VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

GRAMATICKÉ SYSTÉMY A SYNTAKTICKÁ ANALÝZA NA NICH ZALOŽENÁ

GRAMMAR SYSTEMS AND PARSING BASED ON THEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN HRSTKA**
AUTHOR

VEDOUCÍ PRÁCE **Prof. RNDr. ALEXANDER MEDUNA, CSc.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19170/2016/xhrstk02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Hrstka Jan**
Obor: Informační technologie
Téma: **Gramatické systémy a syntaktická analýza na nich založená**
Grammar Systems and Parsing Based on Them
Kategorie: Překladače

Pokyny:

1. Dle instrukcí vedoucího se seznamte s gramatickými systémy a vybranými metodami syntaktické analýzy.
2. Zaveďte nové metody syntaktické analýzy, které jsou založeny na gramatických systémech. Tyto metody budou založeny na kombinaci několika dílčích metod syntaktické analýzy.
3. Studujte vlastnosti metod z bodu 2. Porovnejte jejich sílu s klasickými metodami syntaktické analýzy.
4. Dle pokynů vedoucího uvažujte řadu syntaktických struktur, včetně struktur, které nejsou bezkontextové. Popište jejich analýzu prostřednictvím modifikovaných metod z bodu 2.
5. Aplikujte metody z bodu 2 v kompilátorech. Aplikaci zaměřte na analýzu syntaktických struktur, které nejsou bezkontextové.
6. Implementujte aplikaci navrženou v bodě 5 a testujte ji.
7. Zhodnoťte dosažené výsledky. Diskutujte další vývoj projektu.

Literatura:

- Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Volume 1-3, Springer, 1997, ISBN 3-540-60649-1
- Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition), Pearson Education, 2006, ISBN 0-321-48681-1

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Meduna Alexander, prof. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce poskytuje ucelený přehled poznatků z oblasti gramatických systémů. Zaměřuje se především na systémy orientované sekvenčně. Cílem práce je obohatit syntaktickou analýzu o přístupy na těchto systémech založené. Vychází z bezkontextových metod syntaktické analýzy, které propojuje a rozšiřuje. Poskytuje návod k zjednodušení bezkontextových gramatik dekompozicí na vzájemně komunikující komponenty. Velká pozornost je věnována zvýšení generativní kapacity LL syntaktické analýzy. V rámci práce se podařilo sestrojit bezkontextové struktury, které jsou schopny generovat kontextové jazyky. Práce zároveň poskytuje návod k jejich implementaci. Prezentuje syntaktický analyzátor založený na LL tabulce, který dokáže zpracovat deterministické kontextové jazyky. Pomocí uvedených metod je možné rozšířit řadu používaných jazyků o kontextové prvky, především o prvky popírající větu o iteraci.

Abstract

This thesis provides a summary of knowledge of grammar systems. It focuses primarily on sequentially oriented grammar systems. The aim of thesis is to introduce approaches to syntactic analysis base on grammar systems. Thesis is based on context-free methods of syntactic analysis, extending them and connecting them together. It provides recipe for simplification of big context-free grammars using decomposition to cooperating components. Great attention is dedicated to increase generative capacity of LL parsing. There were created structures whithin this thesis, which are capable to generate context-sensitive languages. This work also provides a simple recipe for implementation of these structures. We introduced LL table based parser, which is capable to parse deterministic context-sensitive languages. Using presented method is possible to extend many of often used languages with context-sensitive elements, especially elements contradicting with pumping lemma.

Klíčová slova

gramatické systémy, LL syntaktická analýza, bezkontextové metody, zpracování kontextových jazyků, transformace na LL tabulku

Keywords

grammar systems, LL parsing, context-free approach, parsing context languages, LL table transformation

Citace

HRSTKA, Jan. *Gramatické systémy a syntaktická analýza na nich založená.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Meduna Alexander.

Gramatické systémy a syntaktická analýza na nich založená

Prohlášení

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

.....
Jan Hrstka
10. května 2017

Poděkování

Chtěl bych poděkovat vedoucímu práce, panu prof. RNDr. Alexandru Medunovi CSc., za poskytnutou odbornou pomoc a panu PhDr. Miroslavu Hrstkovi Ph.D. za pravopisnou a typografickou korekci.

Obsah

Seznam tabulek

1 Úvod

V klasické teorii formálních jazyků od počátků převládaly centralizované výpočetní prostředky, gramatiky, automaty, u nichž je výpočet řízen jednou centrální autoritou. Jazyky byly generovány pouze jednou gramatikou či přijímány jedním automatem. V 80. letech 20. století se objevila myšlenka distribuovaného výpočtu a začalo se zkoumat, jak se změní vlastnosti gramatik, pokud je seskupíme do větších celků, gramatických systémů. Ukázalo se, že takové seskupení zvyšuje generativní sílu gramatik a zároveň dochází ke snížení složitosti – pomocí relativně jednoduchých gramatických systémů je možné popsat jazyky, jež nespadají do třídy bezkontextových jazyků. Dnes již distribuovaný výpočet hraje na poli teoretické informatiky majoritní roli, tato myšlenka se uchytila především v oblastech počítačových sítí či databázových systémů. K rozšíření přispěla i možnost paralelního zpracování a rozvoj víceprocesorových výpočetních strojů.

Co to tedy je gramatický systém? Jedná se o množinu společně pracujících gramatik generujících jeden jazyk. Tyto gramatiky mezi sebou komunikují pomocí *kooperačního protokolu*. Právě způsob komunikace je zcela klíčový, v některých případech je teorie gramatických systémů spíše chápána jako teorie komunikačních protokolů. Výzkum gramatických systémů se rozštěpil podle způsobu komunikace na větev sekvenční a větev paralelní[4].

Pro sekvenčně orientované gramatické systémy se vžil název *kooperačně distribuované* (CD). V takovém systému všechny gramatiky pracují na jediné větné formě. V systému je vždy právě jedna gramatika *aktivní*. Ta je vybírána na základě kooperačního protokolu. Aktivní gramatika pracuje na větné formě tak dlouho, dokud není splněna *ukončující podmínka*, rovněž stanovená kooperačním protokolem. Následně je činnost aktivní gramatiky ukončena a řízení je předáno další gramatice. Ukončujících podmínek je mnoho, my se v následujících úvahách zaměříme na pět základních: (i) gramatika provede nejvýše k kroků, (ii) právě k kroků, (iii) nejméně k kroků, (iv) nejvyšší možný počet kroků či (v) libovolný počet kroků (krokem je myšlena aplikace přepisovacího pravidla).

Druhou větev tvoří *paralelně komunikující* (PC) gramatické systémy. Každá gramatika v takovém systému má svůj počáteční neterminál a svoji vlastní větnou formu, na které pracuje. Celý systém je synchronizován podle hodin – v každém taktu provedou všechny gramatiky aplikaci některého přepisovacího pravidla. Klíčovou vlastností je komunikace pomocí *žádostí* – pokud některá gramatika potřebuje komunikovat v rámci systému, zveřejní ve své větné formě *dotazovací symbol* Q_i. Po jeho zveřejnění některou z gramatik dochází ke *komunikačnímu kroku*, ve kterém se všechny výskyty dotazovacího symbolu Q_i nahradí větnou formou i-té gramatiky. V PC gramatickém systému se nachází jedna vyvolená komponenta nazývaná *master*. Jazyk generovaný touto komponentou je rovněž jazykem generovaným celým systémem.

