž nejdůležitějším je algoritmus pro derivaci řetězce v SCG_n, neboť demonstruje funkci této gramatiky, a to on volá navrženou heuristiku. Dalo by se tedy říci, že celý zbytek aplikace je pouze jakýmsi rozhraním pro tento algoritmus. Ostatní algoritmy popsané v této kapitole jsou pomocné algoritmy pro normalizaci frontové gramatiky, její převod na SCG_n a zpracování řetězce ve frontové gramatice.

Pro řešení problému derivace řetězce v gramatice s rozptýleným kontextem je možné použít analýzu shora dolů založenou na LL-tablulce [4]. SCG_n má ale velmi specifický tvar a způsob derivace řetězců (viz podsekci 2.2.2), a proto není vhodné použít poměrně složitý obecný algoritmus.

4.1 Návrh algoritmů

Díky tomu, že je proces derivace řetězce v SCG_1 a SCG_n stejný, mohou algoritmy interně pracovat s SCG_n namísto přímé práce s SCG_1. Všechny algoritmy samozřejmě pracují s datovými strukturami popsanými v podsekci 3.1.7.

4.1.1 Derivace řetězce v SCG_n

Derivace řetězce v SCG_n má velmi specifický průběh (viz podsekci 2.2.2), kdy jsou nejprve používána pouze bezkontextová pravidla. Mimo to kontextová pravidla nemá smysl aplikovat libovolně, ale existuje pouze jediný způsob jak pravidlo aplikovat, který může vést k úspěšné derivaci. Proto algoritmu stačí řešit pouze aplikaci bezkontextových pravidel.

Základním algoritmem pro demonstraci derivace řetězce v SCG_n je tedy prohledávání stavového prostoru se zpětným navracením (*backtracking*). Stavům odpovídají větné formy a operátorům odpovídají bezkontextová pravidla, která lze na větnou formu aplikovat. Pokud větná forma obsahuje speciální symbol $\bar{R}$, nelze již aplikovat žádná další pravidla, ale zda se jedná o cílový stav musí být ověřeno eliminací větné formy pomocí kontextových pravidel. Metoda prohledávání se zpětným navracením není úplná ani optimální

Zacyklení

Metoda nemusí obsahovat rozšíření, které zabraňuje zacyklení, protože v tomto případě nemůže dojít k zacyklení metody v pravém slova smyslu. Je to proto, že všechna bezkontextová pravidla přepisují jeden symbol větné formy na nejméně dva symboly (viz podsekci 2.2.1).

Výjimkou jsou v tomto pravidla vytvářená v pátém kroku konstrukce, ale aplikace takového pravidla značí konec derivace a nelze tedy tato pravidla aplikovat opakovaně. Aplikací libovolné sekvence pravidel nad výchozí větnou formou tedy nelze nikdy znovu dosáhnout výchozí větné formy, protože každé kontextové pravidlo větnou formu prodlouží o další symboly, ale neexistuje pravidlo, které by větnou formu zkracovalo.

Naproti tomu však může dojít k zacyklení způsobenému potenciálně nekonečným stavovým prostorem, kdy dojde k opakované aplikaci jednoho kontextového pravidla a větná forma tak bude neustále prodlužována. Tomuto způsobu zacyklení nelze snadno zabránit, protože správná derivace řetězce může vyžadovat opakovanou aplikaci pravidla. Výše zmíněného faktu o neustálém prodlužování větné formy by se dalo využít k omezení maximální hloubky prohledávání tak, že se omezí například maximální počet R-symbolů ve větné formě (který v důsledku značí počet použitých pravidel). Nicméně to by mohlo znamenat, že algoritmus nebude schopen pro některé složitější gramatiky najít řešení za žádných okolností, a proto je na heuristice, aby libovolným způsobem zabránila zacyklení algoritmu.

Heuristika

Algoritmus je rozšířen o heuristiku, která se stará o výběr pravidla v každém stavu. Heuristika je volána pouze tehdy, pokud existuje více než jedno pravidlo aplikovatelné na větnou formu. Pokud lze aplikovat pouze jedno pravidlo, algoritmus jej aplikuje bez nutnosti volání heuristiky. Heuristika má rovněž možnost nevybrat žádné z dostupných pravidel a tím donutit algoritmus ke zpětnému navrácení.

4.1.2 Ověření správnosti derivace řetězce

K ověření správnosti derivace řetězce v $\mathrm{SCG_n}$ je nutné generovat ten samý řetězec i pomocí frontové gramatiky. Ověření bude prováděno na normalizované gramatice tak, aby v případě algoritmické normalizace mohla být ověřena její správnost. Zpracování řetězce pomocí frontové gramatiky bude implementováno opět pomocí prohledávání stavového prostoru se zpětným navracením. Díky o poznání menšímu množství pravidel frontové gramatiky nemusí algoritmus pracovat s žádnou heuristikou, ale pro potřeby aplikace bude stačit vybírat pravidla dle pořadí v seznamu pravidel. Jedinými zbývajícími problémy tak jsou neomezená maximální hloubka prohledávání a možnost zacyklení algoritmu.

Zacyklení je řešeno rozšířením metody prohledávání se zpětným navracením tak, že nelze použít operátor, který by vedl na již dříve expandovaný stav. Větná forma totiž jednoznačně určuje množinu pravidel, která na ni lze aplikovat a tím tedy i množinu dosažitelných větných forem. Pokud se tedy byla větná forma již expandována, jejím opětovným zpracováním se nedosáhne žádných nových větných forem.

Maximální hloubka prohledávání může být opět omezena délkou větné formy. V tomto případě není takový problém, pokud nedojde k nalezení řešení z důvodu příliš striktního omezení hloubky prohledávání, protože cílem algoritmu je pouze provádět kontrolu. Uživatel by však měl mít možnost nastavit si maximální hloubku prohledávání této metody.

4.1.3 Normalizace a odvození

Algoritmy pro normalizaci frontové gramatiky a odvození $\mathrm{SCG_n}$ od frontové gramatiky odpovídají jejich formálnímu zápisu. Dochází k sekvenčnímu procházení původních pravidel a vytvářením příslušných nových pravidel na základě daných podmínek. Algoritmy by se

pravděpodobně daly optimalizovat jak po návrhové, tak po implementační stránce, ale po potřeby aplikace to není nutné a zůstane tak zachována vyšší přehlednost algoritmů.

4.2 Implementace algoritmů

Algoritmy jsou jako součást aplikace rovněž implementovány v jazyce C. Jejich implementace není nijak zvlášť oddělena od zbytku aplikace, a proto se v ní nachází i příkazy pro počítání statistik či tisk průběžných výsledků.

4.2.1 Derivace řetězce v SCG_n

Algoritmus pro derivaci řetězce je implementován běžným způsobem pomocí zásobníku. Zásobník bohužel nelze, kvůli limitacím jazyka C, v aplikaci znovu využít pro jiné účely, a proto je implementován ve stejném modulu jako samotný algoritmus.

Každá položka zásobníku reprezentuje stav a obsahuje příslušnou větnou formu, společně se seznamem pravidel aplikovatelných na větnou formu, která ještě nebyla v daném stavu použita. Obě tyto informace jsou během derivace předávány heuristice.

Proces derivace

Algoritmus derivuje řetězec následujícím způsobem:

1. Algoritmus tedy umístí na zásobník počáteční symbol jako větnou formu společně s pravidly, která na ni lze aplikovat;
2. Pokud je zásobník prázdný, derivace je ukončena neúspěchem, jinak algoritmus pokračuje;
3. Uzel na vrcholu zásobníku může být zpracován několika způsoby.
   (a) Pokud uzel na vrcholu zásobníku obsahuje jedno pravidlo, pak je toto pravidlo aplikováno a odstraněno z aplikovatelných pravidel v položce na vrcholu zásobníku. Algoritmus pokračuje bodem 4;
   (b) Pokud uzel na vrcholu zásobníku obsahuje více než jedno pravidlo, pak jsou příslušné informace předány heuristice a pokud ta vrátí zvolené pravidlo, toto pravidlo je aplikováno a odstraněno z aplikovatelných pravidel v položce na vrcholu zásobníku. Algoritmus pokračuje bodem 4;
   (c) Pokud uzel na vrcholu zásobníku neobsahuje žádná aplikovatelná pravidla nebo heuristika nevrátila žádné pravidlo, je uzel odstraněn a algoritmus pokračuje bodem 2;
4. Pokud větná forma nově aplikovaného pravidla obsahuje speciální symbol $\bar{R}$, tak je provedena eliminace a v případě jejího úspěchu je derivace ukončena také úspěchem. Jinak je položka s touto větnou formou a na ni aplikovatelnými pravidly umístěna na zásobník a algoritmus pokračuje bodem 2.

