VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘEVOD GRAMATIK DO NORMÁLNÍCH FOREM

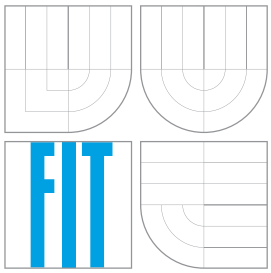
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MIRKA KLAPUCHOVÁ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PŘEVOD GRAMATIK DO NORMÁLNÍCH FOREM

TRANSFORMATION OF GRAMMARS INTO NORMAL FORMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIRKA KLAPUCHOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR ZEMEK
SUPERVISOR

BRNO 2012

Abstrakt

Práce se zabývá problematikou normálních forem z teorie formálních jazyků. Jsou zde uvedeny základní pojmy z této oblasti, dále různé typy gramatik a především normální formy a algoritmy pro převod gramatik do těchto forem. Součástí práce je popis návrhu a implementace programu, který slouží k převodu vstupní gramatiky do zadané normální formy.

Abstract

This thesis deals with issues of normal forms from theory of formal languages. Basic terms from this area are listed here, different types of grammars as well and especially normal forms and algorithms that transfer grammars into normal forms. Description of design and implementation of program which is used to transfer input grammar into entered normal form is included.

Klíčová slova

Formální jazyky, bezkontextová gramatika, kontextová gramatika, neomezená gramatika, Chomského normální forma, Greibachové normální forma, Kurodova normální forma, Geffertova normální forma, Pentonnenova normální forma.

Keywords

Formal languages, context-free grammar, context-sensitive grammar, unrestricted grammar, Chomsky normal form, Greibach normal form, Kuroda normal form, Geffert normal form, Pentonnen normal form.

Citace

Mirka Klapuchová: Převod gramatik do normálních forem, bakalářská práce, Brno, FIT VUT v Brně, 2012

Převod gramatik do normálních forem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Petra Zemka.

.....
Mirka Klapuchová
16. května 2012

Poděkování

Nejdříve bych chtěla poděkovat svému vedoucímu práce, panu Ing. Petrovi Zemkovi, za velice užitečné odborné rady, vstřícnost, se kterou se mnou vždy jednal, a obdivuhodnou rychlost reakcí na mé dotazy. Také děkuji své nejbližší rodině a mému příteli za podporu při dokončování této práce.

Obsah

Kapitola 1

Úvod

V dnešní době dochází k velkému rozšiřování oboru informačních technologií. Jednotlivá odvětví patřičně rostou a vznikají další nová, která je třeba zkoumat. Jedním z takových odvětví je teorie formálních jazyků. Jedná se o velice rozsáhlý obor, jehož komplexní popis by vydal na obrovské množství stran [7]. Tato práce se zaměřuje na jednu z jeho částí, a tou jsou normální formy. Jsou to speciální formy gramatik, kde všechna pravidla jsou v určitém tvaru, přičemž těchto tvarů může být vícero. Dále uvádí jednotlivé důležité definice a vysvětluje dané pojmy. Cílem této práce je seznámit se s normálními formami, demonstrovat jejich význam v praxi a v neposlední řadě navrhnout, implementovat a otestovat program pro převod gramatik do těchto forem.

Normálními formami je dobré se zabývat hned z několika důvodů. Jednak mohou zjednodušit matematické důkazy, protože jsou předem dány tvary pravidel, se kterými se pracuje. Dále některé analýzy (např. precedenční analýza), jenž jsou využívány v praxi, předpokládají již upravenou gramatiku. Samotný program pro převod pak mohou využít studenti jako didaktickou pomůcku k ověření, zda se jim podařilo vlastními silami správně převést gramatiku do některé normální formy.

Úvodní kapitola čtenáře seznámí se základními pojmy z teorie formálních jazyků. Jsou zde popsány úplně elementární prvky, které postupně expandují do větších celků. Nejprve jsou tedy uvedeny ty jednodušší pojmy, poté jsou popsány gramatiky, které jsou rozděleny na bezkontextové, neomezené a kontextové gramatiky.

Následuje samostatná kapitola o normálních formách. Vysvětluje formát jednotlivých normálních forem a uvádí algoritmy, které převedou zadanou vstupní gramatiku do dané normální formy.

Kapitola o návrhu a implementaci programu pak popisuje postup při jeho vytváření, myšlenky a cíle, kterých bylo nutné dosáhnout, a především způsob vzniku samotného programu.

Na to navazuje kapitola o testování výsledného programu, kde jsou popsány jednotlivé prováděné testy, včetně ukázek takovýchto testů a jejich výstupů.

Závěrem jsou pak shrnuty a zhodnoceny dosažené výsledky a je diskutován další možný vývoj projektu.

Kapitola 2

Základní pojmy

Tato kapitola je věnována základním pojmům a jejich definicím z oblasti teorie formálních jazyků. Tyto pojmy budou následně v textu často využívány. Slouží tedy ke stručnému úvodu pro čtenáře, kteří se v této oblasti příliš nepohybují, a zároveň jako lehké osvěžení paměti pro znalé. Zvláštní sekce je pak věnována gramatikám bezkontextovým, neomezeným a kontextově závislým. Definice jsou převzané z [4], [6] a [10].

2.1 Abeceda, řetězec, jazyk

Definice 2.1.1. *Abeceda* je konečná, neprázdná množina elementárních *symbolů*.

Definice 2.1.2. Nechť Σ je abeceda. *Řetězec* nad abecedou Σ je posloupnost symbolů abecedy Σ, definováno následovně:

- ε značí prázdný řetězec,
- pokud x je řetězec nad Σ a $b \in \Sigma$, tak i bx je řetězec nad Σ.

Definice 2.1.3. Nechť x je řetězec nad abecedou Σ. *Délka řetězce* x, značena $|x|$, je definována následovně:

- pokud $x = \varepsilon$, pak $|x| = 0$
- pokud $x = a_1 \dots a_n$, pak $|x| = n$ pro $n \geq 1$ a $a_i \in \Sigma$ pro všechna $i = 1, \dots, n$.

Definice 2.1.4. Nechť x a y jsou dva řetězce. *Konkatenace* těchto dvou řetězců je definována jako xy.

Definice 2.1.5. Nechť Σ je abeceda. Pak Σ^* značí *množinu všech řetězců nad* Σ. Σ^+ je množina $\Sigma^* - \{\varepsilon\}$.

Definice 2.1.6. *Jazyk* je každá podmnožina $L \subseteq \Sigma^*$, kde Σ je abeceda.

Jazyk si z množiny všech řetězců nad abecedou Σ vybírá takové, které pro něj mají nějaký význam. I český jazyk obsahuje pouze řetězce (slova), které něco znamenají, a to přesto, že z české abecedy lze sestavit nekonečně mnoho řetězců.

2.2 Gramatiky

V teorii formálních jazyků existují různé modely pro popis jazyků. V této práci se zaměříme na gramatiky bezkontextové, kontextové, neomezené a jejich normální formy. Gramatiky jsou prostředkem pro popis daného jazyka, určují, jak je možné řadit jednotlivé symboly za sebou, aby takto sestavené věty splňovaly syntaktické požadavky na jazyk. Zároveň tak definují z pohledu popisovaného jazyka i určitou smysluplnost věty.

2.2.1 Bezkontextová gramatika

Definice 2.2.2. *Bezkontextová gramatika* je čtveřice ve tvaru $G = (N, T, P, S)$, kde

- N je abeceda *neterminálů*,
- T je abeceda *terminálů* takových, že platí $N \cap T = \emptyset$,
- P je konečná množina *přepisovacích pravidel* ve tvaru $A \to x$, kde $A \in N$ a $x \in (N \cup T)^*$,
- $S \in N$ je počáteční symbol.

Neterminály zastupují pomocné prvky, které nejsou součástí jazyka a lze je pravidly přepsat. Terminály jsou již přímo elementy jazyka a objevují se pouze na pravé straně pravidel – přepsat je nelze. Tyto gramatiky se nazývají bezkontextové, neboť při přepisování neterminálů nezáleží na řetězcích či symbolech nalevo a napravo od nich.

Definice 2.2.3. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika, $p : A \to z \in P$ a $x, y \in (N \cup T)^*$. Pak xAy *přímo derivuje* xzy za použití p, zapsáno jako $xAy \Rightarrow xzy\ [p]$ nebo zjednodušeně $xAy \Rightarrow xzy$.

Definice 2.2.4. Pokud $xAy \Rightarrow xxy$ v G, můžeme říct, že G provádí *derivační krok* z xAy do xxy.

Derivační krok u bezkontextových gramatik je nahrazení neterminálu v řetězci, který se vyskytuje na levé straně přepisovacího pravidla, příslušným řetězcem, vyskytujícím se na pravé straně tohoto pravidla.

Příklad 2.2.5. Příkladem velmi jednoduché bezkontextové gramatiky může být následující, která generuje správně párované závorky. Mějme neterminál $B \in N$, terminály $(,) \in T$ a počáteční symbol B. Pravidla jsou:

1. $B \to (B)$
2. $B \to \varepsilon$

První pravidlo generuje vždy dvě závorky, levou a pravou, a zajišťuje, aby se další případné závorky zapsaly dovnitř těchto vygenerovaných. K tomu by bylo využito stejného pravidla. Pokud již nebudeme chtít generovat závorky, aplikujeme druhé pravidlo, čímž se neterminál B nahradí prázdným řetězcem.