Výše zmíněné systémy navazují na dva populární modely řešení problémů. Prvním modelem je *model tabule*[4]. V tomto modelu je celý problém (zbývající otázky k vyřešení i současný stav) formulován na tabuli. Problém řeší několik *autorit* (agentů), které působí jako zdroje znalostí, ale pouze jedna autorita může v rámci svých kompetencí pracovat na tabuli. Po provedení svých zamýšlených kroků přenechá práci další autoritě. Domluva, která další autorita bude na tabuli pracovat, probíhá podle výše zmíněného kooperačního protokolu.

Druhým z modelů je *třídní model*[4]. V tomto modelu máme jednu centrální autoritu – vyučujícího a poté libovolný počet autorit jí podřízených – studentů. Celý problém je rovněž formulován na tabuli, ale jediný, kdo ji může editovat, je právě vyučující. Vyučující rozdělí problém na tabuli do podproblémů, které nechá řešit studenty. Každý student poté pracuje na problému ve svém sešitu. Pokud některý student potřebuje další informace, může se zeptat ostatních studentů či vyučujícího v závislosti na komunikačním protokolu – protokol určuje, zdali studenti mohou klást otázky pouze na výzvu od vyučujícího, nebo kdykoliv během řešení problému. Vyučující se rovněž může dotazovat studentů na jejich vyřešené podproblémy, nebo pouze počká, až mu studenti sami sdělí jejich výsledky.

Oba teoretické modely jsou už známy mnoho let a existují způsoby, jak je interpretovat a implementovat. Před jejich objevením se svět informatiky v této oblasti zaměřoval především na výzkum bezkontextových gramatik. Vzniklo několik prakticky zaměřených metod určených k syntaktické analýze bezkontextových jazyků – od prediktivní syntaktické analýzy s využitím LL tabulky, přes precedenční syntaktickou analýzu až k LR syntaktické analýze, která poskytuje stejnou generativní sílu, jako samotné zásobníkové automaty. S rozvojem informačních prostředků přestávala síla bezkontextových gramatik stačit. Chceme-li na počítači zpracovávat přirozený jazyk, je nutné se posunout za hranice množiny jazyků generovaných zásobníkovými automaty. Dnes již mnohé programovací jazyky v sobě zahrnují prvky, jež nejsou bezkontextové – příkladem můžeme uvést C++[6] nebo Python[7]. Z těchto důvodů se rozběhl výzkum kontextových gramatik a na ně navázaných metod syntaktické analýzy. Ty ovšem nejsou zdaleka tak jednoduché, jako metody bezkontextové. Samotná pravidla kontextových gramatik jsou špatně čitelná, a proto odhalení jazyku generovaného takovou gramatikou vyžaduje jistou míru úsilí.

V následujících stranách se pokusíme využít bezkontextové metody k analýze kontextových struktur. Několika jednoduchými úpravami docílíme toho, že bude možné analyzovat některé kontextové jazyky pomocí bezkontextových pravidel, což je významný přínos k čitelnosti gramatik a umožňuje to jednodušší zpracování těchto jazyků. Kromě teorie potřebné k pochopení problematiky, si zavedeme několik množin, pomocí kterých provedeme transformaci gramatických systému do mírně rozšířené LL tabulky. Tyto algoritmy implementujeme v jazyce Python a s jejich využitím vytvoříme syntaktické analyzátory pro některé dobře známé kontextové jazyky.

1.1 Prerekvizity z formálních jazyků

Tato práce je určena čtenářům, kteří již z jiné literatury získali základní znalosti z teorie formálních jazyků. Vhodným průvodcem základy pro čtenáře nevyhýbající se anglické literatuře může být práce Rozenberga a Salomaa [5] či Meduny [3], ovšem lze nalézt i česky psanou literaturu, například [8], skýtající ucelený přehled témat, na které v tato práce navazuje.

Pro lepší orientaci v textu si nyní zavedeme jednotné konvence označování, které jsou v celé práci dodržovány. Nechť Σ je *abeceda* (konečná množina symbolů), poté označení Σ^* reprezentuje množinu všech řetězců nad touto abecedou. Prázdný řetězec budeme značit ε a $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ je množina všech řetězců nad abecedou Σ obsahujících alespoň jeden symbol. Jazykem rozumíme libovolnou podmnožinu Σ^*.

Řetězce budeme označovat malými písmeny z konce abecedy, typicky x, y, z, v, w. Délku řetězce označíme jako $|x|$ a pro $\sigma \subseteq \Sigma$ bude označení $|x|_\sigma$ reprezentovat počet výskytů symbolů z σ v řetězci x (délka řetězce získaného z x odstraněním všech symbolů z $\Sigma - \sigma$). *Chomského gramatikou* myslíme uspořádanou n-tici $G = (N, T, P, S)$, kde N je *abeceda neterminálů*, T je *abeceda terminálů*, S je *počáteční neterminál* (též *axiom*) a P je konečná množina *přepisovacích pravidel* nad $N \cup T$ ve tvaru $u \to v$, kde $u, v \in (N \cup T)^*$, $|u|_N \geq 1$. Pravidla se stejnou levou stranou budeme pro zjednodušení značit $u \to v|w$, kde $w \in (N \cup T)^*$. Terminální symboly gramatik označíme malými písmeny ze začátku abecedy, typicky a, b, c, d. Neterminální symboly budou značeny velkými písmeny. *Derivační krok* délky 1, značeno $\Rightarrow$, je definován

$$x \Rightarrow y, \text{ pokud } x = x_1 u x_2,\ y = x_1 v x_2,\ \text{kde } x_1, x_2 \in (N \cup T)^* \\ \text{pro nějaké pravidlo } p : u \to v \in P$$

Chceme-li explicitně vyjádřit podle jakého pravidla se derivační krok provedl, píšeme $x \Rightarrow y[p]$. Se znalostí definice derivačního kroku délky 1 můžeme definovat derivační krok délky n, značeno $\Rightarrow^n$, následovně:

$$x_0 \Rightarrow^n x_n, \text{ pokud } x_0, \ldots, x_n \in (N \cup T)^*,\ n \geq 1,\ x_{i-1} \Rightarrow x_i \text{ podle } p \in P \text{ pro } \forall i : 1 \leq i \leq n.$$

Dále zavedeme derivace libovolné délky, tedy *reflexivní uzávěr* relace $\Rightarrow$, značený $\Rightarrow^+$, respektive *reflexivní a tranzitivní uzávěr* relace $\Rightarrow$, značený $\Rightarrow^*$, jsou definovány:

$$x \Rightarrow^+ y, \text{ pokud } x \Rightarrow^n y \text{ pro } n \geq 1, \\ x \Rightarrow^* y, \text{ pokud } x \Rightarrow^n y \text{ pro } n \geq 0.$$

Větnou formou rozumíme libovolný řetězec x_i nad $N \cup T$ odvoditelný z počátečního neterminálu. *Větou* označujeme větnou formu x_i, pro kterou platí $x_i \in T^*$. *Jazyk generovaný gramatikou* je množina $L(G) = \{w \in T^* | S \Rightarrow^* w\}$. Pomocí zkratek REG, LIN, CF, CS a RE si označíme rodiny *regulárních*, *lineárních*, *bezkontextových*, *kontextových* a *rekurzivně spočetných* jazyků. Označení MAT reprezentuje rodiny jazyků generovaných *maticovými gramatikami*. U těchto rodin jazyků nejsou využita ε-pravidla. Přítomnost ε-pravidel indikujeme pomocí ε psaného horním indexem (například CF^ε). Obdobně dolním indexem psané ac indikuje povolenou kontrolu výskytů. Takto dostáváme rodiny jazyků MAT_{ac}, MAT^ε, MAT^ε_{ac}. Dále označíme $ET0L$ a $EDT0L$ rodiny jazyků generovaných $ET0L$, respektive $EDT0L$ systémy.