4.2.2 Derivace řetězce ve frontové gramatice

Implementace derivace řetězce ve frontové gramatice je velmi podobná již popsané implementaci derivace v SCG_n. Bohužel opět kvůli limitacím jazyka C nebylo možné implemen-

taci algoritmu znovu použít. Zásobník je ze stejných důvodů také implementován v modulu s algoritmem.

Díky absenci heuristiky však bylo možné využít zjednodušit výběr pravidel, a v každé položce zásobníku je tedy mimo větnou formu pouze index posledního aplikovaného pravidla nad touto větnou formou.

Navíc bylo nutné implementovat rozšíření bránící zacyklení algoritmu. Algoritmus tedy nově vzniklou větnou formu přidá do zásobníku pouze v případě, že se tam ještě položka s takovou větnou formou nenachází. Maximální délka větné formy je omezena na 2 symboly pro řetězce délky 1 a menší. Pro řetězce o délce větší než 1 je pak maximální délka větné formy omezena na délku řetězce krát 3. Tyto koeficienty byly získány empiricky a není tak zaručena jejich správnost. Uživatel proto musí mít i možnost nastavit maximální délku větné formy na libovolnou hodnotu dle svého uvážení.

4.2.3 Paměť

Hlavně při práci se složitějšími gramatikami může aplikace být velmi náročná na paměť. To je způsobeno hlavně velkým množstvím pravidel, jejichž počet se může po normalizaci a konverzi na $\mathrm{SCG_n}$ pohybovat až v řádu milionů. Je to způsobeno hlavně druhým krokem konstrukce (viz podsekci 2.2.1), kde je pro každé pravidlo vytvořeno

$$\binom{m + |y| - 1}{|y|} \cdot n$$

nových pravidel, kde m je počet nekoncových stavů frontové gramatiky, $|y|$ je délka zapisovaného řetězce v původním pravidle a n je počet neterminálů frontové gramatiky. Jak je vidět, množství pravidel velmi rychle roste hlavně s přibývajícími stavy a délkou zapisovaných řetězců v pravidlech.

Gramatiky s takovým množstvím pravidel již z hlediska demonstrace nemají valnou hodnotu, je pro uživatele prakticky nemožné se ve výběru pravidel orientovat. Aplikace proto tento problém neřeší přímo, ale pouze se snaží eliminovat množství případných chyb při nedostatku paměti. Všechna pole, která mohou nabývat velikosti vyšší, než desítky prvků jsou tedy alokována dynamicky tak, aby případná chyba při alokaci mohla být korektně zachycena a zpracována.

Další zátěží na paměť je kopírování datových struktur. Struktury jsou sice v aplikaci předávány referencí, nicméně pro vyšší přehlednost následného uvolňování paměti je často výhodné alokovat novou kopii struktury, tu předat a později uvolnit, protože se tím snižuje riziko nechtěné editace struktury na více místech, pokusu o uvolnění struktur, které již byly uvolněny, nebo naopak úniku paměti.

Například při vytváření nového pravidla je z hlediska programátora podstatně jednodušší vždy vytvořit nový L-symbol a během uvolňování pravidla jej pak vždy uvolnit. Zjišťovat, zda již daný L-symbol byl někde v aplikaci použit, případně použít referenci na existující symbol a uvolnit jej pak pouze jednou je obtížnější.

Tento problém by se dal samozřejmě řešit příslušným mechanismem, nicméně by to stále nevyřešilo velký počet pravidel, takže je rozumnější raději snížit riziko chyb při psaní kódu. Aplikace by se v této oblasti dala rozhodně optimalizovat, ale pro demonstraci jednodušších gramatik je současné řešení dostačující.

4.2.4 Částečná oprava chybné konstrukce SCG_n

Během testování aplikace byla odhalena chyba v algoritmu konstrukce SCG_n převzatém z článku [6], která je popsána v sekci 5.6. Konstrukce SCG_n je tedy implementována odlišně od algoritmu popsaného v podsekci 2.2.1.

V 2. fázi konstrukce jsou navíc pro každé pravidlo $(a, q, y, p) \in R$, kde $|y| = 1$ vytvářena pravidla ve tvaru:

$$\langle q, 1\rangle \rightarrow \lceil \underline{y}, q_1 \rceil \langle p, 1\rangle \lfloor b, \underline{p} \rfloor,$$

pro všechny nekoncové stavy q_1. Kompletní úprava algoritmu je popsána právě v sekci 5.6.

4.3 Heuristika

Jak již bylo zmíněno, derivaci řetězce v SCG_n aplikace provádí pomocí prohledávání stavového prostoru. Protože i jednoduché frontové gramatiky mají po převodu na SCG_n velké množství pravidel, je nutné, aby algoritmus používal pro volbu pravidel heuristiku.

4.3.1 Návrh

Navržená heuristika se soustředí na snížení počtu pravidel, ze kterých musí algoritmus vybírat v každém derivačním kroku, čímž by mělo dojít k redukci stavového prostoru.

Kontextové pravidlo odstraňuje vždy nejpravější L-symbol[1] a nejlevější R-symbol[2] z větné formy. Z toho vyplývá, že větná forma musí mít na konci derivace takový tvar, aby byly L-symboly a R-symboly během závěrečné eliminace komplementární a ve stejném pořadí. L-symbol $\lceil \underline{a}, q \rceil$ a R-symbol $\lfloor b, \underline{p} \rfloor$ jsou komplementární, jestliže platí $a = b \wedge q = p$. Pořadí L-symbolů je bráno zprava, zatímco pořadí R-symbolů je bráno zleva. Současně kontextové pravidlo ve větné formě vyžaduje speciální symboly $\bar{L}$ a $\bar{R}$.

Konkrétní příklad žádoucí výsledné větné formy je:

$$\lceil \underline{S}, s \rceil \lceil \underline{S}, n \rceil \lceil \underline{A}, t \rceil \bar{L} aabb \bar{R} \lfloor A, \underline{t} \rfloor \lfloor S, \underline{n} \rfloor \lfloor S, \underline{s} \rfloor,$$

Validní větná forma SCG_n nemůže nikdy obsahovat více R-symbolů než L-symbolů. L-symboly totiž představují všechny neterminály zapsané původní frontovou gramatikou do větné formy během derivace, zatímco R-symboly naopak představují všechny neterminály, které frontová gramatika čte. Vyšší počet R-symbolů než L-symbolů by tedy znamenal, že frontová gramatika během derivace věty odstranila z fronty více neterminálů než do ní bylo vloženo, což je nemožné. V SCG_n sice tento stav může teoreticky nastat, nicméně nikdy nepovede ke korektní derivaci řetězce (viz podsekci 2.2.2).

Volba pravidel

Heuristika tedy volí pravidla tak, aby zachovala správné pořadí symbolů na obou stranách větné formy. Jak vyplývá z podsekce 2.2.1, každé pravidlo SCG_n obsahuje právě jeden R-symbol, a heuristika proto vybírá pravidlo právě s ohledem na tento symbol. Výjimkou jsou pouze pravidla přepisující speciální symboly $\bar{L}$ a $\bar{R}$, která ale nepřepisují žádné L-symboly nebo R-symboly, a proto je jejich aplikace přímočará.

Volba pravidla tedy probíhá takto:

[2] L-symbol je neterminál ve tvaru $\lceil \underline{a}, q \rceil$, kde a je neterminál frontové gramatiky a q je nekoncové stavy frontové gramatiky.

[2] R-symbol je neterminál ve tvaru $\lfloor a, \underline{q} \rfloor$, kde a je neterminál frontové gramatiky a q je nekoncové stavy frontové gramatiky.

- Pokud je počet L-symbolů roven počtu R-symbolů ve větné formě, musí být aplikováno pravidlo, jehož nejlevější L-symbol je komplementární k jedinému R-symbolu na pravé straně pravidla, čímž zůstane zachováno správné pořadí symbolů;
- Pokud je počet L-symbolů vyšší než počet R-symbolů ve větné formě, musí být aplikováno takové pravidlo, aby byl jeho jediný R-symbol komplementární k příslušnému L-symbolu ve větné formě a bylo tak zachováno správné pořadí symbolů.

4.3.2 Implementace

Heuristika sekvenčně prochází aplikovatelná pravidla a hodnotí je. Po ohodnocení všech pravidel aplikuje to, které má nejvyšší hodnocení. Pravidlo nemůže být aplikováno, pokud by došlo k překročení maximální hloubky prohledávání nebo pokud by počet terminálů ve větné formě překročil délku derivované věty.