Definice 2.2.6. Nechť $u \in (N \cup T)^*$. G provede *nula derivačních kroků* z u do u; zapisujeme: $u \Rightarrow^0 u\ [\varepsilon]$ nebo zjednodušeně $u \Rightarrow^0 u$.

Definice 2.2.7. Nechť $u_0, \ldots, u_n \in (N \cup T)^*, n \geq 1$ a $u_{i-1} \Rightarrow u_i\ [p_i], p_i \in P$ pro všechna $i = 1, \ldots, n$, což znamená: $u_0 \Rightarrow u_1\ [p_1] \Rightarrow u_2\ [p_2] \Rightarrow \cdots \Rightarrow u_n\ [p_n]$. Pak G provede n *derivačních kroků z* u_0 do u_n; zapisujeme: $u_0 \Rightarrow^n u_n\ [p_1 \ldots p_n]$ nebo zjednodušeně $u_0 \Rightarrow^n u_n$. Pokud $u_0 \Rightarrow^n u_n\ [\pi]$ pro nějaké $n \geq 1$, pak u_0 *derivuje* u_n v G, zapisujeme: $u_0 \Rightarrow^+ u_n\ [\pi]$. Pokud $u_0 \Rightarrow^n u_n\ [\pi]$ pro nějaké $n \geq 0$, pak u_0 *derivuje* u_n v G, zapisujeme: $u_0 \Rightarrow^* u_n\ [\pi]$.

Příklad 2.2.8. Mějme bezkontextovou gramatiku $G = (N, T, P, A)$, která obsahuje pravidla

1. $A \to xB$
2. $B \to x$
3. $B \to \varepsilon$

kde $A, B \in N$ a $x \in T$. Nechť xxA je řetězec. Pak xxA derivuje $xxxB$ za použití pravidla číslo 1. Tento krok můžeme zapsat jako $xxA \Rightarrow xxxB$. A je počáteční symbol, začali jsme tedy s nahrazováním tohoto neterminálu. V původním řetězci se navíc žádný jiný neterminál neobjevil, tedy ani není co jiného přepsat. Po přepsání A na xB se v řetězci vysktytl neterminál B. Další postup by mohl spočívat v použití pravidla 2 nebo 3, která obě přepisují právě B. V případě jejich aplikace by tedy byl výsledný řetězec $xxxx$ nebo xxx.

Definice 2.2.9. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pokud $S \Rightarrow^* w$ v G, pak w je *větná forma* G. Taková větná forma w, kde $w \in T^*$, je *věta* generovaná pomocí G. Pak *jazyk generovaný gramatikou* G, $L(G)$, je množina všech vět, které G generuje. Formálně $L(G) = \{w \mid w \in T^* \text{ a } S \Rightarrow^* w \text{ v } G\}$.

Bezkontextové gramatiky generují tzv. bezkontextové jazyky.

Definice 2.2.10. Jazyk L je *bezkontextový*, existuje–li bezkontextová gramatika G taková, že $L(G) = L$.

2.2.11 Neomezená gramatika

Definice 2.2.12. *Neomezená gramatika* je čtveřice $G = (N, T, P, S)$, kde:

- N je abeceda neterminálů;
- T je abeceda terminálů a platí $N \cap T = \emptyset$;
- P je množina pravidel ve tvaru $x \to y$, kde $x \in (N \cup T)^+$ a $y \in (N \cup T)^*$;
- $S \in N$ je počáteční symbol.

Z definice vyplývá, že bezkontextová gramatika je v podstatě speciálním případem gramatiky neomezené. Rozdíl spočívá v tom, jaký je přípustný tvar levé strany pravidel. Neomezená gramatika může mít totiž na levé straně pravidla jak neterminály, tak terminály.

V následujícím textu si pro větší přehlednost označíme levou stranu pravidla p jako $\mathrm{lhs}(p)$ a pravou stranu $\mathrm{rhs}(p)$.

Definice 2.2.13. Nechť $G = (N, T, P, S)$ je neomezená gramatika, $p \in P$ a $x, y \in (N \cup T)^*$. Pak $x\mathrm{lhs}(p)y$ *přímo derivuje* $x\mathrm{rhs}(p)y$ pomocí pravidla p v G, zapisujeme: $x\mathrm{lhs}(p)y \Rightarrow x\mathrm{rhs}(p)y\ [p]$ nebo zjednodušeně $x\mathrm{lhs}(p)y \Rightarrow x\mathrm{rhs}(p)y$.

Obdobně jako v definici 2.2.7 definujeme $\Rightarrow^*$ a $\Rightarrow^+$ pro neomezené gramatiky.

Definice 2.2.14. Nechť $G = (N, T, P, S)$ je neomezená gramatika. Pokud $S \Rightarrow^* w$ v G, pak w je *větná forma* gramatiky G. Větná forma w, $w \in T^*$, reprezentuje větu generovanou pomocí G. *Jazyk generovaný neomezenou gramatikou* G, značený $L(G)$, je množinou všech vět, které G generuje. Formálně: $L(G) = \{w \mid w \in T^* \text{ a } S \Rightarrow^* w \text{ v } G\}$.

2.2.15 Kontextově závislá gramatika

Definice 2.2.16. Nechť $G = (N, T, P, S)$ je neomezená gramatika. G je *kontextově závislá gramatika*, pokud $p \in P$ implikuje $|\text{lhs}(p)| \leq |\text{rhs}(p)|$.

Kontextově závislé gramatiky jsou speciálním případem gramatik neomezených. Aby byla gramatika kontextově závislá, každé její pravidlo musí mít pravou stranu alespoň tak dlouhou jako levou. Tedy pokud jsou na levé straně 3 symboly, na pravé straně jich musí být 3 a více.

Existuje ještě jiná, alternativní definice kontextově závislé gramatiky, kterou lze najít například ve studijní opoře k předmětu Teoretická informatika [3] na straně č. 16.

Definice 2.2.17. Jazyk L je *kontextově závislý jazyk*, pokud existuje kontextově závislá gramatika G taková, že $L = L(G)$.

Kapitola 3

Normální formy gramatik

Gramatiky se dají převést na různé formy splňující určitá pravidla. Tyto převody se velmi dobře uplatní, pokud chceme například upravit gramatiku pro lepší a efektivnější implementaci v praxi. Těchto forem je poměrně mnoho, zde si uvedeme některé z nich. Než se dostaneme přímo k normálním formám gramatik, zmíníme převody, jejichž výstupy se dají pokládat za další formu. Často se tyto jednodušší formy používají jako mezivýsledek pro převody do normálních forem, v některých případech vstup algoritmu předpokládá již upravenou gramatiku. Kapitola vychází z [4].

3.1 Eliminace nepoužitelných symbolů

Při vytváření gramatik mohou vzniknout symboly, které jsou zbytečné. Ať už je to důsledkem nepozornosti, či se takové symboly objeví během převodu do některé normální formy. Odstranění těchto symbolů je velmi vhodné, neboť gramatiku zbytečně znepřehledňují a degradují.

3.1.1 Odstranění neukončujících symbolů

Definice 3.1.2. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika a $A \in N \cup T$. A je *ukončující*, pokud existuje $w \in T^*$ takové, že $A \Rightarrow^* w$ v G. Jinak je A *neukončující*.

Jinými slovy se dá říci, že ukončující symbol je takový symbol, ze kterého je možné postupnými derivacemi získat řetězec nad terminální abecedou. Jinak je symbol neukončující. Zbytečnost neukončujících symbolů je zřetelná. Není možné z nich vygenerovat žádný terminální řetězec, a tedy je dobré je odstranit. Jazyk generovaný touto gramatikou tento převod nikterak nenaruší.

Následující algoritmus najde v bezkontextové gramatice množinu všech ukončujících symbolů, již pak využijeme v algoritmu, který odstraní ty neukončující.

Algoritmus 3.1.3.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Množina V, obsahující všechny ukončující symboly z G.
Metoda:

begin
$V := T \cup \{A \mid A \to x \in P \text{ a } x \in T^*\}$;
repeat

$U := V$;
for všechny $p : A \to x \in P$, kde $A \notin U$ **do**
if $x \in U^+$
then $V := V \cup \{A\}$
until $U = V$
end.

Převod bezkontextové gramatiky na ekvivalentní bezkontextovou gramatiku bez všech neukončujících symbolů:

Algoritmus 3.1.4.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Bezkontextová gramatika $G_{term} = (N_{term}, T, P_{term}, S)$ taková, že $L(G) = L(G_{term})$ a $N_{term} \cup T$ obsahuje pouze ukončující symboly.
Metoda:

begin
použij algoritmus 3.1.3 k získání množiny V všech ukončujících symbolů v G;
$N_{term} := \{N \cap V\}$;
$P_{term} := \{p : B \to x \in P, B \in N_{term}, \text{ a } x \in V^*\}$;
produkuj $G_{term} = (N_{term}, T, P_{term}, S)$
end.