2 Kooperačně distribuované gramatické systémy

Kooperačně distribuované (dále jen *CD*) gramatické systémy navazují na výše zmíněný model tabule. Tyto systémy pracují sekvenčně, jednotlivé gramatiky se střídají v práci na větné formě na základě kooperačního protokolu.

Definice 2.0.1. *Kooperačně distribuovaný gramatický systém*[4] je uspořádaná n-tice

$$\sigma = (N, T, S, P_1, P_2, \ldots, P_n),$$

kde N je abeceda *neterminálů*, T je abeceda *terminálů*, přičemž $N \cap T = \emptyset$. S je *počáteční neterminál* a $P_1, P_2, \ldots, P_n$ jsou konečné množiny *přepisovacích pravidel* nad abecedou $N \cup T$, rovněž nazývané jako *komponenty* systému σ.

Na jednotlivé komponenty se můžeme podívat jako na samostatné gramatiky sdílející stejnou abecedu – každá komponenta má vlastní seznam pravidel, které může aplikovat na větnou formu. Typickým rysem CD gramatického systému je aktivita právě jedné komponenty v každém okamžiku. Na základě kooperačního protokolu je vybrána některá komponenta a ta se stane *aktivní*. Aktivní komponenta pracuje na sdílené (jediné) větné formě dokud není splněna *ukončující podmínka*, poté se na základě kooperačního protokolu vybere další komponenta a ta se stane aktivní. Klíčovou roli hraje ukončující podmínka, po jejímž splnění znovu nastává výběr komponenty. V literatuře lze narazit na mnoho ukončujících podmínek, my se zaměříme na několik základních. U klasického CD gramatického systému sdílí všechny komponenty jednu jedinou ukončující podmínku – o takovém systému můžeme prohlásit, že všechny jeho komponenty pracují ve stejném módu. *Módem* budeme označovat způsob provádění derivačních kroků. Mód práce komponenty určuje, kdy může komponenta provést derivaci a jaká je délka derivace. Mód formálně definujeme jako prvek z množiny $D = \{t, *\} \cup \{=k, \geq k, \leq k \mid k \geq 1\}$. Nyní si formálně definujme derivační kroky prováděné v těchto módech – *ukončující derivaci*, derivaci dlouhou *právě* k kroků, *nejméně* k kroků a *nejvýše* k kroků.

Definice 2.0.2. *Ukončující derivace* i-té komponenty[4], značeno $\Rightarrow^{t}_{P_i}$, je definována:

$$x \Rightarrow^{t}_{P_i} y, \text{ pokud } x \Rightarrow^{*}_{P_i} y \text{ a zároveň neexistuje takové } z \in (N \cup T)^*, \text{ že } y \Rightarrow_{P_i} z.$$

Právě k kroků dlouhá derivace i-té komponenty, značeno $\Rightarrow^{=k}_{P_i}$, je definována:

$$x \Rightarrow^{=k}_{P_i} y, \text{ pokud existují taková } x_1, \ldots, x_{k+1} \in (N \cup T)^*, \text{ že platí } x = x_1, y = x_{k+1}$$
$$\text{a zároveň } x_j \Rightarrow_{P_i} x_{j+1} \text{ pro } \forall j : 1 \leq j \leq k.$$

Nejvýše k kroků dlouhá derivace i-té komponenty, značeno $\Rightarrow_{P_i}^{\leq k}$, je definována:

$$x \Rightarrow_{P_i}^{\leq k} y, \text{ pokud } x \Rightarrow_{P_i}^{=\overline{k}} y \text{ pro nějaké } \overline{k} \leq k.$$

Nejméně k kroků dlouhá derivace i-té komponenty, značeno $\Rightarrow_{P_i}^{\geq k}$, je definována:

$$x \Rightarrow_{P_i}^{\geq k} y, \text{ pokud } x \Rightarrow_{P_i}^{=\overline{k}} y \text{ pro nějaké } \overline{k} \geq k.$$

Neformálně řečeno, v módu $=k$ komponenta P_i provede na větné formě právě k derivačních kroků, kde krokem je myšlena aplikace přepisovacího pravidla. Mód $\leq k$ značí aplikaci nejvýše či právě k přepisovacích pravidel, zatímco mód $\geq k$ značí aplikaci alespoň k přepisovacích pravidel během derivace. Mód t značí užití ukončující derivace, tedy komponenta pracuje, dokud lze aplikovat nějaké přepisovací pravidlo. Mód $*$ označuje tranzitivní a reflexivní uzávěr relace $\Rightarrow$, tedy komponenta provádí derivační kroky tak dlouho, dokud se sama nerozhodne předat aktivitu další komponentě. V módu $*$ může komponenta provádět i derivace nulové délky, což lze využít především z teoretického hlediska.

Definice 2.0.3. *Jazyk generovaný CD gramatickým systémem* σ s n komponentami v módu $f \in D$ je podle [4] definován:

$$L_f(\sigma) = \left\{ w \in T^* | S \Rightarrow_{P_{i_1}}^{f} w_1 \Rightarrow_{P_{i_2}}^{f} \ldots \Rightarrow_{P_{i_m}}^{f} w_m = w,\ m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m \right\}.$$

Do jazyka generovaného gramatických systémem spadá každý řetězec odvozený z počátečního neterminálu s užitím libovolných komponent. Všechny derivace jsou při odvozování prováděny ve stejném módu, jenž je explicitně uveden a není pevnou součástí systému. Z toho plyne, že jeden systém může generovat vícero jazyků v závislosti na zvoleném derivačním módu. Pro lepší porozumění CD gramatickým systémům si jejich funkčnost ověříme na příkladu.

Příklad 2.0.4. Uvažujme jazyk $L_1 = \{a^n b^n c^n | n \geq 1\}$, o němž je dobře známo, že nespadá do třídy bezkontextových jazyků, což se velmi snadno dokáže pomocí pumping lemmatu. Pokud se pokusíme nalézt bezkontextovou gramatiku, která tento jazyk generuje, pak zajisté selžeme. Takovou bezkontextovou gramatiku nelze nalézt, na druhou stranu je možné sestrojit CD gramatický systém využívající pouze bezkontextových komponent, jež tento jazyk generuje. Zaměřme se na následující systém:

$$\begin{aligned}
\sigma_1 &= \left(\{S, A, \overline{A}, C, \overline{C}\},\ \{a, b, c\},\ AC,\ P_1,\ P_2\right) \\
P_1 &= \{1|2 : A \to a\overline{A}b \,|\, ab,\ \ 3|4 : C \to c\overline{C} \,|\, c\} \\
P_2 &= \{5|6 : \overline{A} \to aAb \,|\, ab,\ \ 7|8 : \overline{C} \to cC \,|\, c\}
\end{aligned}$$

Nechť tento CD gramatický systém pracuje v módu $=2$, tedy každá komponenta provede vždy právě dva kroky. Systém provede sekvenci derivačních kroků následovně:

$$AC \Rightarrow_{P_1}^{=2} a\overline{A}bc\overline{C} \Rightarrow_{P_2}^{=2} aaAbbccC \Rightarrow_{P_1}^{=2} aaa\overline{A}bbbccc\overline{C} \Rightarrow_{P_2}^{=2} aaaabbbbcccc.$$