Hloubka prohledávání

Jelikož každé pravidlo, které lze během derivace aplikovat opakovaně, přidává do větné formy nejméně jeden R-symbol (viz podsekci 2.2.1), může být maximální hloubka prohledávání určována pomocí počtu L-symbolů a R-symbolů ve větné formě. Pokud tedy počet L-symbolů nebo počet R-symbolů překročí zadanou hodnotu, znamená to, že byla překročena maximální hloubka. Heuristika nedokáže sama určit maximální hloubku prohledávání, výchozí hodnota je nastavena na 10, ale je na uživateli, aby tuto hodnotu upravil na základě vstupní frontové gramatiky a délky řetězce.

Speciální symbol $\bar{L}$

Protože heuristika neví, kolik L-symbolů bude nutné přidat do větné formy, aby derivace byla úspěšná, neví ani, kdy aplikovat pravidlo přepisující speciální symbol $\bar{L}$. Přepis symbolu $\bar{L}$ v důsledku ukončuje derivaci řetězce, protože $\mathrm{SCG_n}$ přechází do stavů označených číslicí 2, kde mohou být aplikována pouze pravidla, vytvářená ve 2. fázi konstrukce v podsekci 2.2.1, následujících tvarů:

$$\langle q, 2\rangle \to y\langle p, 2\rangle \lfloor b, \underline{p} \rfloor,$$
$$\langle q, 2\rangle \to y\bar{R}.$$

Z toho vyplývá, že pokud se gramatika ve stavu označeném číslicí 2, může již docházet pouze k přidávání terminálů do větné formy a nelze se žádným pravidlem vrátit zpět do stavu označeného číslicí 1. Heuristika proto na každou větnou formu aplikuje pravidlo přepisující symbol $\bar{L}$, je li to možné, a tím ověří zda již nelze aplikovat pravidla obsahující terminály.

Hodnocení pravidel

Jak bylo naznačeno v návrhu, hodnocení pravidel se odvíjí od současné větné formy. Konkrétně od počtu L-symbolů a R-symbolů. Jestliže více pravidel získá stejné hodnocení, jsou aplikována podle pořadí v seznamu pravidel.

Pokud je počet L-symbolů roven počtu R-symbolů, pak jsou pravidla hodnocena následovně:

1. Nejvýše jsou hodnocena pravidla přepisující speciální symbol $\bar{R}$, protože toto pravidlo ukončuje derivaci a zahajuje tak eliminaci. L-symboly a R-symboly ve větné formě jsou komplementární a ve stejném pořadí, a proto je žádoucí zahájit eliminaci;

2. Nižší ohodnocení je přiřazeno pravidlům, která mají alespoň jeden L-symbol a jejichž nejlevější L-symbol je komplementární k jedinému R-symbolu. To znamená, že stejné pořadí L-symbolů a R-symbolů zůstane zachováno. Současně je tak zajištěno, že větná forma nemůže nikdy obsahovat více R-symbolů než L-symbolů.

3. Ostatní pravidla nemohou být nikdy aplikována.

Pokud je počet L-symbolů ve větné formě vyšší než počet R-symbolů, jsou pravidla vybírána podle R-symbolu, který obsahují. R-symbol v pravidle musí odpovídat L-symbolu na $n + 1$ pozici zleva, kde n je počet R-symbolů ve větné formě. Tím zůstane zajištěno stejné pořadí L-symbolů a R-symbolů ve větné formě. Ostatní pravidla opět nemohou být aplikována.

Například jestliže má současná větná forma tvar:

$$\lceil \underline{S}, s \rceil \lceil \underline{S}, n \rceil \lceil \underline{A}, t \rceil \langle n, 1 \rangle \lfloor S, \underline{s} \rfloor,$$

pak musí být aplikováno pravidlo obsahující R-symbol $\lfloor S, \underline{n} \rfloor$.

Kapitola 5

Testování

Aplikace a heuristika je testována, aby bylo možné vyhodnotit kvalitu navržené heuristiky, správnost implementace všech algoritmů, ale i ověřit závěry článku [6]. Tato kapitola popisuje mimo způsobu a výsledků testování i chyby nalezené v článcích [5][6].

5.1 Gramatiky

Aplikace a heuristika je testována na 10 různých frontových gramatikách v druhé normální formě a na 4 dalších frontových gramatikách, které nejsou v druhé normální formě a jsou tedy normalizovány algoritmicky přímo aplikací.

Soubor	Jazyk	Přijaté řetězce	Odmítnuté řetězce
`example1.json`	$\{aabb\}$	$aabb$	ε, a, $abab$, ...
`example2.json`	$\{n(t+f)\}$	nt, nf	ε, nn, $nfnt$, ...
`example3.json`	$\{a^{2n}, n \geq 0\}$	ε, a, aa, $aaaa$, ...	aaa, $aaaaa$, ...
`example4.json`	$\{a^{2n}b^{2n}, n \geq 1\}$	$aabb$, $aaaabbbb$, ...	ε, a, ab, $abba$, ...
`example5.json`	$\{(a+b)^n, n \geq 0\}$	ε, a, ab, baa, ...	
`example6.json`	$\{a^n + b^n, n \geq 0\}$	ε, aa, bbb	ab, $abab$, $aaab$, ...
`example7.json`	$\{x^m y^n, m, n \geq 1\}$	xy, xxy, $xyyy$, ...	ε, x, yy, $xyxx$, ...
`example8.json`	$\{a^{2^n}, n \geq 0\}$	ε, a, aa, $aaaa$, ...	aaa, $aaaaaa$, ...
`example9.json`	$\{a^n b^n c^n, n \geq 1\}$	abc, $aabbcc$, ...	ε, $aabb$, $abcac$, ...
`example10.json`	$\{vv, v \in \{a,b,c\}^+\}$	aa, $cbcb$, $bbbb$, ...	ε, ac, $aabb$, ...

Tabulka 5.1: Přehled testovaných souborů s gramatikami v druhé normální formě.

Soubor	Jazyk	Přijaté řetězce	Odmítnuté řetězce
`example11.json`	$\{\varepsilon\}$	ε	a, aa, aaa, ...
`example12.json`	$\{a^n, n \geq 0\}$	ε, a, aa, aaa, ...	
`example13.json`	$\{a^n b^n, n \geq 0\}$	ε, ab, $aabb$, ...	a, ba, $aaab$, ...
`example14.json`	$\{vv, v \in \{a,b,c\}^+\}$	aa, $cbcb$, $bbbb$, ...	ε, ac, $aabb$, ...

Tabulka 5.2: Přehled testovaných souborů s gramatikami, které nejsou v druhé normální formě.

Gramatiky budou dále odkazovány pomocí čísla v názvu příslušného souboru.

5.2 Metodika

Aplikace je pro každou frontovou gramatiku opakovaně spouštěna pro všechny řetězce maximální délky 4. Řetězce jsou složeny pouze z terminálů frontové gramatiky a maximální hloubka prohledávání je nastavena na konstantní hodnotu pro všechny gramatiky. Pro vyšší délku řetězců by již celková doba testování byla neúnosná.

Během testování je navržená heuristika porovnávána s heuristikou, která aplikuje pravidla podle pořadí v seznamu pravidel (dále jen *základní heuristika*). Obě heuristiky určují hloubku prohledávání pomocí L-symbolů a R-symbolů způsobem popsaným v podsekci 4.3.2.

Při testování základní heuristiky, byla maximální hloubka prohledávání nastavena na hodnotu 6, vyšší hodnoty vedly u většiny gramatik k době zpracovat jednoho řetězce v řádech hodin. I pro takto nízkou hodnotu se nepodařilo zpracovat žádné řetězce v gramatikách 4, 7, 9, 10 a 11–14, protože doba zpracování jediného řetězce v těchto gramatikách překročila 1 hodinu. Maximální hloubka prohledávání 6 nestačí k přijetí některých testovaných řetězců.

Navržená heuristika byla rovněž testována pro maximální hloubku 6, aby mohlo dojít k přímému porovnání výsledků s výsledky testů základní heuristiky. Dále byla heuristika testována pro maximální hloubku 15, která zajišťuje, že každý testovaný řetězec může být přijat. Takto bylo možné testovat pouze gramatiky 1–9, u gramatiky 10 opět trvalo zpracování řetězce déle než hodinu.

Gramatiky 11–14 jsou normalizovány algoritmicky aplikací. Gramatiky 13 a 14 mají po normalizaci příliš velké množství pravidel, a proto byly testovány pouze pro maximální hloubku prohledávání 1.