Nyní víme, jak odstranit všechny neukončující symboly. Mezi nepoužitelné symboly ovšem patří ještě jedna skupina, na kterou se zaměříme dále.

3.1.5 Odstranění nedostupných symbolů

Definice 3.1.6. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika a $X \in N \cup T$. X je *dostupný*, pokud $S \Rightarrow^* uXv$ v G, pro nějaké $u, v \in (N \cup T)^*$. Jinak je X *nedostupný*.

Pro pravidla gramatik nemají nedostupné symboly žádný smysl. Jedná se o ty symboly, kterých nelze dosáhnout postupnými derivacemi z počátečního symbolu. Takový symbol se tedy v generovaném řetězci nikdy neobjeví. Opět je zde vidět zbytečnost takových symbolů. Pomocí níže zmíněných algoritmů tedy dosáhneme toho, abychom ve výsledné gramatice měli pouze dostupné symboly.

V nadcházejícím algoritmu využijeme zápisu alph(x), který značí množinu symbolů, které obsahuje řetězec uvedený v závorkách – v našem případě tedy řetězec x.

Následující algoritmus vyhledá v bezkontextové gramatice všechny dostupné symboly:

Algoritmus 3.1.7.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Množina W, obsahující všechny dostupné symboly v G.
Metoda:

begin
$W := \{S\}$;
$STARE := \emptyset$;
repeat
$NOVE := W - (STARE \cup T)$;
$STARE := W - T$;

for všechny $A \in NOVE$ **do**
for každé $p : B \to x \in P$, kde $B = A$ **do**
$W := W \cup \{\text{alph}(x)\}$
until $W - T = STARE$
end.

Předchozí algoritmus využijeme v algoritmu, sloužícímu k převodu bezkontextové gramatiky na ekvivalentní bezkontextovou gramatiku bez nedostupných symbolů:

Algoritmus 3.1.8.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Bezkontextová gramatika $G_{acc} = (N_{acc}, T_{acc}, P_{acc}, S)$, taková, že $L(G) = L(G_{acc})$ a $(N_{acc} \cup T_{acc})$ obsahuje pouze dostupné symboly.
Metoda:

begin
použij algoritmus 3.1.7 k nalezení množiny W, obsahující všechny dostupné symboly v G;
$N_{acc} := \{N \cap W\}$;
$T_{acc} := \{T \cap W\}$;
$P_{acc} := \{p : B \to x \in P, B \in N_{acc}, x \in W^*\}$;
produkuj $G_{acc} = (N_{acc}, T_{acc}, P_{acc}, S)$
end.

Nyní přejdeme k odstranění všech symbolů, které se nikdy nepoužijí.

3.1.9 Odstranění nepoužitelných symbolů

Definice 3.1.10. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika a $X \in N \cup T$ je symbol. X je *použitelný*, pokud $S \Rightarrow^* uXv \Rightarrow^* uxv$ v G, pro nějaké $u, v \in (N \cup T)^*$ a $x \in T^*$. Jinak je X *nepoužitelný*.

V předchozích dvou podsekcích byly zmíněny dva typy symbolů – neukončující a nedostupné, včetně algoritmů, které zajistí jejich odstranění. Tyto dvě kategorie symbolů se nazývají nepoužitelné symboly. Opačný případ jsou pak ukončující a dostupné symboly, jenž se nazývají použitelné.

Algoritmus, sloužící k odstranění nepoužitelných symbolů, tedy spojuje algoritmy, jenž odstraní neukončující a nedostupné symboly:

Algoritmus 3.1.11.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Bezkontextová gramatika $G_{useful} = (N_{useful}, T_{useful}, P_{useful}, S)$, kde $L(G) = L(G_{useful})$ a $(N_{term} \cup T)$ obsahuje pouze použitelné symboly.
Metoda:

begin
použij algoritmus 3.1.4 ke konverzi $G = (N, T, P, S)$
na $G_{term} = (N_{term}, T, P_{term}, S)$;
použij algoritmus 3.1.8 ke konverzi $G_{term} = (N_{term}, T, P_{term}, S)$
na $G_{acc} = (N_{acc}, T_{acc}, P_{acc}, S)$;
$N_{useful} := N_{acc}$;

$T_{useful} := T_{acc}$;
$P_{useful} := P_{acc}$;
produkuj $G_{useful} = (N_{useful}, T_{useful}, P_{useful}, S)$
end.

3.2 Eliminace ε-pravidel

Definice 3.2.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Potom *ε-pravidla* jsou taková pravidla $p : A \to \varepsilon \in P$.

Z teoretického pohledu je používání ε-pravidel jistým zjednodušením. Ovšem pro aplikace gramatik v praxi platí pravý opak. Odstranění ε-pravidel je velmi důležité např. pro precedenční syntaktickou analýzu, která je nesmí obsahovat [5].

Definice 3.2.2. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika a $A \in N$. A je *ε-neterminál*, pokud $A \Rightarrow^* \varepsilon$ v G.

Pokud tedy máme neterminál, z nějž se buď přímo nebo postupnými derivacemi můžeme dostat k takovému pravidlu, na jehož pravé straně bude ε, jedná se o ε-neterminál.

Algoritmus pro nalezení množiny všech ε-neterminálů:

Algoritmus 3.2.3.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Množina N_ε, která obsahuje všechny ε-neterminály z G, $N_\varepsilon = \{A \mid A \in N$, a $A \Rightarrow^* \varepsilon$ v $G\}$.
Metoda:

begin
$N_\varepsilon := \{A\colon A \to \varepsilon \in P\}$;
repeat
$W := N_\varepsilon$;
for všechny $A \to x \in P$, kde $A \in N - W$ **do**
if $W \neq \emptyset$ a $x \in W^*$
then $N_\varepsilon := N_\varepsilon \cup \{A\}$
until $W = N_\varepsilon$
end.

Tento algoritmus opět slouží jako mezistupeň převodu. Jeho výstup v podobě množiny všech ε-neterminálů bude dále využit.

Definice 3.2.4. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. G je *bezkontextová gramatika bez ε-pravidel*, pokud pro každé $p : A \to x \in P$ platí, že $x \neq \varepsilon$.

Převod bezkontextové gramatiky na ekvivalentní gramatiku bez ε-pravidel:

Algoritmus 3.2.5.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, kde $(N \cup T)$ obsahuje pouze použitelné symboly.
Výstup: Bezkontextová gramatika bez ε-pravidel, $G_{\varepsilon\text{-}free} = (N_{\varepsilon\text{-}free}, T, P_{\varepsilon\text{-}free}, S)$, mající tyto vlastnosti:

- $L(G_{\varepsilon\text{-}free}) = L(G) - \{\varepsilon\}$

- $(N_{\varepsilon\text{-}free} \cup T)$ obsahuje pouze použitelné symboly.

Metoda:

begin
použij algoritmus 3.2.3, tím získej množinu N_ε, která obsahuje všechny ε-neterminály v G;
$P' := P - \{p : p$ je ε-pravidlo v $P\}$;
for každé $p : B \to y \in P$ takové, že $y = x_0A_1x_1 \dots A_nx_n$ a $x_0x_1 \dots x_n \neq \varepsilon$,
kde $A_i \in N_\varepsilon$ a $x_i \in (N \cup T)^*$, pro každé $i = 1, \dots, n$, a kde $n \in \{1, \dots, |y| - 1\}$ **do**
$P' := P' \cup \{B \to x_0x_1x_2 \dots x_n\}$;
použij algoritmus 3.1.4 pro převod $G' = (N, T, P', S)$ na bezkontextovou gramatiku $G_{term} = (N_{term}, T, P_{term}, S)$, kde $L(G) = L(G')$ a $N_{term} \cup T$ obsahuje pouze ukončující symboly;
$N_{\varepsilon\text{-}free} := N_{term}$;
$P_{\varepsilon\text{-}free} := P_{term}$;
produkuj $G_{\varepsilon\text{-}free} = (N_{\varepsilon\text{-}free}, T, P_{\varepsilon\text{-}free}, S)$
end.

Takto je možné odstranit ε-pravidla. Algoritmus předpokládal vstupní gramatiku již bez nepoužitelných symbolů. Spoustu algoritmů na sebe navazuje, proto můžeme v popisu používat již upravené gramatiky.

3.3 Eliminace jednotkových pravidel

Definice 3.3.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pravidlo $p \in P$ ve tvaru $A \to B$ je *jednotkovým pravidlem*, pokud $B \in N$.

Jednotková pravidla opět gramatiky zbytečně komplikují. Přepsáním jednoho neterminálu na jiný se gramatika zdržuje. Nicméně i pro jednotková pravidla existuje algoritmus, který je odstraní.

Definice 3.3.2. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pokud $A \Rightarrow^* B$ v G, kde $A, B \in N$, pak $A \Rightarrow^* B$ reprezentuje *jednotkovou derivaci*. *Cyklus* je jednotková derivace ve tvaru $A \Rightarrow^+ A$ v G, kde $A \in N$.