V uvedeném příkladu se komponenty v práci na větné formě pravidelně střídají, což je dáno strukturou ukázkového systému, nikoliv způsobem práce CD gramatických systémů. V tomto příkladu volba komponenty odpadá, protože lze vždy využít pouze jednu. V praxi

zcela jistě může nastat situace, kdy bude možné na větnou formu aplikovat více komponent. V takovém případě určí kooperační protokol, jaká z komponent se použije. Pozorný čtenář si rovněž všimne, že první komponenta využívá pravidla pouze v kombinacích 1-3 a 2-4, ačkoliv je zde $\binom{4}{2} = 6$ možností, jak pravidla zkombinovat. Zatímco kombinace 1-3 zachová počet neterminálů ve větné formě, kombinace 2-4 odstraní všechny neterminály z větné formy, tedy získáme větu. Pokud bychom využili kombinace 1-2, 1-4, 2-3 či 3-4, pak bychom z větné formy během derivace odstranili pouze jeden neterminál. Protože všechny komponenty pracují v módu $=2$, nelze samotný neterminál odstranit – nejméně 2 neterminály jsou nutné pro aplikaci některé z komponent. Jinak řečeno, využitím ostatních kombinací pravidel dostáváme větnou formu, ze které nelze odvodit větu. Naprosto analogické to je s druhou komponentou. S Při tomuto způsobu odvozování se zachovává počet neterminálů ve větné formě a poté opravdu platí $L_{=2}(\sigma_1) = L_1$.

Uvedený příklad nás přivádí k otázce síly CD gramatických systémů. Označme si $CD_n(f)$ rodiny jazyků generované CD gramatickými systémy o nejvýše n komponentách pracující v módu $f \in D$, přičemž neuvažujeme existenci ε-pravidel. Dále označením $CD_\infty(f)$ rozumíme systém s neomezeným počtem komponent. Následující věta shrnuje výsledky prezentované v [4].

Věta 2.0.5. *Generativní kapacitu CD gramatických systémů lze popsat relacemi:*

(i) $CD_\infty(f) = CF, \forall f \in \{=1, \geq 1, *\} \cup \{\leq k | k \geq 1\},$

(ii) $CF = CD_1(f) \subset CD_2(f) \subseteq CD_r(f) \subseteq CD_\infty(f) \subseteq MAT,$
$\forall f \in \{=k, \geq k | k \geq 2\},\ r \geq 3,$

(iii) $CD_r(=k) \subseteq CD_r(=sk),\ \forall (k, r, s) \geq 1,$

(iv) $CD_r(\geq k) \subseteq CD_r(\geq k+1),\ \forall (k, r) \geq 1,$

(v) $CD_\infty(\geq k) \subseteq CD_\infty(=k),\ k \geq 1,$

(vi) $CF = CD_1(t) = CD_2(t) \subset CD_3(t) = CD_\infty(t) = ET0L.$

Při záměně inkluze $CD_\infty(f) \subseteq MAT$ *za* $CD^\varepsilon_\infty(f) \subseteq MAT^\varepsilon$*, jsou všechny uvedené relace platné i pro systémy s* ε*-pravidly.*

Z prvních dvou bodů věty 2.0.5 plyne několik významných poznatků. Systém o jedné komponentě (nezávisle na módu) je speciálním případem bezkontextové gramatiky a také poskytuje stejnou sílu. Užitím derivací v módech $=1$, ≥ 1 či $*$ generativní kapacitu rovněž nezvětšíme, nezávisle na počtu komponent – stejného chování bychom dosáhli vytvořením běžné bezkontextové gramatiky obsahující pravidla všech komponent. Hodnotné je zjištění, že ačkoliv je mód $\leq k$ zajímavý z teoretického hlediska, nevede ke zvýšení síly bezkontextových gramatik. Máme-li zadáno pouze horní omezení na délku derivace, pak můžeme vždy použít i derivaci délky 1, což degraduje chování takového systému na chování běžné bezkontextové gramatiky. Bod (ii) ukazuje, že jsme schopni se pomocí CD gramatických systémů přiblížit až k síle maticových gramatik. Podle (iii) a (iv) je možné zvýšit generativní kapacitu systému prodloužením derivace. Z bodu (v) vidíme, že jazyky generované systémy pracujícími v módu $=k$ jsou nadmnožinou jazyků generovaných v módu $\geq k$. Podle bodu (vi) lze zvýšit sílu i využitím systémů používající ukončující derivace, jež mají alespoň 3 komponenty. My se ve zbytku práce zaměříme především na derivace v módech $=k$ a t, neboť u nich lze snáze odstranit nedeterminismus a zároveň jsme jejich použitím schopni zvýšit generativní kapacitu gramatického systému.

2.1 Hybridní CD gramatické systémy

Všechny komponenty výše zmíněných systémů pracovaly ve stejném módu, tedy jednalo se homogenní CD gramatické systémy. Odstraníme-li požadavek na homogenitu, dostáváme takzvané *hybridní kooperačně distribuované gramatické systémy* (dále jen *HCD*). Dále se podíváme na to, v čem se liší chování těchto systémů a jaký to má důsledek na jejich generativní kapacitu.

Definice 2.1.1. Hybridní kooperačně distribuovaný gramatický systém[4] je uspořádaná n-tice

$$\sigma = \big(N, T, S, (P_1, f_1), \ldots, (P_n, f_n)\big),\ n \geq 1,$$

kde N je abeceda *neterminálů*, T je abeceda *terminálů*, S je *počáteční neterminál*. (P_1, f_1), $\ldots$, (P_n, f_n) jsou *komponenty* systémů, kde P_i, $1 \leq i \leq n$ jsou konečné množiny přepisovacích pravidel nad $N \cup T$ a $f_i \in D$ je mód, v němž i-tá komponenta systému pracuje.

Struktura HCD systému se od klasického CD systému liší pouze tím, že mód práce je specifikován pro každou komponentu zvlášť – každá komponenta může mít jinou ukončující podmínku. Způsob práce je obdobný jako u CD gramatického systému, ovšem generovaný jazyk nikoliv.

Definice 2.1.2. Jazyk generovaný HCD gramatickým systémem[4] s n komponentami je množina

$$L(\sigma) = \left\{ w \in T^* | S \Rightarrow_{P_{i_1}}^{f_{i_1}} w_1 \Rightarrow_{P_{i_2}}^{f_{i_2}} \ldots \Rightarrow_{P_{i_m}}^{f_{i_m}} w_m = w,\ m \geq 1, 1 \leq i_j \leq n,\ 1 \leq j \leq m \right\}.$$

Na rozdíl od klasického CD gramatického systému generuje HCD právě jeden jazyk, který je definován zvolenými módy jednotlivých komponent. Pro lepší pochopení je vhodné se rovnou podívat na ukázku práce takového systému.