5.3 Struktura výsledků

Výsledky jsou zapisovány ve formě tabulky, kde jsou pro každý řetězec zaznamenány následující statistiky ohledně jeho derivace:

- Konzistence platí, jestliže byl řetězec přijat oběma gramatikami nebo odmítnut oběma gramatikami. Oběma gramatikami se myslí vstupní frontová gramatika po případné normalizaci a z ní odvozená $\mathrm{SCG_n}$;
- Počet rozhodnutí říká kolikrát algoritmus vybíral pravidlo k aplikaci;
- Optimální délka říká kolik kroků má výsledná derivace řetězce, tím pádem tedy kolikrát by se algoritmus rozhodl pokud by každé jeho rozhodnutí bylo správné;
- Úspěšnost počítána jako $\frac{\text{optimální délka}}{\text{počet rozhodnutí}}$;
- Zda byl řetězec přijat $\mathrm{SCG_n}$;
- Počet zpětných navrácení je počet případů, kdy nemohlo být aplikováno žádné pravidlo a algoritmus se tak vrátil o úroveň výš.

5.4 Výsledky

Kompletní výsledky testování, včetně výsledků před částečnou opravou chyby (viz podsekci 4.2.4), jsou k dispozici na přiloženém mediu (viz přílohu B).

Při testování gramatik 11 a 13 byla odhalena chyba v normalizaci, kdy některé řetězce patřící do jazyka nebyly přijaty po normalizaci frontovou gramatikou. Ve výsledcích testování se toto projevilo tak, že SCG_n řetězec patřící do jazyka přijímaného vstupní frontovou gramatikou nepřijímá, ale ve sloupci `Konzistence` je hodnota `Ano`. Protože je kontrolní derivace řetězce ve frontové gramatice prováděna po normalizaci, značí to, že je algoritmus normalizace chybný. Chyba je podrobně popsána v následující sekci 5.5.

Na přiloženém médiu jsou obsaženy i původní výsledky testování pro maximální hloubku 15, kdy byla odhalena chyba v konstrukci SCG_n (viz sekci 5.6). Chyba se projevuje nepřijetím řetězce v SCG_n, přestože patří do jazyka přijímaného frontovou gramatikou, a zasaženy jsou ní všechny gramatiky 3–10. Nejlépe je chyba vidět na gramatice 5, která by měla přijímat libovolné řetězce terminálů, ale během původního testování nebyl přijat ani jeden řetězec. Testování po částečné opravě chyby popsané v podsekci 4.2.4 a v sekci 5.6 pro maximální hloubku 15, ukazuje, že oprava chyby byla úspěšná pro gramatiky 1–7 a 9, nicméně gramatiky 8 a 10 nyní navíc přijímají i některé řetězce jazyka, které nepatří do jazyka přijímaného vstupní frontovou gramatikou.

5.4.1 Časová složitost

Odvození časové složitosti z výsledků testování je obtížné, protože gramatiky většinu řetězců nepřijímají a dojde tak vždy k prohledání celého stavového prostoru omezeného maximální hloubkou. Nicméně výsledky testování ukazují značné snížení počtu rozhodnutí při použití navržené heuristiky, což koresponduje s tím, že tato heuristika snižuje počet aplikovatelných pravidel v každém kroku. Toto snížení se týká jak přijatých, tak nepřijatých řetězců.

K odmítnutí řetězce je nutné prohledat celý stavový prostor. Počet rozhodnutí nutný k odmítnutí řetězce proto může být přibližným ukazatelem výkonnosti heuristiky. Tabulka 5.3 ukazuje, že navržená heuristika značně zvyšuje výkonnost aplikace.

Gramatika	Základní heuristika	Navržená heuristika
1	25	25
2	18497	53
3	141265	191
5	17268641	827
6	8698825	471
8	14400061	243

Tabulka 5.3: Počty rozhodnutí potřebných k odmítnutí řetězce v testovaných gramatikách při použití základní a navržené heuristiky. Maximální hloubka prohledávání byla nastavena na hodnotu 6.

Gramatika 5 přijímá všechny řetězce, a proto se nejvíce hodí pro určení časové složitosti navržené heuristiky. Nicméně počty rozhodnutí pro jednotlivé délky řetězců klesají, jak je vidět v tabulce 5.4. Důvodem je, že heuristika stále vybírá pravidla podle pořadí v seznamu pravidel a současně metoda prohledávání se zpětným navracením není optimální. Proto aplikace derivuje všechny řetězce sekvencí 17 derivačních kroků i přesto, že například prázdný řetězec lze derivovat 5 kroky. Řešením by mohlo být testování pro vyšší délky řetězců, ale

to by znamenalo nutné zvýšení maximální hloubky prohledávání, což by vedlo k neúnosné časové náročnosti testování.

Očekávaná časová složitost algoritmu používajícího navrženou heuristiku je exponenciální, protože časová složitost prohledávání stavového prostoru se zpětným navracením je rovněž exponenciální a navržená heuristika pouze snižuje velikost stavového prostoru [1].

Délka řetězce	0	1	2	3	4
Průměrný počet rozhodnutí	778095	264775	133598	71547	45556

Tabulka 5.4: Průměrný počet rozhodnutí podle délky řetězce pro gramatiku 5, který nevypovídá o časové složitosti algoritmu.

5.5 Chyba v převodu frontové gramatiky do 1. normální formy

Při testování aplikace vyšlo najevo, že některé gramatiky po algoritmickém převodu do 2. normální formy nepřijímají řetězce, které přijímaly před převodem. To je způsobeno chybným algoritmem převodu gramatiky do první normální formy v publikacích [5][6].

Tuto chybu v normalizaci lze nejsnáze demonstrovat na jednoduchém příkladu 5.1, nicméně se dotýká i složitějších gramatik jako je ta v příkladu 5.2. Jak je vidět na příkladech, problémem je speciální symbol 1, který nelze vygenerovat tak, aby jeho přepsáním bylo možné dosáhnout koncového stavu gramatiky v 1. normální formě.

Bohužel se zatím nepodařilo nalézt žádné řešení tohoto problému, které by nezpůsobovalo, že gramatika po normalizaci přijímá i řetězce, které před normalizací nepřijímala. Přestože lze jazyk popisovaný gramatikou v příkladu 5.1 popsat frontovou gramatikou ve 2. normální formě podobou gramatice v souboru `example1.json` a stejně tak lze jazyk popisovaný gramatikou v příkladu 5.2 popsat frontovou gramatikou ve 2. normální formě podobnou gramatice v souboru `example.4.json`.

Příklad 5.1. Mějme frontovou gramatiku $H = (\bar{V}, T, \bar{W}, \{f\}, Ae, \bar{R})$, kde $\bar{V} = \{A, a\}$, $T = \{a\}$, $\bar{W} = \{e, f\}$ a $\bar{R} = \{(A, e, \varepsilon, f)\}$.

Gramatiku H převedeme podle podsekce 2.1.2 na gramatiku $Q = (V, T, W, f_Q, A'e', R)$ v 1. normální formě, kde $V = \{A', a', 1, a\}$, $W = \{e', f', e'', f'', f_Q, \langle \varepsilon, f \rangle\}$ a množina pravidel R obsahuje:

1. $(A', e', \varepsilon, f')$
2. $(A', e', 1, f')$
3. $(A', e', \varepsilon, \langle \varepsilon, f \rangle)$
4. $(1, \langle \varepsilon, f \rangle, \varepsilon, f'')$
5. $(A', e'', \varepsilon, f'')$
6. $(A', e'', \varepsilon, f_Q)$

Gramatika Q by podle publikací [5] a [6] měla být ekvivalentní s gramatikou H.

Větu ε lze zřejmě derivovat v gramatice H pomocí jediného derivačního kroku $Ae \Rightarrow \varepsilon f$. Pokud se však pokusíme derivovat větu ε v gramatice Q, existují pouze 3 možné derivační kroky z počátečního symbolu a počátečního stavu:

1. Krok $A'e' \Rightarrow \varepsilon f'$ [1], kde však množina pravidel R neobsahuje žádné pravidlo s výchozím stavem f' a není tak možné v derivaci pokračovat;

2. Krok $A'e' \Rightarrow 1f'$ [2], kde opět množina pravidel R neobsahuje žádné pravidlo s výchozím stavem f' a není tak možné v derivaci pokračovat;

3. Krok $A'e' \Rightarrow \varepsilon\langle\varepsilon, f\rangle$ [3], kde sice množina R obsahuje pravidlo 4 s výchozím stavem $\langle\varepsilon, f\rangle$, ale na začátku fronty není symbol 1, tudíž rovněž nelze v derivaci pokračovat.

Z toho plyne, že gramatika Q nepřijímá řetězec ε a není tak ekvivalentní s gramatikou H.