Analogicky z předchozích definic – pokud se můžeme z jednoho neterminálu dostat přímo či postupnou derivací k použití pravidla, které má na pravé straně jiný neterminál, pak se jedná o jednotkovou derivaci. A pokud je na pravé straně tohoto pravidla stejný neterminál jako ten, z nějž jsme vycházeli, jde o cyklus.

Následuje algoritmus pro nalezení množiny Unit(A), jenž obsahuje neterminály, ke kterým se můžeme z neterminálu A dostat pomocí jednotkové derivace:

Algoritmus 3.3.3.
Vstup: Bezkontextová gramatika bez ε-pravidel a nepoužitelných symbolů $G = (N, T, P, S)$ a neterminál $A \in N$.
Výstup: Množina Unit(A), definovaná jako Unit$(A) = \{B \mid B \in N$ a $A \Rightarrow^* B$ v $G\}$.
Metoda:

begin
Stare$(A) := \emptyset$;

Unit(A) := $\{A\}$;
repeat
Nove(A) := Unit(A) – Stare(A);
Stare(A) := Unit(A);
for každé $B \in$ Nove(A) **do**
if existuje $p : B \to C \in P$, kde $C \in N$
then Unit(A) := Unit(A) $\cup \{C\}$
until Unit(A) = Stare(A)
end.

Definice 3.3.4. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. G je *bezkontextová gramatika bez jednotkových pravidel*, pokud pro všechna $p : A \to x \in P$ platí, že $x \notin N$.

Pokud všechna pravidla gramatiky neobsahují na pravé straně samotný neterminál, jedná se o gramatiku bez jednotkových pravidel.

Převod bezkontextové gramatiky bez ε-pravidel na ekvivalentní bezkontextovou gramatiku bez ε-pravidel a bez jednotkových pravidel:

Algoritmus 3.3.5.
Vstup: Bezkontextová gramatika bez ε-pravidel $G = (N, T, P, S)$ taková, že $N_{\varepsilon\text{-}free} \cup T$ obsahuje pouze použitelné symboly.
Výstup: Bezkontextová gramatika bez ε-pravidel a bez jednotkových pravidel $G_{unit\text{-}free} = (N_{unit\text{-}free}, T, P_{unit\text{-}free}, S)$ splňující tyto podmínky:

- $L(G_{unit\text{-}free}) = L(G)$
- $N_{unit\text{-}free} \cup T$ obsahuje pouze použitelné symboly.

Metoda:

begin
$P_{unit\text{-}free} := \emptyset$;
for každé $A \in N$ **do**
použij algoritmus 3.3.3 se vstupy G a A;
for každé $A \in N$ a každé $p : B \to x, p \in P$, kde $B \in$ Unit(A) a $x \notin N$ **do**
$P_{unit\text{-}free} := P_{unit\text{-}free} \cup \{A \to x\}$;
$N_{unit\text{-}free} := \{D : D \to x \in P_{unit\text{-}free}\}$;
produkuj $G_{unit\text{-}free} = (N_{unit\text{-}free}, T, P_{unit\text{-}free}, S)$
end.

3.4 Vlastní bezkontextová gramatika

Definice 3.4.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika splňující následující podmínky:

- $N \cup T$ obsahuje pouze použitelné symboly;
- G je bez ε-pravidel;
- G je bez jednotkových pravidel.

Pak je G *vlastní bezkontextová gramatika.*

Vlastní bezkontextovou gramatiku získáme použitím všech doposud uvedených algoritmů a tedy odstraněním nežádoucích symbolů a pravidel:

Algoritmus 3.4.2.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Vlastní bezkontextová gramatika $G_{proper} = (N_{proper}, T_{proper}, P_{proper}, S)$.
Metoda:

begin
použij algoritmus 3.1.11 se vstupem G k odstranění nepoužitelných symbolů;
použij algoritmus 3.2.5 se vstupem G_{useful} k odstranění ε-pravidel;
použij algoritmus 3.3.5 se vstupem $G_{\varepsilon\text{-}free}$ k odstranění jednotkových pravidel;
$N_{proper} := G_{unit\text{-}free}$;
$T_{proper} := T$;
$P_{proper} := P_{unit\text{-}free}$;
produkuj $G_{proper} = (N_{proper}, T_{proper}, P_{proper}, S)$;
end.

3.5 Chomského normální forma

Definice 3.5.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. G je v *Chomského normální formě*, pokud jsou všechna její pravidla $p \in P$ ve tvaru $A \to x$, kde $A \in N$ a $x \in (N^2 \cup T)$.

Chomského normální forma obsahuje pouze pravidla, na jejichž pravých stranách je buď jeden samotný terminál nebo dva neterminály. Využití této normální formy je například u algoritmu pro syntaktickou analýzu bezkontextové gramatiky, CYK (Cocke-Younger-Kasami) [1], který pracuje pouze s gramatikami, jenž jsou v Chomského normální formě.

Algoritmus 3.5.2.
Vstup: Vlastní bezkontextová gramatika $G = (N, T, P, S)$.
Výstup: Bezkontextová gramatika $G_{CNF} = (N_{CNF}, T, P_{CNF}, S)$ splňující tyto dvě vlastnosti:

- $L(G_{CNF}) = L(G)$
- G_{CNF} je v Chomského normální formě.

Metoda:

begin
$P_{CNF} := \{p : A \to x \in P, \text{ kde } x \in (T \cup N^2)\}$;
$P' := \{p : A \to x \in P, \text{ kde } |x| \leq 2 \text{ a } p \notin P_{CNF}\}$;
$N_{CNF} := N$;
for každé $p : B \to X_1X_2...X_n \in P$, $X_i \in (N \cup T), i = 1, \ldots, n$, pro nějaké $n \geq 3$ **do**
begin

$$P' := P' \cup \{B \to X_1\langle X_2 \ldots X_n\rangle,$$
$$\langle X_2 \ldots X_n\rangle \to X_2\langle X_3 \ldots X_n\rangle,$$
$$\vdots$$
$$\langle X_{n-2} \ldots X_n\rangle \to X_{n-2}\langle X_{n-1} \ldots X_n\rangle,$$

$\langle X_{n-1}X_n\rangle \to X_{n-1}X_n\}$;
$N_{CNF} = N_{CNF} \cup \{\langle X_i \dots X_n\rangle : i = 2, \dots, n-1\}$
// každé $\langle X_i \dots X_n\rangle$ je nový neterminál

end;

for každé $p \in P'$, kde alph(rhs(p)) $\cap\, T \neq \emptyset$ **do**
begin
nahraď každý terminál $a \in T$ novým neterminálem a' v rhs(p);
$N_{CNF} := N_{CNF} \cup \{a'\}$;
$P_{CNF} := P_{CNF} \cup \{a' \to a\}$
end;
$P_{CNF} := P_{CNF} \cup \{p : A \to x, p \in P' \text{ a } x \in (T \cup N_{CNF}^2)\}$;
produkuj $G_{CNF} = (N_{CNF}, T, P_{CNF}, S)$
end.

3.6 Eliminace přímo levě-rekurzivních neterminálů

Definice 3.6.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Pravidlo $p : A \to x \in P$ reprezentuje *levě-rekurzivní pravidlo*, pokud $x \in \{A\}(N \cup T)^*$. Neterminál $B \in N$ je *přímo levě-rekurzivním neterminálem* v G, pokud existuje levě-rekurzivní pravidlo $p : B \to y \in P$.

Pokud se v pravidle gramatiky na levé straně vyskytne neterminál, který je v témže pravidle jako první symbol na pravé straně, pak se jedná o levě-rekurzivní pravidlo. Neterminál, který se v levě-rekurzivním pravidle objeví na levé straně, je přímo levě-rekurzivním neterminálem.

Následující algoritmus pro zadaný přímo levě-rekurzivní neterminál přetvoří vstupní gramatiku tak, že bude obsahovat méně přímo levě-rekurzivních neterminálů, než obsahovala a zadaný přímo levě-rekurzivní neterminál jím již nebude:

Algoritmus 3.6.2.
Vstup: Vlastní bezkontextová gramatika $G = (N, T, P, S)$ a neterminál $A \in N$, kde G obsahuje r přímo levě-rekurzivních neterminálů pro nějaké $r \geq 1$ a A je jeden z nich.
Výstup: Vlastní bezkontextová gramatika $G' = (N', T, P', S)$, mající tyto vlastnosti:

- G' obsahuje méně než r přímo levě-rekurzivních neterminálů;
- A není levě-rekurzivním neterminálem v G';
- $L(G) = L(G')$.

Metoda:

begin
$P'' := \{p : B \to x \in P, \text{ kde } B \notin A\}$;
$N'' := N \cup \{A'\}$; // A' je nový neterminál
for každé $p : C \to y \in P$ takové, že $C = A$ a $y \notin \{A\}(N \cup T)^+$ **do**
$P'' := P'' \cup \{A \to y, A \to yA'\}$;
for každé $p : A \to Ax \in P$ s $x \in (N \cup T)^+$ **do**
$P'' := P'' \cup \{A' \to x, A' \to xA'\}$;
použij algoritmus 3.3.5 se vstupem $G'' = (N'', T, P'', S)$ a produkuj výslednou gramatiku $G' = (N', T, P', S)$
end.