Příklad 2.1.3. Uvažujme jazyk $L_{H1} = \{a^n b a^n b a^n \,|\, n \geq 0\}$, který je bezesporu kontextový. Máme zadán HCD gramatický systém, jenž tento jazyk generuje:

$$\begin{aligned}
\sigma_2 &= \big(\{S, A, \overline{A}\},\ \{a, b\},\ S,\ (P_1, =1),\ (P_2, =3),\ (P_3, =3),\ (P_4, t)\big),\\
P_1 &= \{S \to AAA\},\\
P_2 &= \{A \to a\overline{A}\},\\
P_3 &= \{\overline{A} \to aA\},\\
P_4 &= \{A \to b,\ \overline{A} \to b\}.
\end{aligned}$$

Tento systém nedeterministicky provede posloupnost derivací:

$$S \Rightarrow_{P_1}^{=1} AAA \Rightarrow_{P_2}^{=3} a\overline{A}a\overline{A}a\overline{A} \Rightarrow_{P_3}^{=3} aaAaaAaaA \Rightarrow_{P_2}^{=3} aaa\overline{A}aaa\overline{A}aaa\overline{A} \Rightarrow_{P_4}^{t} aaabaaabaaa.$$

V tomto případě jsou řetězce přijímány pouze poslední komponentou systému, která z větné formy odstraní všechny neterminály. Komponenty P_2 a P_3 se pravidelně střídají a generují nové terminály „a“ při zachování počtu neterminálů ve větné formě. První komponenta není nezbytně nutná, ale jejím vložením jsme se vyhnuli vložení takzvaného čekacího pravidla ve tvaru $A \to A$. Stejného výsledku bychom dosáhli, pokud bychom komponentu P_1 vypustili a rozšířili komponentu P_2 o pravidla $\{S \to S,\ S \to AAA\}$. Poté by bylo možné

odvodit AAA z S s využitím dvou čekacích kroků. Uvedený systému po provedení ukončující derivace již přijal řetězec a nemohl dále pracovat. To ovšem není pravidlem, v jistých případech, jak demonstruje následující příklad, lze po provedení ukončující derivace pokračovat v práci. Komponenta pracující s ukončující derivací dokonce nemusí ani přijímat řetězce, respektive být poslední v pořadí.

Příklad 2.1.4. Rozšíříme-li jazyk L_{H1} z příkladu 2.1.3, dostaneme jazyk $L_{H2} = \{a^n b^m a^{2n} b^m a^n \mid n, m \geq 0\}$, který je bezesporu kontextový. Odpovídají HCD gramatický systém je definován jako:

$$\begin{aligned}
&\sigma_3 = \big(\{S, A, \overline{A}, B, \overline{B}\},\ \{a, b\},\ S,\ (P_1, =2),\ (P_2, =2),\ (P_3, =2),\ (P_4, t)\big),\\
&P_1 = \{S \to S,\ S \to AA,\ B \to \varepsilon,\ \overline{B} \to \varepsilon\}\\
&P_2 = \{A \to a\overline{A}a,\ B \to b\overline{B}\},\\
&P_3 = \{\overline{A} \to aAa,\ \overline{B} \to bB\},\\
&P_4 = \{A \to B,\ \overline{A} \to B\}
\end{aligned}$$

Tento systém nedeterministicky provede posloupnost derivací:

$$\begin{aligned}
S &\Rightarrow^{=2}_{P_1} AA \Rightarrow^{=2}_{P_2} a\overline{A}aa\overline{A}a \Rightarrow^{=2}_{P_3} aaAaaaaAaa \Rightarrow^{=2}_{P_2} aaa\overline{A}aaaaaa\overline{A}aaa\\
&\Rightarrow^{t}_{P_4} aaaBaaaaaaBaaa \Rightarrow^{=2}_{P_2} aaab\overline{B}aaaaaab\overline{B}aaa\\
&\Rightarrow^{=2}_{P_3} aaabbBaaaaaabbBaaa \Rightarrow^{=2}_{P_1} aaabbaaaaaabbaaa.
\end{aligned}$$

Jednoduchým systémem jsme popsali kontextový a ke zpracování relativně složitý jazyk. V systému σ_3 hraje komponenta P_4 roli stavové podmínky, která se provede pouze jednou – ukončí se generování symbolů a a začnou se generovat symboly b. Jedná se o zajímavou alternativu k hlubokým zásobníkovým automatům.

Opět se dostáváme k otázce síly, kterou shrnuje následující věta z [4]:

Věta 2.1.5. *Generativní kapacitu HCD gramatických systémů lze popsat relacemi:*

(i) $CF = HCD_1 \subset HCD_2 \subseteq HCD_3 \subseteq HCD_4 = HCD_\infty \subset MAT_{ac}$,

(ii) $ET0L \subset HCD_4$,

(iii) $CD_\infty(=) \subset HCD_3$,

(iv) $CD_\infty(=) \subset HCD_\infty(fin - t) \subset (HCD_4 \cap MAT)$,

kde HCD_i *označuje rodinu jazyků generovaných HCD gramatickými systémy i komponentách, pokud není počet komponent omezen, píšeme* HCD_∞. $HCD_\infty(fin - t)$ *značí rodiny jazyků generovaných HCD systémy obsahující konečný počet komponent pracujících v módu* t.

Z (i) je na první pohled patrné, že počet komponent nevytváří nekonečnou hierarchii rodin jazyků. Uvedené relace ukazují, že HCD mají opravdu velikou generativní sílu. Podle (iii) stačí nám HCD systém o pouze třech komponentách, abychom popsali jazyky generované CD systémy o neomezeném počtu komponent a podle (iv) pouze 4 komponenty pro získání větší síly, než poskytují $ET0L$ systémy. Následující věta konstatuje sílu HCD v porovnání s CF – analogicky jakou u CD systému platí:

Věta 2.1.6. *Pokud HCD gramatický systém bez ε-pravidel σ splňuje následující vlastnosti, pak* $L(\sigma) \in CF$*:*

(i) σ *neobsahuje komponentu pracující v módu* $f \in \{=k, \geq k \mid k \geq 2\}$,

(ii) σ *obsahuje nejvíce 2 komponenty pracující v módu* t.

2.2 Systémy pracující v týmech

Doposud jsme uvažovali systémy, ve kterých pracovaly jednotlivé komponenty nezávisle na sobě. Aktivní komponenta byla vždy vybrána na základě kooperačního protokolu. Nyní komponenty sdružíme do větších celků – týmů, a podíváme se, jak se změnilo jejich chování.

Definice 2.2.1. CD gramatický systém s týmy o proměnlivé délce[4] je konstrukt

$$\sigma = (N, T, S, P_1, \ldots, P_n, R_1, \ldots, R_m),\ n, m \geq 1,$$

kde N je abeceda *neterminálů*, T je abeceda *terminálů* a S je *počáteční neterminál*. P_i, $1 \geq i \geq n$ jsou konečné množiny *přepisovacích pravidel* nad $N \cup T$, též zvané *komponenty* systému. R_j, $1 \geq j \geq n$ jsou podmnožiny $\{P_1, \ldots, P_n\}$ zvané *týmy*.

Jednoduše řečeno doplněním omezení, která určují, kdy lze použít kterou komponentu, do CD gramatického systému, získáváme systémy pracující v týmech (dále jen *PTCD*) poskytující ještě větší generativní sílu. Tyto systémy poskytují další možnosti, jak definovat vazby mezi pravidly. Derivace probíhá mírně odlišně od klasického CD gramatického systému.

Definice 2.2.2. Derivace provedená týmem $R_i = \{P_{j_1}, \ldots, P_{j_s}\}$, značeno $\Rightarrow_{R_j}$, je podle [4] definována:

$$x \Rightarrow_{R_j} y, \text{ pokud } x = x_1, A_1, x_2, A_2, \ldots, x_s, A_s, x_{s+1},\ y = x_1, y_1, x_2, y_2, \ldots, x_s, y_s, x_{s+1},$$
$$x_z, y_r \in (N \cup T)^*,\ 1 \leq z \leq s+1,\ A_r \to y_r \in P_{j_r},\ A_r \in N,\ 1 \leq r \leq s.$$

Z definice 2.2.2 je patrné, že týmová derivace o délce 1 je proveditelná, jestliže některá z komponent týmu obsahuje aplikovatelné pravidlo. Obdobným způsobem můžeme zavést derivace právě k kroků, nejméně k kroků, nejvíce k kroků či libovolný počet kroků dlouhé, značené $\Rightarrow^{=k}_{R_i}$, $\Rightarrow^{\geq k}_{R_i}$, $\Rightarrow^{\leq k}_{R_i}$ a $\Rightarrow^{*}_{R_i}$. Týmové pojetí nám umožňuje zavést několik variant ukončujících derivací.