Příklad 5.2. Mějme frontovou gramatiku $H = (\bar{V}, T, \bar{W}, \{f\}, Se, \bar{R})$, kde $\bar{V} = \{S, a, b\}$, $T = \{a, b\}$, $\bar{W} = \{e, f\}$ a $\bar{R} = \{(S, e, bSa, e), (S, e, \varepsilon, f), (a, e, a, e), (b, e, b, e)\}$. Tato gramatika popisuje jazyk $\{a^n b^n, n \geq 0\}$ a řetězec ab může být derivován následujícím způsobem:

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

Následně se pokusíme derivovat větu ab v gramatice $Q = (V, T, W, \{f_Q\}, S'e', R)$ v 1. normální formě, která byla převedena podle algoritmu v podsekci 2.1.2. Gramatika Q je poměrně složitá, a proto zde není uvedena celá (pro kompletní gramatiku viz přiložené médium).

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

$$abf_Q.$$

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

$$S'abe'' \Rightarrow abf_Q \left[(S', e'', \varepsilon, f_Q)\right].$$

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 $S'ab$:

1. $(1, \langle b, e\rangle, b, e'')$,

2. (b', e'', b, e'').

Nejprve tedy zkusíme aplikovat pravidlo 1 a provedeme příslušný derivační krok:

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

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:

$$b'S'ae' \Rightarrow 1S'a\langle b, e\rangle \left[(b', e', \varepsilon, \langle b, e\rangle)\right] \Rightarrow S'abe'' \Rightarrow abf_Q.$$

V gramatice Q 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 $A'e' \Rightarrow^* b'S'ae'$ a tímto způsobem nelze řetězec ab derivovat.

Pokud provedeme stejný postup i pro pravidlo 2, dostaneme tyto tři sekvence derivačních kroků:

$$S'1e' \Rightarrow 1b'S'\langle a,e\rangle\ [(S',e',b'S',\langle a,e\rangle)] \Rightarrow b'S'ae''\ [(1,\langle a,e\rangle,a,e'')] \Rightarrow S'abe''\ [2] \Rightarrow abf_Q,$$

$$a'1b'S'e' \Rightarrow 1b'S'\langle a,e\rangle\ [(a',e',\varepsilon,\langle a,e\rangle)] \Rightarrow b'S'ae''\ [(1,\langle a,e\rangle,a,e'')] \Rightarrow S'abe''\ [2] \Rightarrow abf_Q,$$

$$1a'b'S'\langle\varepsilon,e\rangle \Rightarrow a'b'S'e''\ [(1,\langle\varepsilon,e\rangle,\varepsilon,e")] \Rightarrow b'S'ae''\ [(a',e'',a,e'')] \Rightarrow S'abe''\ [2] \Rightarrow abf_Q.$$

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 Q neexistuje pravidlo s příslušnou kombinací cílového stavu a zapisovaného řetězce.

Z toho vyplývá, že neexistuje sekvence derivačních kroků $S'e' \Rightarrow^* abf_Q$ v gramatice Q a tato gramatika tedy nepřijímá řetězec ab. Gramatiky H a Q proto nejsou ekvivalentní.

5.6 Chyba v konstrukci SCG$_n$

Při testování aplikace některé frontové gramatiky po převodu na SCG$_n$ nepřijímaly řetězce, které přijímaly před převodem. Tyto gramatiky byly převedeny do druhé normální formy ručně, a proto nejsou ovlivněny výše zmíněnou chybou v normalizaci. Například frontová gramatika ve 2. normální formě v souboru `example5.json`, která neodmítá žádné řetězce terminálů, po převodu na SCG$_n$ nepřijala ani jeden řetězec. Toto je způsobeno chybou v konstrukci SCG$_n$, která je způsobena chybným algoritmem konstrukce v článku [6]. Příklad 5.3 ukazuje jednu z gramatik, pro kterou frontová gramatika v 2. normální formě a SCG$_n$ nejsou ekvivalentní.

Příklad 5.3. Mějme frontovou gramatiku $Q = (V,T,W,F,Ss,R)$ ve 2. normální formě, kde $V = \{S,N,A,a\}$, $T = \{a\}$, $W = \{s,n_1,n_2,r_1,r_2,f\}$, $F = \{f\}$ a $R = \{(S,s,N,r_2)$, $(S,s,NA,n_2),(A,n_1,AA,n_1),(A,n_1,AA,n_2),(N,n_2,N,n_1),(N,n_2,N,n_2),(N,n_2,N,r_1)$, $(A,r_1,a,r_1),\ (A,r_1,a,r_2),\ (N,r_2,\varepsilon,f)\}$.

Tato gramatika, která popisuje jazyk $\{a^{2^n}, n \geq 0\}$, se rovněž nachází na přiloženém médiu v souboru `example8.json`. Řetězec a může být v gramatice Q derivován následujícím způsobem:

$$Ss \Rightarrow NAn_2 \Rightarrow ANr_1 \Rightarrow Nar_2 \Rightarrow af.$$

Následně se pokusíme derivovat větu a v SCG$_n$ $H = (U \cup N \cup M, T, P, S_H)$. Tato gramatika byla zkonstruována podle algoritmu v podsekci 2.2.1. Abecedy a množina pravidel jsou poměrně rozsáhlé, a proto zde nejsou vypsány (pro kompletní množinu pravidel viz přiložené médium).

Derivační krok 1

První pravidlo je aplikováno tak, aby L-symbol[1] a R-symbol[2] byly komplementární. Protože každé další pravidlo přepisuje symboly ve větné formě mezi touto dvojicí symbolů, budou tyto symboly posledními dvěma symboly, které kontextové pravidlo přepíše, a musí proto být komplementární. Provedeme tedy derivační krok:

$$S_H \Rightarrow \lceil\underline{S},s\rceil\langle s,1\rangle\lfloor S,\underline{s}\rfloor.$$

[1] L-symbol je neterminál ve tvaru $\lceil\underline{a},q\rceil$, kde a je neterminál frontové gramatiky a q je nekoncový stav frontové gramatiky.

[2] R-symbol je neterminál ve tvaru $\lfloor a,\underline{q}\rfloor$, kde a je neterminál frontové gramatiky a q je nekoncový stav frontové gramatiky.

Derivační krok 2

Při derivaci věty v gramatice Q bylo v dalším derivačním kroku aplikováno pravidlo (S, s, NA, n_2). Protože by gramatika H měla simulovat derivaci řetězce v Q, vyplývá z toho, že v gramatice H by mělo být aplikováno pravidlo vytvořené při konstrukci ve 2. fázi právě z pravidla (S, s, NA, n_2). Současně, nejlevější L-symbol v pravidle a nejpravější R-symbol v pravidle musí být opět komplementární. Všechna pravidla, která můžeme aplikovat v dalším derivačním kroku, mají tedy tvar:

$$\langle s, 1\rangle \to \lceil \underline{N}, n_2\rceil\lceil \underline{A}, q\rceil\langle n_2, 1\rangle\lfloor N, \underline{n_2}\rfloor,$$

kde $q \in W - F$.

Nedeterministicky zvolíme $q = r_1$ a provedeme derivační krok:

$$\lceil \underline{S}, s\rceil\langle s, 1\rangle\lfloor S, \underline{s}\rfloor \Rightarrow \lceil \underline{S}, s\rceil\lceil \underline{N}, n_2\rceil\lceil \underline{A}, r_1\rceil\langle n_2, 1\rangle\lfloor N, \underline{n_2}\rfloor\lfloor S, \underline{s}\rfloor.$$

Derivační krok 3

Protože gramatika Q v odpovídajícím derivačním kroku přechází do stavu r_1, měla by i gramatika H přejít do stavu $\langle r_1, 1\rangle$. Všechna žádoucí aplikovatelná pravidla mají tedy tvar:

$$\langle n_2, 1\rangle \to \lceil \underline{N}, r_1\rceil\langle r_1, 1\rangle\lfloor b, \underline{r_1}\rfloor,$$

kde $b \in V - T$.