3.7 Eliminace levé rekurze

Definice 3.7.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika, $p : u \to v \in P, x \in T^*$ a $y \in (N \cup T)^*$. Pak *xuy přímo levě derivuje xvy* pomocí p v G, zapsáno $xuy \Rightarrow_{lm} xvy\ [p]$, zkráceně $xuy \Rightarrow_{lm} xvy$.

Přímá levá derivace znamená, že se v řetězci přepíše nejlevější neterminál.

V následující definici se vyskytuje značení $\Rightarrow^+_{lm}$, jenž má obdobný význam jako v definici 2.2.7.

Definice 3.7.2. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika. Neterminál $A \in N$ je *levě-rekurzivním neterminálem* v G pokud $A \Rightarrow^+_{lm} Ay$ v G, pro nějaké $y \in (N \cup T)^*$. Pokud existuje levě-rekurzivní neterminál v G, pak G je *levě-rekurzivní bezkontextová gramatika*, jinak je G *bezkontextová gramatika bez levé rekurze.*

Pokud se z určitého neterminálu dostaneme pomocí alespoň jedné levé derivace k použití pravidla, na jehož pravé straně je na prvním místě stejný neterminál, jedná se o levě-rekurzivní bezkontextovou gramatiku.

Algoritmus 3.7.3.
Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, neterminál $B \in N$ a pravidlo $p : A \to xBy \in P$, kde $A \neq B$ a $x, y \in (N \cup T)^*$.
Výstup: Bezkontextová gramatika $G' = (N, T, P', S)$ splňující tyto podmínky:

- $p \notin P'$
- $\{A \to xwy : B \to w \in P\} \subseteq P'$
- $L(G) = L(G')$

Metoda:

begin
$P' := \{A \to xwy : B \to w \in P\}$;
$P' := P' \cup (P - \{p\})$;
produkuj $G' = (N, T, P', S)$
end.

Algoritmus pro odstranění levé rekurze ze zadané bezkontextové gramatiky v Chomského normální formě:

Algoritmus 3.7.4.
Vstup: Levě-rekurzivní bezkontextová gramatika $G = (N, T, P, A_s)$ v Chomského normální formě taková, že $N = \{A_1, \ldots, A_n\}$ pro nějaké $n \geq 1$ a A_s je počáteční symbol, kde $s \in \{1, \ldots, n\}$.
Výstup: Vlastní bezkontextová gramatika bez levé rekurze, $G_{NLR} = (N_{NLR}, T, P_{NLR}, A_s)$, a platí $L(G) = L(G')$.
Metoda:

begin
for $i := 1$ **to** n **do**
begin
for $j := 1$ **to** $i - 1$ **do**

begin
for každé $p : A_i \to A_j y \in P$ **do**
begin
použij algoritmus 3.7.3 se vstupy $G = (N, T, P, A_s)$, A_j a p;
$P := P'$;
$N := N'$;
end;
end;
if A_i je přímo levě-rekurzivní neterminál
then
begin
použij algoritmus 3.6.2 se vstupy G a A_i;
$P := P'$;
$N := N'$;
end;
end;
$P_{NLR} := P$;
$N_{NLR} := N$;
produkuj $G_{NLR} = (N_{NLR}, T, P_{NLR}, A_s)$
end.

Odstranění levé rekurze je důležité pro další normální formu, kterou zmíníme.

3.8 Greibachové normální forma

Definice 3.8.1. Nechť $G = (N, T, P, S)$ je bezkontextová gramatika bez levé rekurze. G je v *Greibachové normální formě*, pokud jsou všechna její pravidla $p \in P$ ve tvaru $A \to x$, kde $x \in TN^*$.

Všechna pravidla gramatiky, která je v Greibachové normální formě, musí mít takový tvar, kde na pravé straně pravidla je právě jeden terminál, následovaný 0 až n neterminály.

Převod do Greibachové normální formy:

Algoritmus 3.8.2.
Vstup: Vlastní bezkontextová gramatika bez levé rekurze $G = (N, T, P, S)$ splňující tyto podmínky:

- $N = N_A \cup N_B$ a $N_A \cap N_B = \emptyset, N_A = \{A_1, \ldots, A_n\}$ pro nějaké $n \geq 1$ a $N_B = \{B_1, \ldots, B_m\}$ pro nějaké $m \geq 0$ ($m = 0$ implikuje $N_B = \emptyset$)
- P obsahuje tyto tři druhy pravidel:
  1. $A_i \to A_j x$, kde $i = 1, \ldots, n$, $j \in \{i + 1, ..., n\}$ a $x \in N^*$
  2. $A_i \to ax$, kde $i = 1, \ldots, n$, $a \in T$ a $x \in N^*$
  3. $B_i \to y$, kde $i = 1, \ldots, n$, $y \in (\{A_1, ..., A_n\} \cup T)N^*$
- $S = A_s$, pro nějaké $s \in \{1, \ldots, n\}$.

Výstup: Bezkontextová gramatika $G_{GNF} = (N_{GNF}, T, P_{GNF}, S)$, mající tyto vlastnosti:

- $L(G_{GNF}) = L(G)$

- G_{GNF} je v Greibachové normální formě.

Metoda:

begin
  for $i := n - 1$ **downto** 1 **do**
    for každé $p : A_i \to A_j x \in P$, kde $A_j \in \{A_{i+1}, \ldots, A_n\}$,
    $x \in (\{A_1, \ldots, A_n\} \cup T)^*$ **do**
    begin
      použij algoritmus 3.7.3 se vstupy $G = (N, T, P, A_s)$, A_j a p;
      $P := P'$;
      $N := N'$;
    end;
  for $i := n - 1$ **downto** 1 **do**
    for každé $p : A_i \to A_j y \in P$, kde $A_j \in \{A_{i+1}, \ldots, A_n\}$,
    $x \in (\{A_1, \ldots, A_n\} \cup T)^*$ **do**
    begin
      použij algoritmus 3.7.3 se vstupy $G = (N, T, P, A_s)$, A_j a p;
      $P := P'$;
      $N := N'$;
    end;
  for $i := 1$ **to** m **do**
    for každé $p : B_i \to A_j z \in P$, kde $A_j \in \{A_1, \ldots, A_n\}$ a $z \in N^*$ **do**
    begin
      použij algoritmus 3.7.3 se vstupy $G = (N, T, P, A_s)$, A_j a p;
      $P := P'$;
      $N := N'$;
    end;
  použij algoritmus 3.1.8 se vstupem G;
  $N_{GNF} := N_{acc}$;
  $P_{GNF} := P_{acc}$;
  produkuj $G_{GNF} = (N_{GNF}, T, P_{GNF}, S)$
end.

Nyní přejdeme k normálním formám pro neomezené gramatiky.

3.9 Kurodova normální forma

Definice 3.9.1. Nechť $G = (N, T, P, S)$ je neomezená gramatika. G je v *Kurodově normální formě* [8], pokud každé pravidlo $p \in P$ má jeden z těchto čtyř tvarů:

- $AB \to DC$, kde $A, B, C, D \in N$
- $A \to BC$, kde $A, B, C \in N$
- $A \to a$, kde $A \in N$ a $a \in T$
- $A \to \varepsilon$, kde $A \in N$.

Definice 3.9.2. Nechť $G = (N, T, P, S)$ je kontextově závislá gramatika. G je v *Kurodově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto tří tvarů:

- $AB \to DC$, kde $A, B, C, D \in N$
- $A \to BC$, kde $A, B, C \in N$
- $A \to a$, kde $A \in N$ a $a \in T$.

Rozdíl mezi těmito dvěma formami spočívá pouze v přípustnosti čtvrtého tvaru pravidla, které obsahuje na pravé straně ε.

3.10 Pentonnenova normální forma

Silnější formou Kurodovy normální formy je Pentonnenova normální forma [2], definována níže.

Definice 3.10.1. Nechť $G = (N, T, P, S)$ je neomezená gramatika. G je v *Pentonnenově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto čtyř tvarů:

- $AB \to AC$, kde $A, B, C \in N$
- $A \to BC$, kde $A, B, C \in N$
- $A \to a$, kde $A \in N$ a $a \in T$
- $A \to \varepsilon$, kde $A \in N$.

Definice 3.10.2. Nechť $G = (N, T, P, S)$ je kontextově závislá gramatika. G je v *Pentonnenově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto tří tvarů:

- $AB \to AC$, kde $A, B, C \in N$
- $A \to BC$, kde $A, B, C \in N$
- $A \to a$, kde $A \in N$ a $a \in T$.

Pentonnenova normální forma se od Kurodovy normální formy tolik neliší. Změnil se pouze první z množných tvarů pravidel a to tak, že první neterminál na levé straně pravidla je stejný jako první neterminál na pravé straně.

3.11 Geffertova normální forma

Existují i další, tzv. Geffertovy normální formy [11] pro neomezené gramatiky.