Definice 2.2.3. Ukončující týmové derivace[4] značeny $\Rightarrow^{t_0}_{R_i}$, $\Rightarrow^{t_1}_{R_i}$ a $\Rightarrow^{t_2}_{R_i}$, jsou definovány:

$x \Rightarrow^{t_0}_{R_i} y$, pokud $x \Rightarrow^{\geq 1}_{R_i} y$ $\wedge$ neexistuje takové z, že $y \Rightarrow_{R_i} z$,
$x \Rightarrow^{t_1}_{R_i} y$, pokud $x \Rightarrow^{\geq 1}_{R_i} y$ $\wedge$ pro žádnou komponentu $P_{jr} \in R_i$ neexistuje z, že $x \Rightarrow_{P_{jr}} z$,
$x \Rightarrow^{t_2}_{R_i} y$, pokud $x \Rightarrow^{\geq 1}_{R_i} y$ $\wedge$ pro některou komponentu $P_{jr} \in R_i$
není derivace $y \Rightarrow_{P_{jr}} z$ proveditelná

Věta 2.2.3 zavádí tři varianty ukončujcích derivací, respektive 3 módy práce komponent – t_0, t_1 a t_2. V módu t_0 komponenta ukončí svou činnost, jestliže nemůže tým jako celek

provést další krok. V módu t_1 končí práce na větné formě, až nemůže žádná komponenta systému provést další krok, zatímco v módu t_2 se končí, pokud nejméně jedna komponenta nemůže přepsat žádný symbol ve větné formě.

Jaká je tedy síla PTCD gramatických systémů? V [4] byly publikovány výsledky shrnuté větou 2.2.4.

Věta 2.2.4. *Generativní kapacitu PTCD gramatických systémů lze popsat relacemi:*

(i) $PT_cCD(f) = PT_\infty CD(f) = MAT$, $\forall\big(f \in \{*\} \cup \{\leq k, =k, \geq k \mid k \geq 1\}\big)$,

(ii) $PT_cCD = PT_\infty CD(f) = MAT_{ac}$, $\forall\big(s \geq 2 \wedge f \in t_0, t_1, t_2\big)$,

kde $PT_cCD(f)$ *značí rodiny jazyků generované PTCD systémy bez* ε*-pravidel,* c *udává konstantní počet komponent a* $f \in D^{PT}$, $D^{PT} = (D \setminus \{t\}) \cup \{t_0, t_1, t_2\}$ *mód práce systému. Pokud počet komponent není omezen, píšeme* $PT_\infty CD(f)$*. Stejné relace jsou zachovány i při práci s* ε*-pravidly, pouze dochází k záměně* MAT *za* MAT^ε*, respektive* MAT_{ac} *za* MAT^ε_{ac}*.*

Můžeme tedy konstatovat, že PTCD gramatické systémy jsou velice silné – mají stejnou generativní kapacitu jako samotné maticové gramatiky.

2.3 Zvýšení síly zavedením kooperačního protokolu

Již několikrát jsme v předchozích úvahách zmiňovali kooperační protokol, jeho význam a úlohu, ale doposud jsme tento pojem ponechali na vysoké úrovni abstrakce. V této kapitole se podíváme na praktický příklad užití takového protokolu. Vhodně zvolený komunikační protokol může být účinným nástrojem vedoucím ke zvýšení síly gramatického systému.

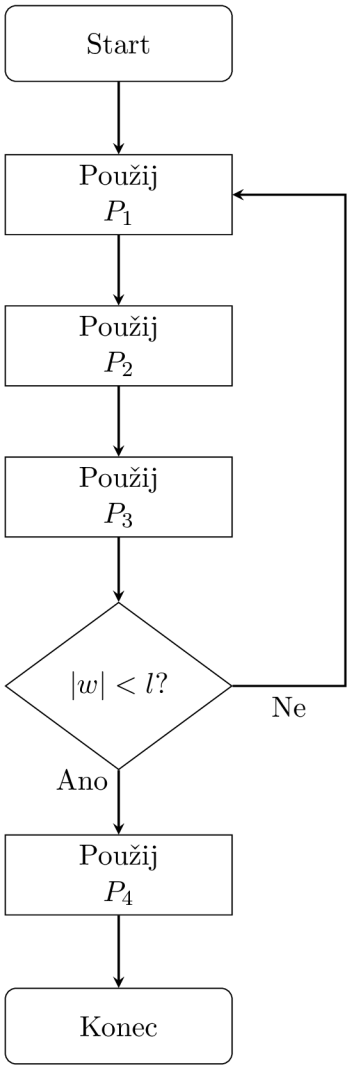
Příklad 2.3.1. Uvažujme známý kontextový jazyk

$$MIX = \big\{w \mid w \in \{a, b, c\}^*,\ |a| = |b| = |c|\big\}.$$

Tento jazyk přijímá pouze takové řetězce, které obsahují pouze stejný počet terminálů „a“, „b“ a „c“. Jedná se o silně nedeterministický jazyk, který je velmi těžké generovat běžnými systémy. My si zavedeme HCD gramatický systém, který toho po drobných úpravách bude schopen.

$$\begin{aligned}
\sigma_4 &= \big\{\{S_1, A, B, C\}, \{a, b, c\}, S_1, (P_1, =1), (P_2, =1), (P_3, =1), (P_4, t)\big\},\\
P_1 &= \{S_1 \to S_1 A S_1\},\\
P_2 &= \{S_1 \to S_1 B S_1\},\\
P_3 &= \{S_1 \to S_1 C S_1\},\\
P_4 &= \{S_1 \to \varepsilon,\ A \to a,\ B \to b,\ C \to c\}.
\end{aligned}$$

Pozorný čtenář si zajisté všimne, že uvedený systém negeneruje požadovaný jazyk. Ve skutečnosti platí $L(\sigma_4) = \big\{w \mid w \in \{a, b, c\}^*\big\}$, tedy rozhodně není dodrženo omezení na stejný počet terminálů. Tento nedostatek lze vyřešit velice jednoduše a elegantně pomocí vhodného kooperačního protokolu. Je mnoho způsobů, jak formálně zapsat kooperační protokol. Může se jednat o sadu matematických rovnic, modelů či klasický algoritmus. My pro názornost zvolíme právě algoritmus vyjádřený diagramem, uvedeném na obrázku 2.3.1.



Obrázek 2.3.1 Kooperační protokol τ gramatického systému σ_4.

Uvedený diagram kooperačního protokolu τ z našeho systému velice dobře odstraňuje nedeterminismus z pohledu komponent – přesně víme, kterou komponentu v jakém odvozovacím kroku použít. Odvozovat začneme s pomocí komponenty P_1, poté je nutné provést krok komponentou P_2 a do třetice přichází na řadu P_3. Pokud jsme dosáhli toho, že máme větnou formu o požadované délce, použijeme poslední ukončující komponentu. V opačném případě se vracíme na začátek, tedy opakujeme odvozování pomocí komponent P_1, P_2 a P_3. Po skončení práce komponenty P_4 stačí pouze porovnat symboly se vstupním řetězcem, a pokud se ve všech místech shodují, můžeme prohlásit že vstupní řetězec spadá do jazyka generovaného σ_4 s užitím kooperačního protokolu τ. Z teoretického pohledu je vše v pořádku a jsme hotovi, ale chceme-li použít takovýto aparát prakticky k ověření příslušnosti větné formy do jazyka MIX, je před námi ještě mnoho práce.