Aplikované pravidlo musí navíc do větné formy doplnit R-symbol komplementární k symbolu $\lceil \underline{A}, r_1\rceil$. Nedeterministická volba stavu q v předchozím derivačním kroku tedy byla správná. Zvolíme $b = A$ a provedeme derivační krok:

$$\ldots \Rightarrow \lceil \underline{S}, s\rceil\lceil \underline{N}, n_2\rceil\lceil \underline{A}, r_1\rceil\lceil \underline{N}, r_1\rceil\langle r_1, 1\rangle\lfloor A, \underline{r_1}\rfloor\lfloor N, \underline{n_2}\rfloor\lfloor S, \underline{s}\rfloor.$$

Derivační krok 4

Ve stavu $\langle r_1, 1\rangle$ lze provést jediný derivační krok, kterým je přepis speciálního symbolu $\bar{L}$. Tento krok provedeme:

$$\ldots \Rightarrow \lceil \underline{S}, s\rceil\lceil \underline{N}, n_2\rceil\lceil \underline{A}, r_1\rceil\lceil \underline{N}, r_1\rceil\bar{L}\langle r_1, 2\rangle\lfloor A, \underline{r_1}\rfloor\lfloor N, \underline{n_2}\rfloor\lfloor S, \underline{s}\rfloor.$$

Derivační krok 5

Aby mohlo dojít k eliminaci, je nutné přidat do větné formy speciální symbol $\bar{R}$. Jediné pravidlo gramatiky H s tímto symbolem je vytvořeno v 5. fázi konstrukce z pravidla (N, r_2, ε, f) a má tvar:

$$\langle r_2, 2\rangle \to \varepsilon\bar{R}.$$

Proto by v tomto derivačním kroku měla gramatika H vygenerovat terminál a a přejít do stavu $\langle r_2, 2\rangle$. Gramatika Q navíc v příslušném derivačním kroku také přechází do stavu r_2. Současně s tím musí pravidlo doplnit do větné formy R-symbol, který je komplementární k L-symbolu $\lceil \underline{N}, r_1\rceil$, aby byla zajištěna úspěšnost eliminace.

Ale aplikovatelná pravidla v tomto kroku jsou:

$$\begin{aligned}
\langle r_1, 2\rangle &\to a\langle r_1, 2\rangle \lfloor S, \underline{r_1}\rfloor,\\
\langle r_1, 2\rangle &\to a\langle r_1, 2\rangle \lfloor A, \underline{r_1}\rfloor,\\
\langle r_1, 2\rangle &\to a\langle r_1, 2\rangle \lfloor N, \underline{r_1}\rfloor,\\
\langle r_1, 2\rangle &\to a\langle r_2, 2\rangle \lfloor S, \underline{r_2}\rfloor,\\
\langle r_1, 2\rangle &\to a\langle r_2, 2\rangle \lfloor A, \underline{r_2}\rfloor,\\
\langle r_1, 2\rangle &\to a\langle r_2, 2\rangle \lfloor N, \underline{r_2}\rfloor.
\end{aligned}$$

Z toho již vidíme, že není možné aplikovat pravidlo splňující obě podmínky současně. Pokud však gramatika H v tomto derivačním kroku nepřejde do stavu $\langle r_2, 2\rangle$, nemůže dojít k eliminaci, a proto nedeterministicky zvolíme neterminál N a provedeme derivační krok:

$$\ldots \Rightarrow \lceil \underline{S}, s\rceil \lceil \underline{N}, n_2\rceil \lceil \underline{A}, r_1\rceil \lceil \underline{N}, r_1\rceil \bar{L} a \langle r_2, 2\rangle \lfloor N, \underline{r_2}\rfloor \lfloor A, \underline{r_1}\rfloor \lfloor N, \underline{n_2}\rfloor \lfloor S, \underline{s}\rfloor.$$

Derivační krok 6

Ve stavu $\langle r_2, 2\rangle$ lze provést jediný derivační krok, kterým je přepis speciálního symbolu $\bar{R}$. Tento krok provedeme:

$$\ldots \Rightarrow \lceil \underline{S}, s\rceil \lceil \underline{N}, n_2\rceil \lceil \underline{A}, r_1\rceil \lceil \underline{N}, r_1\rceil \bar{L} a \bar{R} \lfloor N, \underline{r_2}\rfloor \lfloor A, \underline{r_1}\rfloor \lfloor N, \underline{n_2}\rfloor \lfloor S, \underline{s}\rfloor.$$

Derivační krok 7

Větná forma nyní sice obsahuje oba speciální symboly $\bar{L}$ a $\bar{R}$ a může tak nastat eliminace pomocí kontextových pravidel. Nicméně hned v prvním kroku narazíme na problém, protože nejpravější L-symbol $\lceil \underline{N}, r_1\rceil$ a nejlevější R-symbol $\lfloor N, \underline{r_2}\rfloor$ nejsou komplementární.

Důsledek

Neexistuje sekvence derivačních kroků gramatiky H, která by derivovala řetězec a (pro ostatní možné sekvence derivačních kroků viz přiložené médium).

Z toho tedy plyne, že gramatika Q a gramatika H nepopisují stejný jazyk. To znamená, že převod frontové gramatiky na $\mathrm{SCG_n}$ popsaný v článku [6] je chybný.

5.6.1 Navrhované řešení

Výše popsanou chybu v konstrukci $\mathrm{SCG_n}$ se zatím nepodařilo zcela vyřešit, nicméně se podařilo nalézt řešení, které tuto chybu alespoň částečně eliminuje. Oprava spočívá v rozdělení fáze 2 při konstrukci $\mathrm{SCG_n}$ v podsekci 2.2.1 na tři dílčí fáze podle délky řetězce y:

2.1. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $p, q \in W - F$, $y = \varepsilon$, pak $\langle q, 1\rangle \to \langle p, 1\rangle \lfloor b, \underline{p}\rfloor$ patří do P_1 pro všechna $b \in V - T$;

2.2. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $p, q \in W - F$, $y \in (V - T)^*$, $|y| = 1$, pak $\langle q, 1\rangle \to \lceil \underline{y}, q_1\rceil \langle p, 1\rangle \lfloor b, \underline{p}\rfloor$ patří do P_1 pro všechna $q_1 \in W - F$ a všechna $b \in V - T$;

2.3. Pokud $(a, q, y, p) \in R$, kde $a \in V - T$, $p, q \in W - F$, $y \in (V - T)^*$, $|y| > 1$ a $y = a_1 a_2 \cdots a_{|y|}$, pak $\langle q, 1\rangle \to \lceil \underline{a_1}, q_1\rceil \lceil \underline{a_2}, q_2\rceil \cdots \lceil \underline{a_{|y|}}, q_{|y|}\rceil \langle p, 1\rangle \lfloor b, \underline{p}\rfloor$, kde $q_1 = p$, patří do P_1 pro všechna $q_2 q_3, \ldots, q_{|y|} \in W - F$ a všechna $b \in V - T$.

Tato úprava zajistí konstrukci nových pravidel, která gramatikám umožní umístit do větné formy potřebné L-symboly, aby bylo možné simulovat derivaci řetězce ve frontové gramatice. Nová pravidla pro všechny nekoncové stavy jsou však vytvářena pouze pro pravidla, kde $|y| = 1$. Je to proto, že pouhé odstranění podmínky $q_1 = p$, a tedy tvorba pravidel pro všechny kombinace nekoncových stavů nehledě na délku řetězce y, opravdu markantně zvyšuje počet pravidel $\mathrm{SCG_n}$ do bodu, kdy se derivace řetězců pro aplikaci stává velmi obtížnou i s použitím heuristiky. Je tedy možné, že odstranění podmínky $q_1 = p$ opravuje algoritmus pro větší procento gramatik a řetězců než navrhované řešení, nicméně kvůli množství pravidel nelze většinu těchto gramatik pomocí aplikace vůbec testovat.

Vedlejším účinkem této úpravy bohužel je, že složitější $\mathrm{SCG_n}$ mohou zneužít nových pravidel k tomu, aby přijaly řetězce, které původní frontová gramatika nepřijímá. Dalším problémem je značné zvýšení počtu pravidel, protože frontové gramatiky ve 2. normální formě se vyznačují velkým počtem stavů.

I přes tyto problémy je však oprava v aplikaci implementována (viz podsekci 4.2.4), protože řeší problém alespoň pro jednodušší gramatiky, na které je aplikace zaměřena.

Kapitola 6

Závěr

Cílem práce bylo napsat aplikaci, která bude demonstrovat závěry článku [6] na konkrétních gramatikách, a současně najít heuristiku, která zvýší efektivitu algoritmické derivace řetězců v těchto gramatikách. Aplikace byla navržena a implementována v jazyce C, je spouštěna z příkazové řádky a používá textové uživatelské rozhraní. K derivaci řetězce je využíván algoritmus prohledávání do hloubky se zpětným navracením, který využívá navrženou heuristiku ke zvýšení výkonnosti.

Aplikace byla testována na 14 různých gramatikách a navržená heuristika byla porovnávána se základním algoritmem prohledávání do hloubky se zpětným navracením. Testování ukázalo, že navržená heuristika zvyšuje výkonnost algoritmu a současně nezpůsobuje nepřijetí žádného řetězce, který byl přijat základním algoritmem. Z testování se bohužel nepodařilo experimentálně vyhodnotit časovou složitost algoritmu, ale byla odvozena od složitosti použitého základního algoritmu. Časová složitost implementovaného algoritmu tedy zůstává exponenciální, ale z testování vyplynulo značné zlepšení oproti prohledávání do hloubky se zpětným navracením bez heuristiky.