3.11.1 První Geffertova normální forma

Definice 3.11.2. Nechť $G = (\{S, A, B, C\}, T, P \cup \{ABC \to \varepsilon\}, S)$ je neomezená gramatika. G je v *první Geffertově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto tvarů:

- $S \to uSa$,
- $S \to uSv$,
- $S \to uv$,

kde $u \in \{A, AB\}^*$, $a \in T$, a $v \in \{BC, C\}^*$.

3.11.3 Druhá Geffertova normální forma

Definice 3.11.4. Nechť $G = (\{S, A, B, C, D\}, T, P \cup \{AB \to \varepsilon, CD \to \varepsilon\}, S)$ je neomezená gramatika. G je ve *druhé Geffertově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto tvarů:

- $S \to uSa$,
- $S \to uSv$,
- $S \to uv$,

kde $u \in \{A, C\}^*$, $a \in T$, a $v \in \{B, D\}^*$.

3.11.5 Třetí Geffertova normální forma

Definice 3.11.6. Nechť $G = (\{S, A, B\}, T, P \cup \{ABBBA \to \varepsilon\}, S)$ je neomezená gramatika. G je ve *třetí Geffertově normální formě*, pokud každé pravidlo $p \in P$ má jeden z těchto tvarů:

- $S \to uSa$,
- $S \to uSv$,
- $S \to uv$,

kde $u \in \{AB, ABB\}^*$, $a \in T$, a $v \in \{BBA, BA\}^*$.

Kapitola 4

Návrh a implementace programu

Součástí této práce je návrh a implementace programu, který převádí vstupní gramatiku do zadané normální formy. Tato kapitola tedy popisuje postup, kterým byl program vytvářen, jaké problémy při této tvorbě vznikly a byly řešeny, a proč bylo zvoleno příslušné řešení. Současně také detailněji rozvádí jednotlivé myšlenky návrhu, jsou zde uvedeny použité funkce a moduly.

4.1 Návrh programu

Nejdříve se zaměříme na samotný návrh programu. Na tuto část spadá veškerá zodpovědnost za následný správný chod programu – pokud je návrh špatný, tak i celá implementace je pak těžkopádná a celý program jde ztuha sestavit, nemluvě o následném intenzivním ladění a vzniku nových chyb. Naopak, pokud je návrh dobrý a vystihuje celou podstatu problému jednoduchou a elegantní cestou, implementace bývá rychlá, snadná, program jako celek pracuje tak, jak má a není větší problém s jeho úpravou. Je tedy velice vhodné věnovat návrhu dostatečnou dobu a opravdu se nad daným problémem zamyslet.

4.1.1 Co bylo třeba řešit

Ještě před samotným návrhem programu bylo třeba si uvědomit, s čím bude pracovat a jaké jsou na něj kladeny nároky. Jedním z požadavků je snadná rozšiřitelnost o další normální formy. Návrh tedy musí být přehledný a jednoduchý, a musí umožňovat snadné přidávání dalších algoritmů, parametrů a jejich zpracování. Program má být konzolová aplikace, která přijímá vstupní gramatiku a parametr, podle nějž se pozná, do které normální formy chce uživatel gramatiku převést.

Celý program pracuje s hlavním a stěžejním prvkem celé této práce, a tím je gramatika. Je nutné zvolit takovou reprezentaci gramatiky, která bude snadno použitelná, vzhledem k častým průchodům jejích jednotlivých prvků.

Dále musíme specifikovat vhodný formát, jímž bude gramatika programu dodávána. Tyto gramatiky si vytváří přímo sám uživatel, tedy jej nesmíme příliš omezovat a je vhodné nabídnout mu takový formát, který pro něj bude snadný na zapamatování a zápis jednotlivých komponent nebude zdlouhavý.

4.1.2 Formát vstupní gramatiky

Při pohledu na různé zápisy gramatik se drtivá většina shoduje s pořadím zápisu jednotlivých komponent tak, jak je uveden v definicích gramatik v kapitole o základních pojmech. Tedy nejprve abeceda neterminálů, poté abeceda terminálů, přepisovací pravidla a nakonec počáteční symbol. Je tedy více než vhodné toto pořadí zanechat takto. Celý zápis gramatiky by měl být nějak ohraničen, aby bylo poznat, kde začíná a kde končí. Stejně tak je dobré označit komponenty a tím je od sebe oddělit. Pro ohraničení celé gramatiky byly zvoleny kulaté závorky, pro komponenty závorky složené. Jednotlivé komponenty jsou od sebe odděleny čárkou.

Nyní k samotným komponentám. Abecedu neterminálů tvoří jednotlivé symboly, které se mohou skládat z různých znaků. Samotný symbol pak může být tvořen více než jedním znakem – toto je ovšem první problém, který je pak třeba řešit u komponenty pravidel (bude zmíněno níže). Jednotlivé neterminály jsou od sebe odděleny čárkami. Mezery okolo těchto oddělovačů mohou být jakkoliv široké nebo nemusí být zapsány vůbec, to si potom program upraví sám pro svou potřebu.

Zápis abecedy terminálů se od abecedy neterminálů téměř neliší, v podstatě pouze svým obsahem. I zde mohou být terminály zapsány jako víceznakové řetězce. Je to proto, aby bylo možné zapsat i složitější znaky, které se přímo nevyskytují na klávesnici, a použití jednoho znaku by mohlo být matoucí, nehledě na to, že by se u rozsáhlých gramatik časem mohl vyčerpat počet takových znaků. Terminály jsou opět rozděleny pomocí čárky.

Pravidla se skládají z výše uvedených symbolů – neterminálů a terminálů. Obvyklým rozdělením levé a pravé strany pravidel bývá šipka složená ze dvou znaků (->), takže to tak bude i v zápisu této komponenty. Na levé straně se mohou vyskytovat buď jeden neterminál (u bezkontextových gramatik), či kombinace terminálů a neterminálů (neomezené gramatiky). U zápisu levé a pravé strany pravidla je ovšem nutné nějakým způsobem oddělit jednotlivé terminály a neterminály. To je dáno právě kvůli možnosti zápisu víceznakových symbolů. Je možné zvolit nespočet oddělovačů, ale hrozí zde, že se takový oddělovač bude vyskytovat jako abeceda ať už terminálů či neterminálů. Proto bylo nakonec rozhodnuto, že to bude mezera. Pokud by tedy chtěl uživatel vyjádřit mezeru jako symbol některé abecedy, má možnost zapsat ji například řetězcem *mezera* a ve výsledné gramatice si za ni dosadit, co potřebuje. Navíc se díky prázdnému místu mezi jednotlivými symboly stává zápis pravidel přehlednější. Jedinou nevýhodou je opravdu nutnost uvedení této mezery, což možná může trochu zdržovat. Jednotlivá pravidla se pak mezi sebou oddělují pomocí čárky.

Samotný počáteční symbol je pak zapsán bez složených závorek.

Zadávaní gramatiky v popsaném formátu je poněkud volnější, není nutné, aby byla přesně odřádkovaná a jak již bylo zmíněno, tak ani počet mezer a bílých znaků není omezen. Nicméně výstupní převedená gramatika by měla být řádně upravená a přehledná. Jak by vypadala takto formátovaná gramatika z příkladu 2.2.5 je vidět zde:

```
({B}, {(, )}, {B -> ( B ), B -> eps}, S)
```

Na uvedeném příkladu formátované gramatiky je vidět, že symbol ε je zapsán jako řetězec *eps*. Tento zápis symbolu je závazný, není možné zapsat jej jinak. Uživatelé mohou gramatiku zapsat klidně pouze na jeden řádek, výstupní formátování bude pak s příslušným odřádkováním. To je přehledné obzvláště u rozsáhlejších gramatik. Neterminály, terminály a počáteční symbol budou zapsány na jeden řádek, na rozdíl od pravidel, jenž budou každé na zvláštním řádku.

4.1.3 Návrh reprezentace gramatiky

Dalším důležitým bodem je samotná reprezentace gramatiky uvnitř programu. Pracovat s textovým souborem je nepohodlné a velice omezující. Nám bude stačit vysbírat jednotlivé komponenty ze vstupního souboru a vhodně je uložit. Zde se objevuje další otázka k řešení, a to, jakou zvolit zmíněnou reprezentaci gramatiky. Zde již návrh zabíhá k samotné implementaci programu, ale bez vzájemné korespondence jej snad ani řešit nelze.

Základem je uvědomění si, co se s danou gramatikou bude provádět. Při pohledu a detailním prostudováním jednotlivých algoritmů se dobereme k závěru, že musíme umět procházet jednotlivá pravidla, v rámci nich pak levou a pravou stranu a symboly, které je tvoří. Samotná oddělovací šipka levé a pravé strany pravidel zde ztrácí význam, ukládat ji není třeba. Je tedy dobré zvolit takovou reprezentaci, kde budeme mít komponentu *pravidla*, složenou ze samotných pravidel, z nichž každé si s sebou ponese uloženou zvlášť levou a zvlášť pravou stranu, a každá z těchto stran bude obsahovat svoje symboly. Takto to vypadá poměrně složitě, ale není. Více o této reprezentaci si ponecháme na popis implementace.