Zásadní problém, na který narazíme je, že matematický model pracuje na libovolné části větné formy. I když se omezíme na nejlevější derivace, pořád nelze zaručit, že se při odvozování přednostně pokryje nejlevější lexéma vstupní věty. Pokud chceme využít tento model ke kontrole syntaxe, pak je nutné mít k dispozici celou vstupní větu w. Z praktického

Krok	Vstup (ověřovaná věta)	Zásobník (větná forma)	Akce	Komponenta	Pravidlo
1	$aabcbaccb\$$	$S_1\$$	$\Rightarrow$	P_1	p_1
2	$aabcbaccb\$$	$S_1aS_1\$$	$=$		
3	$abcbaccb\$$	$S_1S_1\$$	$\Rightarrow$	P_2	p_2
4	$abcbaccb\$$	$S_1bS_1\$$	$=$		
5	$acbaccb\$$	$S_1S_1S_1\$$	$\Rightarrow$	P_3	p_3
6	$acbaccb\$$	$S_1cS_1^3\$$	$=$		
7	$abaccb\$$	$S_1^4\$$	$\Rightarrow$	P_1	p_1
8	$abaccb\$$	$S_1aS_1^4\$$	$=$		
9	$baccb\$$	$S_1^5\$$	$\Rightarrow$	P_2	p_2
10	$baccb\$$	$S_1bS_1^5\$$	$=$		
11	$accb\$$	$S_1^6\$$	$\Rightarrow$	P_3	p_3
12	$accb\$$	$S_1cS_1^6\$$	$=$		
13	$acb\$$	$S_1^7\$$	$\Rightarrow$	P_1	p_1
14	$acb\$$	$S_1aS_1^7\$$	$=$		
15	$cb\$$	$S_1^8\$$	$\Rightarrow$	P_2	p_2
16	$cb\$$	$S_1bS_1^8\$$	$=$		
17	$c\$$	$S_1^9\$$	$\Rightarrow$	P_3	p_3
18	$c\$$	$S_1cS_1^9\$$	$=$		
19	$\$$	$S_1^{10}\$$	$\Rightarrow^t$	P_4	p_4
20	$\$$	$\$$	$=$		
$\$$:	$\emptyset$	$\emptyset$	Přijato		

Tabulka 2.3.1 Ukázka práce HCD analyzátoru na vstupní větě. Znak $\Rightarrow$ reprezentuje derivační krok provedený uvedenou komponentou systému, $\Rightarrow^t$ symbolizuje provedení ukončující derivace a $=$ reprezentuje porovnání vstupního terminálu s terminálem na zásobníků a v případě rovnosti jejich odstranění z větných forem.

pohledu jsou pravidla tvaru $S_1 \to S_1XS_1$, respektive $X \to y$, kde $X \in \{A, B, C\}$ a $y \in \{a, b, c\}$, redundantní. Naopak nám ztěžují deterministické zpracování vstupu, a proto, aniž bychom pozměnili výsledný jazyk, je nahradíme pravidly tvaru $S_1 \to y$, tedy $P_1 = \{S_1 \to a\}$, $P_2 = \{S_1 \to b\}$, $P_3 = \{S_1 \to c\}$ a $P_4 = \{S_1 \to \varepsilon\}$. Nyní už je z kooperačního protokolu zřejmé i jaké pravidlo vybrané komponenty se využije.

V tabulce 2.3.1 je demonstrován způsob práce analyzátoru založeného na uvedeném HCD systému s využitím kooperačního protokolu τ. Začíná se odvozovat komponentou P_1, proto zaměníme S_1 za S_1aS_1 v odvozované větné formě. Následně najdeme ve vstupní větě w nejlevější terminál „a“ a odebereme jej z ní. Pokud tam terminál „a“ není přítomen, jedná o syntaktickou chybu. Stejný postup opakujeme pro P_2 a P_3 s terminály „b“ a „c“. Nyní vezmeme větu w a otestujeme, zdali je neprázdná, pokud je $w = \varepsilon$ (jinak řešeno první terminál je \$), použijeme poslední komponentu a prohlásíme, že věta je přijata. Pokud $w \neq \varepsilon$, znovu aplikujeme trojici komponent v pořadí P_1, P_2 a P_3 a tento test opakujeme.

Z praktického hlediska není toto řešení optimální. Zjistit počet výskytů symbolů ve větě je z pohledu programovacího jazyka triviální úloha. Zde jsme chtěli demonstrovat především to, že si lze poradit pomocí CD gramatických systémů i s problematickými kontexto-

vými jazyky. Později se v této práci budeme hlouběji věnovat zpracování deterministických kontextových jazyků pomocí HCD systémů, což je oblast, kde se dají tyto systémy hojně využít. Uvedený jazyk MIX patří sice do rodiny kontextových jazyků, ale nepatří do větve deterministických kontextových jazyků, a proto je jeho zpracování pomocí HCD poněkud obtížné.

3 Paralelně komunikující gramatické systémy

Dalším typem gramatických systémů jsou systémy složené z paralelně pracujících gramatik (dále jen *PC* gramatické systémy). Tyto systémy navazují na tzv. třídní model zmíněný v kapitole 1. Systém se rovněž skládá z komponent tvořených množinami pravidel, ale na rozdíl od CD gramatických systémů má každá komponenta k dispozici svoji vlastní větnou formu. Všechny komponenty jsou aktivní zároveň a pro zajištění konzistence se komponenty synchronizují podle hodin, tj. každá komponenta provede v jednom taktu hodin nějakou derivaci. Komponenty mezi sebou mohou komunikovat a vyměňovat si informace, respektive větné formy.

Definice 3.0.1. PC gramatický systém[4] je uspořádaná n-tice

$$\sigma = \big(N, T, K, (S_1, P_1), \ldots, (S_n, P_n)\big),\ n \geq 1,$$

kde N je abeceda *neterminálů*, T je abeceda *terminálů*, $K = \{Q_1, Q_2, \ldots, Q_n\}$ je množina *dotazovacích symbolů*, přičemž $N \cap T \cap K = \emptyset$. Prvky $(S_1, P_1), \ldots, (S_n, P_n)$ budeme nazývat *komponenty* systému, kde P_i, $1 \leq i \leq n$ jsou konečné množiny *přepisovacích pravidel* nad $N \cup T \cup K$ a S_i je počáteční neterminál i-té komponenty. Přepisovací pravidla jsou ve tvaru $p : x \to y$, $x \in (N \cup T)^*$, $|x|_N \geq 1$, $y \in (N \cup T \cup K)^*$.

Definice 3.0.2. Konfigurací PC gramatického systému s n komponentami myslíme uspořádanou n-tici $(x_1, x_2, \ldots x_n)$, kde $x_i \in (N \cup T \cup K)^*$, $1 \leq i \leq n$.

Porovnáme-li strukturu PC gramatického systému s klasickou bezkontextovou gramatikou, nalezneme několik zásadních odlišností. Přibyla úplně nová abeceda – abeceda dotazovacích symbolů. Tyto symboly slouží ke komunikaci a umožňují výměnu informací mezi komponentami. Ke sdílení informací dochází poté, co některá komponenta zveřejní ve své větné formě dotazovací symbol. Tyto symboly se mohou objevit pouze na pravé straně pravidel. Konfigurace již není jediný řetězec, ale celá n-tice – každá komponenta má svůj vlastní konfigurační řetězec. Stejně tak má každá komponenta svůj vlastní počáteční neterminál. Abecedy N a T jsou ovšem společné.