Během testování byly odhaleny dvě chyby v článku [6]. Chyba nalezená v algoritmu převodu frontové gramatiky do první normální formy, který byl převzat z článku [5], nemá přímý vliv na výsledky této práce. Bohužel se nepodařilo najít žádné ani částečné řešení této chyby.

Chybu v konstrukci gramatik s rozptýleným kontextem s jedním kontextovým pravidlem, která je popsaná v článku [6], se podařilo částečně eliminovat. K částečné eliminaci chyby pomohla i navržená aplikace, díky které bylo možné podrobně zkoumat gramatiky, které by jinak byly příliš rozsáhlé. Upravený algoritmus je implementován v aplikaci, která díky tomu funguje správně pro jednodušší gramatiky, nicméně chyba přetrvává u složitějších gramatik. I přesto má chybná konstrukce přímý vliv na výsledky práce a nebylo by vhodné pokračovat dokud nebude konstrukce opravena.

Potom by pokračováním této práce mohlo být rozšíření aplikace o grafické uživatelské rozhraní. Navržená heuristika sice úspěšně redukuje stavový prostor, ale bylo by vhodné ji rozšířit o mechanismus, který automaticky určí maximální hloubku prohledávání a nebude tak nutné zadávat tuto hodnotu ručně. Dále by heuristika mohla být rozšířena o informovanou volbu pravidel, která by v ideálním případě mohla zajistit optimálnost metody.

Literatura

[1] El Mouelhi, A., Jegou, P., Terrioux, C. a Zanuttini, B. On the Efficiency of Backtracking Algorithms for Binary Constraint Satisfaction Problems. In: *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2012), Fort Lauderdale, Florida, USA*. Leden 2012. Dostupné z: http://www.cs.uic.edu/pub/Isaim2012/WebPreferences/ISAIM2012_ElMouelhi_etal.pdf.

[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] International, E. *ECMA-404: The JSON Data Interchange Syntax* [online]. Standard, 2. vyd. Rue du Rhône 114 CH-1204 Geneva: ECMA International, prosinec 2017 [cit. 2021-4-21]. Dostupné z: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf.

[4] Jirák, O. Table-Driven Parsing of Scattered Context Grammar. In: *Proceedings of the 16th Conference Student EEICT 2010 Volume 5*. Brno, CZ: Faculty of Information Technology BUT, 2010, s. 171–175. ISBN 978-80-214-4080-7. Dostupné z: https://www.fit.vut.cz/research/publication/9211.

[5] 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.

[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. 1–24. ISSN 0169-2968. Dostupné z: https://www.fit.vut.cz/research/publication/11559.

Příloha A

Manuál

Aplikace vytvořená v této práci slouží k demonstraci gramatik s rozptýleným kontextem a jedním kontextovým pravidlem. Překlad se prování příkazem `make`, který využívá překladač `gcc`.

A.1 Přepínače

Použití z příkazové řádky je následující:

```
demonstrate-scg [-options] [-j path] [-o path] [-d max_depth]
                [-q max_depth] sentence.
```

Jediný povinný argument je tedy `sentence` obsahující řetězec, který má být zpracován.

Aplikace navíc může být konfigurována následujícími přepínači:

- Přepínač `-h` pouze zobrazí krátký návod k použití aplikace;
- Přepínač `-j`, kde `path` je cesta k souboru ve formátu JSON, ze kterého má být čtena vstupní frontová gramatika. Pokud přepínač není nastaven bude aplikace číst ze standardního vstupu.
- Přepínač `-o`, kde `path` je cesta k souboru, do kterého bude tisknout. Pokud přepínač není nastaven bude aplikace tisknout na standardní výstup.
- Přepínač `-n` normalizuje vstupní gramatiku před její konverzí na gramatiku s rozptýleným kontextem;
- Přepínač `-s` zjednoduší názvy stavů po normalizaci. Každý stav bude pojmenován jedním malým písmenem anglické abecedy nebo případně písmenem a jednou číslicí. Tento přepínač nemá žádný efekt, pokud není současně zadán přepínač `-n`;
- Přepínač `-m` změní formát výstupu na HTML místo čistého textu;
- Přepínač `-a` zakáže tisk v kódování unicode, tištěny budou pouze znaky v kódování ASCII. Tento přepínač funguje pouze, pokud není současně zadán přepínač `-m`;
- Přepínač `-e` zakóduje neterminály a při derivaci používá jedno kontextové pravidlo;
- Přepínač `-u` nechá výběr pravidla v každém kroku derivace na uživateli. Stále však zobrazuje, které pravidlo by zvolila navržená heuristika. Rozhraní pro výběr pravidel je vždy tištěno na standardní výstup nezávisle na nastavení přepínače `-o`;

- Přepínač `-b` použije pro výběr pravidel základní heuristiku, která vybírá vždy první dostupné pravidlo. Tento přepínač nelze nemá žádný efekt pokud je současně zadán přepínač `-u`;
- Přepínač `-d`, kde `max_depth` je kladné celé číslo, nastavuje maximální délku větné formy[1]při derivaci řetězce. Výchozí hodnota je 10;
- Přepínač `-q`, kde `max_depth` je kladné celé číslo, nastavuje maximální délku větné formy[2]při derivaci řetězce ve frontové gramatice, pro ověření správnosti. Výchozí hodnota je vypočítána z délky řetězce `sentence`.

A.2 Návratové kódy

Aplikace může skončit s jedním z následujících návratových kódů:

- Kód 0 značí úspěch, tedy že nedošlo za běhu k žádné chybě;
- Kód 10 značí chybu při práci s pamětí (typicky selhání alokace paměti);
- Kód 11 značí chybu při zpracování argumentů příkazové řádky (typicky neznámý přepínač nebo špatný počet argumentů);
- Kód 12 značí nevalidní vstupní gramatiku (například neznámé symboly v pravidlech gramatiky, atp);
- Kód 13 značí chybu během normalizace gramatiky, konkrétně, že se nepodařilo zjednodušit jména normalizovaných stavů.
- Kód 14 značí interní chybu v aplikaci. (například neočekávaná hodnota `NULL`).

Dále může aplikace skončit s jedním z chybových návratových kódů použité knihovny JSMN pro zpracování formátu JSON:

- Kód -3 značí, že na vstupu aplikace nejsou kompletní data ve formátu JSON (chybí závorka }, atp);
- Kód -2 značí, že data nejsou ve validním formátu JSON (obsahují nepovolené znaky, atp);
- Kód -1 značí, že během zpracování dat ve formátu JSON došla paměť.

A.3 Chybová hlášení

Chybová hlášení jsou vypisována na standardní chybový výstup a dělí se podle závažnosti (seřazeno od nejméně závažného):

1. `INFO` pouze informuje uživatele o nastalé události, která typicky není negativního charakteru;

[2]Délkou větné formy gramatiky s rozptýleným kontextem se myslí počet symbolů ve tvaru $\lceil a, q \rceil$ ve větné formě nebo počet symbolů ve $\lfloor a, q \rfloor$ ve větné formě. Počítá se vyšší z těchto dvou hodnot.

[2]Maximální délkou větné formy frontové gramatiky se myslí počet symbolů ve větné formě (nepočítá se stav).

2. `WARNING` upozorňuje uživatele na negativní událost, která ale nebrání běhu programu (ale může způsobit jiné chování než by uživatel čekal);

3. `ERROR [kód]` upozorňuje uživatele na chybu, kvůli které není možné pokračovat dále v běhu a aplikace je ukončena s vypsaným návratovým kódem;

4. `FATAL ERROR [kód]` rovněž upozorňuje uživatele na chybu, kvůli které není možné pokračovat dále v běhu. V tomto případě je ale aplikace ukončena s vypsaným návratovým kódem okamžitě, což znamená že nemusí dojít k uvolnění veškeré alokované paměti.

A.4 Formát vstupu

Vstupní frontová gramatika je zadávána ve formátu JSON.

Gramatika je v tomto zapsána jako jediný objekt s 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 jsou zapisovány jako řetězce o délce jedna, které mohou obsahovat pouze velké nebo malé písmeno anglické abecedy. Stavy jsou zapisovány opět jako řetězce o délce jedna obsahující jedno velké nebo malé písmeno anglické abecedy nebo jako řetězce o délce dva obsahující jedno písmeno anglické abecedy následované právě jednou číslicí.