4.1.4 Program

Na závěr sekce o návrhu bude popsán návrh programu jako celku, jednotlivé rozdělení do souborů a co by měly tyto soubory obsahovat. Je nutné, aby se algoritmy pro převod do normálních forem jednoduše přidávaly. V jednom souboru tedy budou tyto algoritmy jako jednotlivé funkce, které se budou volat v hlavním souboru programu, kde již bude probíhat samotný převod. Soubory podle obsahu a funkce pojmenujeme jako *algoritmy* a *prevod*. Pokud se tedy bude přidávat další normální forma, přidá se algoritmus pro převod do této normální formy do souboru s algoritmy. V hlavním souboru programu se pak musí přidat nový, ještě nepoužitý parametr, který bude signalizovat, že si uživatel přeje převést gramatiku do přidané normální formy. V místě, kde se rozlišují zadané parametry, je nutné tento parametr přidat spolu s voláním příslušného algoritmu pro převod.

Hlavní soubor programu, *prevod*, pak musí ze zadané gramatiky vyjmout její jednotlivé komponenty. Při tomto postupném procházení souboru a vyjímáním komponent, jsou tyto zároveň testovány na správný zápis podle zvoleného formátu gramatiky. Pokud je některá z komponent zapsána nekorektně, mělo by dojít k upozornění uživatele, že zřejmě někde udělal chybu. To je důležité, neboť překlepy jsou docela častou záležitostí a výsledná gramatika by při výskytu takových překlepů mohla být nepředvídatelná. Uživatel by pak nevěděl, co se stalo, nebo by nabyl dojmu, že program nepracuje správně. Komponenty se po kontrole uloží.

Uživatel zadá vstupní gramatiku a parametr, který udává, do jaké normální formy si ji přeje převést. Je vhodné nabídnout uživateli více způsobů zadávání gramatiky – přes soubor či na standardní vstup, stejně tak jako zápis výstupní převedené gramatiky – buď do zadaného souboru či na standardní výstup. Slušností je pak uvedení nápovědy, spustitelné při zadání příslušného parametru, která uživateli podá stručný přehled o funkci programu, obzvláště pak jednotlivé názvy parametrů a jejich význam. Podle zadaného parametru se pak rozpozná, který z algoritmů se zavolá. Po provedení převodu se gramatika zformátuje zpět do přehledné formy a zapíše se na daný výstup.

4.2 Implementace

Teď již k samotné implementaci. V této sekci budou zmíněny jednotlivé principy, které jsou použity ve vytvořeném programu. Nejprve je popsán výběr jazyka a použitých modulů. Následuje blízký pohled na reprezentaci gramatiky, proces kontroly zadané gramatiky, a detailní popis toku programu při jeho spuštění.

4.2.1 Zvolený programovací jazyk a použité moduly

Nejprve bylo třeba zvolit vhodný jazyk pro implementaci programu. Jako nejvhodnější se, vzhledem k zadání, nabízí objektově orientovaný skriptovací jazyk Python. Konkrétně byla použita verze 3.2.1 [9] a program byl řešen procedurálně. Tento jazyk poskytuje velké množství modulů, z nichž některé najdou v programu dobré uplatnění. Python nabízí k použití zajímavé struktury, jako jsou např. seznamy, které se budou velice hodit. Výhodou je také syntaxe cyklu `for`, která je jednoduchá a lehce čitelná. Tento cyklus opět najde v programu využití. Následuje stručný popis funkčnosti a využití jednotlivých importovaných modulů v programu.

Modul getopt

Tento modul slouží k rozpoznání parametrů z příkazové řádky. Konkrétně jeho funkce `getopt()` zajistí uložení zadaných parametrů do dvojic spolu s hodnotou, která u nich byla uvedena. V našem případě se hodnota uloží pouze ke dvěma parametrům, a to jsou `--vstup` a `--vystup`. Pomocí modulu `getopt` také specifikujeme krátké a dlouhé přepínače. Pro jednotlivé parametry, které značí převod do normálních forem, je k dispozici jak krátký, tak dlouhý přepínač. Krátké slouží hlavně k rychlému zápisu parametru pro uživatele, kteří si je již zapamatovali, dlouhé pak mohou být naopak využity pro větší přehlednost.

Modul sys

Modul `sys` je použit v kombinaci s modulem `getopt` a pomocí něj můžeme přistoupit k prostředí, ve kterém běží překladač Pythonu. Díky modulu `sys` se tedy dostaneme k příkazové řádce a parametrům, které pak zpracuje modul `getopt`.

Modul re

Velice užitečným a důležitým modulem je modul `re`, který implementuje regulární výrazy a různé operace, jež nad nimi lze provádět. Tohoto modulu je využito především k upravení, procházení a testování správnosti vstupní gramatiky. Využity jsou hlavně funkce `compile()` k uložení regulárního výrazu a možnosti jeho opakovaného použití, `match()`, který testuje shodu regulárního výrazu se zadaným řetězcem, `group()`, který umožní uložení příslušné shody, a `sub()` pro nahrazení všech výskytů určeným řetězcem.

Modul copy

Posledním použitým externím modulem je modul `copy`. Toho je využito k vytváření kopií objektů.

4.2.2 Použitá reprezentace gramatiky

Jak již bylo zmíněno v návrhu reprezentace gramatiky, je třeba nějakého vhodného způsobu uložení, které umožní snadné používání gramatiky v programu. Pro toto uložení se nám v Pythonu nabízí seznam. Je možné jednoduše procházet a měnit jeho prvky, navíc Python implementuje množství užitečných funkcí, které ulehčí práci s tímto typem. Celá komponenta pravidel tedy bude uložena v jednom seznamu se jménem `pravidla`, jehož prvky budou jednotlivá pravidla. Pravidlo pak je také implementováno jako seznam, jenž má dva prvky – opět dva seznamy – levou a pravou stranu, a tyto strany obsahují jako prvky symboly, které se v nich vyskytují. Komponenty `terminaly` a `neterminaly` pak budou dva samostatné seznamy, jejichž prvky budou příslušné symboly. Počáteční stav pak stačí reprezentovat jako obyčejný řetězec s názvem `start`.

Algoritmům pro převod gramatiky do normálních forem je pak třeba předat celou gramatiku. Všechny komponenty se tedy sjednotí do jednoho seznamu `gramatika`, který se předá danému algoritmu. V tomto seznamu jsou pak jednotlivé komponenty pro jednotnost v pořadí, v jakém byly zapsány ve vstupní gramatice, tedy `neterminaly`, `terminaly`, `pravidla` a `start`.

4.2.3 Kontrola formátu vstupní gramatiky

Gramatika, která byla zadána spolu s požadovanou normální formou, je zkopírována do řetězce. Vzhledem k tomu, že jsou povoleny různě velké mezery, musíme před testováním na správnost omezit tuto velikost. Pomocí jednoduchého regulárního výrazu, který vyhledá všechna bílá místa, a funkce `sub()` z modulu `re`, nahradíme tato místa jedinou mezerou.

Dále můžeme přejít k testování správnosti množiny neterminálů. Pro tuto komponentu je opět vytvořen regulární výraz. Pokud je nalezena shoda ve vstupní gramatice pro tento výraz, je tato shoda uložena do samostatného řetězce. Stejným způsobem, akorát s jinými regulárními výrazy, jsou do svých řetězců uloženy další komponenty. Pokud některý z výrazů nenalezne shodu, program je ukončen a je vypsáno, ve které komponentě se výraz neshodoval, aby se uživateli zúžil prostor při hledání chyby.

Protože je např. u řetězce s neterminály obsažena levá kulatá závorka a současně obě složené závorky, které ji ohraničují, tak se pomocí nahrazování a regulárních výrazů tyto vyhledají a vymažou.

Nyní je třeba rozdělit řetězce do seznamů. Toho je docíleno pomocí operace `split()`. Množina neterminálů a množina terminálů se rozdělí podle oddělovací čárky. U pravidel je to složitější. Rozdělení, kterého potřebujeme docílit, je uvedeno výše. Nejprve tedy následuje rozdělení podle oddělovací čárky. Pak již procházíme seznam pravidel cyklem `for`, a pro každé pravidlo provedeme opět rozdělení, tentokrát ale podle šipky, oddělující obě strany pravidla. Teď stačí akorát projít všechna pravidla a vždy jejich pravou a levou stranu rozdělit podle oddělovací mezery.

4.2.4 Tok programu

Jak bude vypadat tok programu při zadání vstupní gramatiky a normální formy bude popsáno zde. Nejprve si uživatel zvolí, jakým způsobem chce zadat gramatiku. Častější způsob zřejmě bude pomocí vstupního souboru a parametru `--vstup`. Bez uvedení tohoto parametru se automaticky jako vstup bere standardní vstup. Spolu s gramatikou pak zadá příslušný parametr pro převod gramatiky. Například pro odstranění ε-pravidel je to krátký přepínač `-e` nebo dlouhý přepínač `--bez-epsilon`. Program zjistí, které parametry byly

předány, otevře zadaný soubor pro čtení a uloží si jeho obsah, nebo načte text ze standardního vstupu.

Poté následuje kontrola správnosti vstupní gramatiky a uložení komponent do seznamů. Tyto postupy byly podrobněji popsány výše, není již tedy nutné zde uvádět detailnější rozpis.

Podle zadané normální formy se zavolá příslušná funkce z vytvořeného modulu `algoritmy` s parametrem `gramatika`, obsahujícím veškeré komponenty v seznamech, a provede se požadovaný převod. Ten je opět uložen do proměnné `gramatika`. Některé převody generují nové neterminály. Názvy těchto neterminálů se skládají z původního jména neterminálu a přidaného apostrofu.

V převedené gramatice seřadíme jednotlivé prvky komponent podle abecedy a zkopírujeme je ve výstupním formátu do řetězce. Pokud byl zadán parametr `--vystup`, kde byl uveden soubor, kam se má převedená gramatika uložit, se tento soubor otevře pro zápis a vložíme do něj výstupní gramatiku. V opačném případě, bez uvedení tohoto parametru, se gramatika vypíše na standardní výstup.

Kapitola 5

Testování

Vytvořený program je potřeba řádně otestovat. Při testování kontrolujeme, jestli program funguje tak, jak má. Navíc můžeme narazit na chyby, které jsme zapomněli ošetřit. Proto je testování tak důležitá a nesmírně užitečná část celé tvorby programu. Tato kapitola popisuje, jakým způsobem byl vytvořený program testován, jaké testy byly použity a co bylo pomocí nich zjištěno. Dále zde jsou uvedeny příklady těchto testů, jejich vstupy a výstupy.

5.1 Použité testy

Celou tvorbu programu provádělo postupné testování dílčích funkcí. Nejprve to byly různé testy, které kontrolovaly, zda se správně načetly parametry, zda byl otevřen soubor či zda byla správně načtena gramatika.

Velice důležité pak byly testy, zda se zadaná gramatika správně rozdělí do seznamů. Zde se objevily první chyby, které následovalo upravování příslušných regulárních výrazů do doby, než vše fungovalo tak, jak má. Zároveň s tímto bodem se testoval i výpis gramatiky ve výstupním formátu. Bylo potřeba vidět výstup testů přehledně, aby se urychlilo ladění programu.

Poté, co bylo zajištěno, že se gramatika správně uloží, následovalo testování jednotlivých, postupně vytvářených algoritmů pro převod. Toto testování trvalo nejdelší dobu. Výstupy prozradily chyby v implementaci algoritmů, ale také jen třeba nechtěné překlepy. Algoritmus pro převod byl vždy nejdříve vyladěn, a až poté se vytvářely a testovaly další. Některé převody totiž vyžadují používání dílčích funkcí, které bylo třeba otestovat, aby pak nebylo hledání chyby v celém převodu chaotické.

5.2 Příklady testů a jejich výstupy

Tato sekce uvádí některé z testů, jenž byly použity k testování programu, přičemž se jedná o testy, které kontrolují již správnost převodu do zadané normální formy. Některé testy jsou převzaté z příkladů v [4]. Tyto a jiné testy je pak možné najít v přiloženém cd v adresáři `program/testy`.

Test 1 – Eliminace nepoužitelných symbolů

Prvním testem je eliminace nepoužitelných symbolů ze zadané gramatiky. Jedná se jak o odstranění neukončujících symbolů, tak o odstranění nedostupných symbolů. Jsou tedy

testovány dva převody zároveň. Vstupní gramatika byla uložena v souboru `nepouzitelne` v místě, kde byl uložen samotný program.

```
Vstupní gramatika:
({S, A, B, C}, {+, -, a}, {S -> A + S, S -> A - S, S -> A, A -> a,
B -> B + S, B -> S, S -> C}, S)

Zadáno:
./prevod.py --vstup=nepouzitelne --bez-nepouzitelnych

Výstup:
(
{A, S},
{+, -, a},
{
A -> a ,
S -> A ,
S -> A + S ,
S -> A - S
},
S
)
```

Vstupní gramatika, která byla převáděna, obsahuje jeden neukončující symbol C, a pak zbytečné pravidlo S -> C. Dále je zde nedostupný symbol B a nedostupná pravidla s tímto symbolem na začátku.

Výsledná převedená gramatika pak tyto nepoužitelné symboly a pravidla odstranila. Převod na gramatiku bez nepoužitelných symbolů tedy proběhl úspěšně.

Test 2 – Eliminace ε-pravidel

Dalším uvedeným testem je eliminace ε-pravidel. Na vstupu se předpokládá gramatika, která již nemá nepoužitelné symboly. Pro převod na takovouto gramatiku je možné využít implementované převody. Vstupní gramatika byla uložena v souboru `epsilon`.

```
Vstupní gramatika:
({S, A, B, C}, {a, b, c, d}, {S -> A, S -> B, S -> C, A -> a A b,
B -> c B d, A -> eps, B -> eps, C -> eps}, S)

Zadáno:
./prevod.py --vstup=epsilon -e

Výstup:
(
{A, B, S},
{a, b, c, d},
{
A -> a A b ,
A -> a b ,
```

```
B -> c B d ,
B -> c d ,
S -> A ,
S -> B
},
S
)
```

Zadaná gramatika obsahovala několik ε-pravidel. Převodem se tato pravidla odstranila a modifikovala se ta stávající, která měla na pravé straně ε-neterminál. Převod tedy proběhl úspěšně.

Test 3 – Převod do Chomského normální formy

Následuje ukázka převodu gramatiky do Chomského normální formy. Gramatika je uložena v souboru `chomskeho`. Vstup předpokládá gramatiku, která je vlastní.

```
Vstupní gramatika:
({S}, {+, *, (, ), a}, {S -> S + S, S -> S * S, S -> ( S ), S -> a}, S)

Zadáno:
./prevod.py --vstup=chomskeho --chomskeho

Výstup:
(
{(', )', *', *S, +', +S, S, S)},
{(, ), *, +, a},
{
(' -> ( ,
)' -> ) ,
*' -> * ,
*S -> *' S ,
+' -> + ,
+S -> +' S ,
S -> (' S) ,
S -> S *S ,
S -> S +S ,
S -> a ,
S) -> S )'
},
S
)
```

Pravidla, která splňovala Chomského normální formu, byla ponechána. Zde se jednalo pouze o pravidlo S -> a. Ostatní pravidla byla patřičně upravena. Vznikly navíc nové neterminály. Výstupní gramatika obsahuje pravidla, která všechna splňují Chomského normální formu. Převod tedy proběhl úspěšně.

Kapitola 6

Závěr

V rámci této práce byl čtenář seznámen s některými základními pojmy z oblasti teorie formálních jazyků, především pak s normálními formami gramatik a s algoritmy pro převod gramatik do těchto normálních forem.

Byl zde demonstrován návrh a implementace programu, který tyto převody realizuje. Program byl řádně otestován a bylo prokázáno, že funguje správně. Cílem bylo vytvořit snadno rozšiřitelný program, což se povedlo. Vzhledem k tomu je tedy možné snadno přidávat další algoritmy pro převod do normálních forem.

Možný další vývoj práce by mohl spočívat v již zmíněném rozšíření o další algoritmy převodu, například do Pentonnenovy a Geffertovy normální formy. Také by se dalo převést program do objektově orientovaného návrhu. Nebo jen rozšířit tuto práci na základě textu, přidat příklady k různým převodům a normálním formám a detailněji je popsat. Pokud by totiž byla tato textová rozšíření přímo zahrnuta v práci, značně by to prodloužilo její obsah, a ten by se stal méně čitelným.

Literatura

[1] CYK algorithm [online]. `http://en.wikipedia.org/wiki/CYK_algorithm`, poslední úprava 2012-5-7. [cit. 2012-05-14].

[2] M. Penttonen: One-sided and two-sided context in formal grammars. 1974.

[3] M. Češka, T. Vojnar, A. Smrčka: Teoretická informatika (opora k předmětu TIN). `http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf`, 2011.

[4] Meduna, A.: *Automata and Languages: Theory and Applications*. Springer Verlag, 2000, ISBN 1-85233-074-0, 892 s.

[5] Meduna, A.: *Elements of Compiler Design*. Taylor & Francis Informa plc, 2008, ISBN 978-1-4200-6323-3, 304 s.

[6] Meduna, A.; Lukáš, R.: Formální jazyky a překladače (slidy k předmětu IFJ). `http://www.fit.vutbr.cz/study/courses/IFJ/public/materials/`, 2011.

[7] Rozenberg, G.; Salomaa, A.: *Handbook of Formal Languages: Vols. 1 - 3*. Springer, 2004, ISBN 3540614869, 2051 s.

[8] S.-Y. Kuroda: Classes of languages and linear-bounded automata. 1964.

[9] Summerfield, M.: *Python 3: Výukový kurz*. Computer Press, a. s., 2010, ISBN 978-80-251-2737-7, 584 s.

[10] Vaníček, J.; Papík, M.; Pergl, R.; aj.: *Teoretické základy informatiky*. Kernberg Publishing, s.r.o., 2007, ISBN 978-80-903962-4-1, 431 s.

[11] Viliam Geffert: Normal forms for phrase-structure grammars. *ITA*, 1991: s. 473–498.