Pravidla jsou zapisována ve tvaru pole o čtyřech položkách v přesně daném pořadí odpovídajícím formálnímu zápisu pravidla. Zapisovaný řetězec symbolů ve třetí položce může mít délku maximálně 30 znaků. Pro zápis prázdného řetězce se používá `""`.

Aplikace čte soubor pouze po konec prvního objektu, a proto můžou být za závorkou } libovolné komentáře či poznámky aniž by to způsobilo chybu při zpracování dat.

A.4.1 Příklad

Například gramatika $Q = \{V, T, W, \{f\}, As_0, R\}$, kde $V = \{A, a\}$, $T = \{a\}$, $W = \{s_0, f\}$ a $R = \{(A, s_0, aA, s_0), (A, s_0, \varepsilon, f), (a, s_0, a, s_0)\}$, bude vyjádřena zapsána ve tvaru ve výpisu A.1.

```
{
    "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"]
    ]
}
```

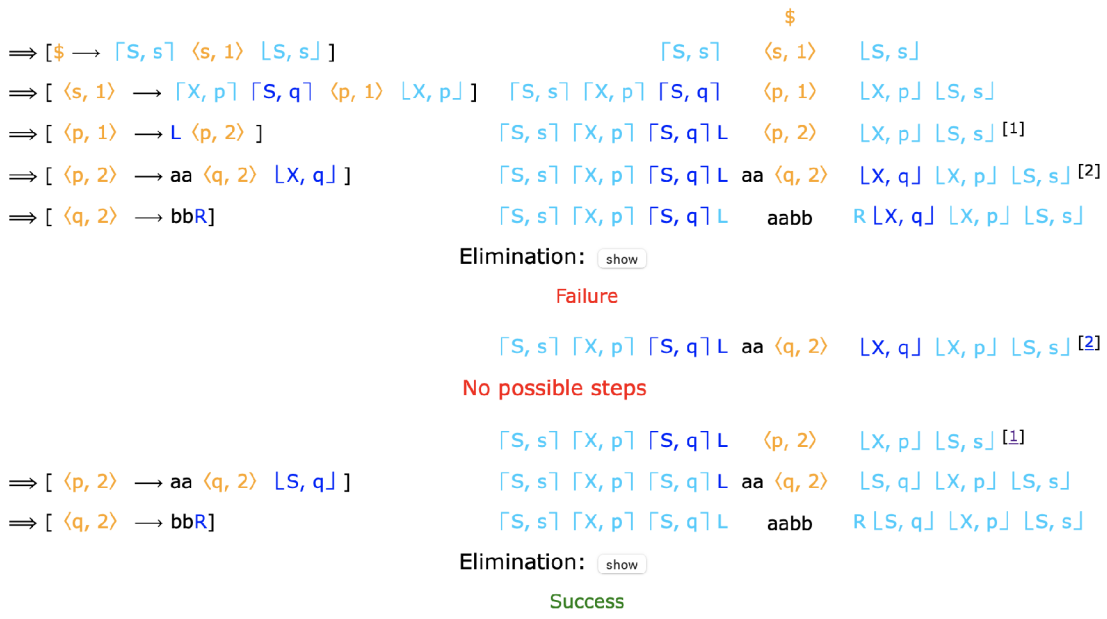
Výpis A.1: Příklad vstupu ve formátu JSON

A.5 Formát výstupu

Aplikace vypisuje výsledky v čistém textu nebo ve formátu HTML. Při výpisu je používán speciální symbol $ pro počáteční symbol $\mathrm{SCG_n}$. Mimoto je používán speciální symbol \ označující koncový stav f frontové gramatiky po normalizaci.

A.5.1 HTML formát

Jak je vidět na obrázku A.1, každý derivační krok je označen na začátku řádku pomocí ⇒, za ní následuje v hranatých závorkách pravidlo použité během tohoto derivačního kroku. Tedy tímto pravidlem byla z větné formy na předchozím řádku získána větná forma následující za tímto derivačním krokem na tomto řádku. Pokud se algoritmus vrátí zpátky do větné formy, která již byla vypsána, všechny výskyty této větné formy jsou označeny stejným horním indexem.



Obrázek A.1: Příklad prezentace derivace řetězce `aabb` ve formátu HTML.

Symboly označující stavy jsou odlišeny oranžovou barvou, zatímco ostatní symboly jsou modré. Ostatní symboly, které nemají na druhé větné formy odpovídající symbol jsou zvýrazněny tmavým odstínem modré, zatímco zbylé symboly jsou odlišeny světlejším odstínem.

A.5.2 Čistý text

Jak ukazuje výpis A.2, výstup v čistém textu nezapisuje pravidla použitá pravidla v derivačním kroku do hranatých závorek, ale místo toho zapisuje použité pravidlo vždy mezi dva řádky s větnými formami. Větné formy zde nejsou, na rozdíl od formátu HTML, označovány indexy.

```
=> [S, s]<s, 1>[S, s]
Used rule: <s, 1> -> [X, p][S, q]<p, 1>[X, p]

=> [S, s][X, p][S, q]<p, 1>[X, p][S, s]
```

Výpis A.2: Ukázka výstupu v čistém textu v kódování ASCII

Kódování ASCII

Při výstupu v kódování ASCII jsou speciální znaky v kódování unicode převedeny způsobem znázorněným v tabulce A.1.

Počáteční symbol gramatiky s rozptýleným kontextem je $. Při převodu gramatiky do první normální formy je vytvořen nový neterminál 1 a nový koncový stav \. Podtržení symbolů v gramatice s rozptýleným kontextem bylo vypuštěno.

Formální zápis	ASCII
ε	_
$\langle$	<
$\rangle$	>
$\lceil$ nebo $\lfloor$	[
$\rceil$ nebo $\rfloor$	]

Tabulka A.1: Reprezentace matematických symbolů v ASCII.

A.5.3 Statistiky

Na konci derivace vypisuje aplikace následující statistiky značící úspěšnost heuristiky.

- Konzistentní validita platí, jestliže byl řetězec přijat oběma gramatikami nebo odmítnut oběma gramatikami. Oběma gramatikami se myslí vstupní frontová gramatika (po případné normalizaci) a z ní odvozená gramatika s rozptýleným kontextem;
- Počet rozhodnutí říká kolikrát algoritmus vybíral pravidlo k aplikaci;
- Optimální délka říká kolik kroků má výsledná derivace řetězce, tím pádem tedy kolikrát by se algoritmus rozhodl pokud by každé jeho rozhodnutí bylo správné;
- Úspěšnost počítána jako $\frac{\text{optimální délka}}{\text{počet rozhodnutí}}$;
- Počet zpětných navrácení je počet případů, kdy nemohlo být aplikováno žádné pravidlo a algoritmus se tak vrátil o úroveň výš.

A.6 Výběr pravidla uživatelem

Při výběru pravidla uživatelem je uživateli, ve formátu ukázaném ve výpise A.3, zobrazena

- Současná větná forma;
- Neterminál, který by mělo aplikované pravidlo doplnit do větné formy;
- Indexovaný seznam pravidel, která lze na současnou větnou formu aplikovat;
- Index pravidla doporučeného heuristikou.

Uživatel následně zadává index pravidla, které má být aplikováno. Pokud uživatel zadá index neodpovídající žádnému pravidlu (například -1), pak nebude aplikováno žádné pravidlo, a algoritmus se vrátí o úroveň výš.

Pravidla jsou zobrazena ve tvaru `index (priorita): pravidlo`. Prioritu pravidla určuje heuristika, která podle těchto priorit následně pravidlo vybírá.

```
---------------------------------
Current form: [S, s][X, p][S, q]L<p, 2>[X, p][S, s]
To match: [S, q]

0 (1): <p, 2> -> aa<q, 2>[S, q]
1 (0): <p, 2> -> aa<q, 2>[X, q]

Suggested rule to use: 0
Rule to use: _
```

Výpis A.3: Ukázka výběru pravidla uživatelem.

Příloha B

Obsah přiloženého média

Přiložené médium obsahuje:

1. písemnou zprávu ve formátu PDF,
2. zdrojový tvar písemné zprávy,
3. zdrojové texty aplikace,
4. testované gramatiky ve formátu JSON,
5. kompletní výsledky testování ve formátu CSV,
6. doplňující data k příkladům chyb v článcích.

Soubory s výsledky testování jsou pojmenovány například `example1-impr-d15-s4.csv`, kde `example1` značí testovanou gramatiku, `d15` značí, že maximální hloubka prohledávání je 15 a `s4` značí, že maximální délku řetězců je 4. `impr` znamená použití navržené heuristiky, zatímco `base` značí použití základní heuristiky.