Definice 3.0.3. Nechť $\chi_1 = (x_1, x_2, \ldots, x_n)$ a $\chi_2 = (y_1, y_2, \ldots, y_n)$ jsou 2 konfigurace PC gramatického systému σ a $\chi \notin T^*$. Potom řekneme, že PC gramatický systém provede derivační krok[4] z konfigurace χ_1 do konfigurace χ_2, zapsáno $\chi_1 \Rightarrow \chi_2$, pokud

(i) $\forall i : 1 \leq i \leq n$ platí $x_i \in (N \cup T)^*$ a zároveň $\forall i : 1 \leq i \leq n$ lze provést derivační krok $x_i \Rightarrow y_i$ s využitím některého pravidla z P_i nebo $x_i = y_i \in T^*$,

(ii) Existuje takové i, $1 \leq i \leq n$, že platí $|x_i|_K \geq 1$. Pro každé takové i, $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, $t \geq 1$, kde $z_j \in (N \cup T)^*$, $1 \leq i, j \leq t+1$, $t \in \mathbb{N}$, nalezneme y_i. Pokud x_{i_j}, že $|x_{i_j}|_K = 0$ pro $\forall j : 1 \leq j \leq t$, pak $y_i = z_1 x_{i_1} z_2 1 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ a $y_{i_j} = S_{i_j}$. Pokud $|x_{i_j}|_K \geq 1$, pak $y_i = x_i$.

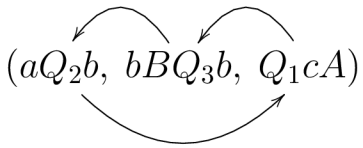
Definice 3.0.3 popisuje způsob odvozování věty PC gramatickým systémem. Jestliže konfigurace $(x_1, x_2, \ldots, x_n)$ neobsahuje žádný dotazovací symbol Q_i (bod (i)), pak systém provádí derivační kroky $x_i \Rightarrow y_i$ každou komponentou systému P_i samostatně – každá komponenta aplikuje na svoji větnou formu právě jedno přepisovací pravidlo. U systému se třemi komponentami by došlo ke třem náhradám, v každém řetězci konfigurace jedna. Vyskytuje-li se v konfiguraci dotazovací symbol, využije se bodu (ii), tedy provede se komunikační krok. Z definice rovněž plyne, že komunikační kroky mají přednost před derivačními. Dokud jsou v některém řetězci přítomny dotazovací symboly, nelze aplikovat další přepisovací pravidla. V komunikačním kroku jsou všechny výskyty dotazovacího symbolu Q_i, jež reprezentuje žádost o komunikaci s i-tou komponentou, nahrazeny současnou větnou formou i-té komponenty (řetězec x_i), vzápětí je větná forma i-té komponenty nahrazena jejím počátečním neterminálem. Poslední část bodu (ii) udává, že pokud řetězec x_i sám obsahuje dotazovací symboly, pak se tato náhrada neuskuteční a je pozdržena od dalšího kroku.

Z tohoto způsobu práce plyne nebezpečí uváznutí systému v mrtvém bodě. To může nastat pokud:

(i) v konfiguraci $(x_1, x_2, \ldots, x_n)$ není přítomen dotazovací symbol, řetězec x_i obsahuje neterminál, avšak žádné pravidlo z P_i nelze aplikovat, nebo

(ii) dojde k zacyklení komunikace.

Příkladem bodu (ii) je situace na obrázku 3.0.1, kdy P_i obsahuje Q_2, P_2 obsahuje Q_3, P_3 obsahuje Q_1. V takovém případě nemůže být proveden žádný komunikační krok a kvůli prioritě komunikace jsou zablokovány i možné derivační kroky.



Obrázek 3.0.1 Zacyklení PC gramatického systému během komunikačního kroku.

PC gramatické systémy můžeme rozčlenit na *centralizované* a *decentralizované*. Pro první jmenované, značené CPC, je typickým rysem, že dotazovací symboly produkuje pouze jedna (centrální) komponenta systému. U decentralizovaných systémů, značených DPC, je přítomno více komponent, jež mohou produkovat dotazovací symboly. Je zřejmé, že problém s možným zacyklením komunikace nastává pouze u decentralizovaných systémů.

PC gramatický systém může provádět derivace ve dvou módech – *vracejícím* (k axiomu) a *nevracejícím*. Ve vracejícím módu se provede komunikační krok přesně podle bodu (ii) definice 3.0.3 – nejprve je provedena náhrada všech výskytů dotazovacích symbolů Q_i řetězcem x_i a následně je samotný řetězec x_i nahrazen axiomem S_i. To znamená, že i-tá komponenta po poskytnutí své větné formy započne práci zcela od začátku. V nevracejícím módu se řetězec x_i axiomem nenahrazuje, tj. i-tá komponenta pokračuje v práci na stejné větné formě. Jinak řečeno, po provedení komunikačním kroku v nevracejícím módu zůstává

řetězec x_i v konfiguraci v nezměněné podobě. Tohoto chování docílíme, jestliže z bodu (ii) definice 3.0.3 vypustíme podmínku $y_{i_j} = S_{i_j}$. Derivace ve vracejícím módu budeme značit $\Rightarrow_r$, derivace v nevracejícím $\Rightarrow_{nr}$. Není překvapivé, že máme-li k dispozici 2 módy práce, je možné, aby jeden PC gramatický systém generoval 3 různé jazyky – s užitím vracejících, nevracejících a kombinovaných derivací.

Definice 3.0.4. Jazyk generovaný PC gramatickým systémem σ, značený $L(\sigma)$, jazyk generovaný PC gramatickým systémem σ ve vracejícím módu, značený $L_r(\sigma)$, a jazyk generovaný PC gramatickým systémem σ v nevracejícím módu, značený $L_{nr}(\sigma)$, jsou definovány[4]:

$$\begin{aligned} L(\sigma) &= \{x \in T^* \mid (S_1, S_2, \ldots, S_n) \Rightarrow^* (x, y_2, \ldots, y_n),\ y_i \in N \cup T \cup K,\ 2 \leq i \leq n\}, \\ L_r(\sigma) &= \{x \in T^* \mid (S_1, S_2, \ldots, S_n) \Rightarrow_r^* (x, y_2, \ldots, y_n),\ y_i \in N \cup T \cup K,\ 2 \leq i \leq n\}, \\ L_{nr}(\sigma) &= \{x \in T^* \mid (S_1, S_2, \ldots, S_n) \Rightarrow_{nr}^* (x, y_2, \ldots, y_n),\ y_i \in N \cup T \cup K,\ 2 \leq i \leq n\}. \end{aligned}$$

Z definice 3.0.4 je patrné, že jazyk generovaný první komponentou PC gramatického systému je ekvivalentní s jazykem generovaným celým systémem. Z toho důvodu se první komponenta označuje jako *master*. Ve chvíli, kdy je řetězec přijat první komponentou, není složení ostatních větných forem podstatné (mohou obsahovat neterminály). Činnost PC gramatického systému ověříme na příkladu.

Příklad 3.0.5. Uvažujme jazyk $L_1 = \{a^{2^n} \mid n \geq 1\}$. Nyní sestrojíme odpovídající PC gramatický systém, který generuje tento bezkontextový jazyk.

$$\begin{aligned} \sigma_5 &= \big(\{S_1, \overline{S_1}, S_2\}, \{a\}, \{Q_1, Q_2\}, (S_1, P_1), (S_2, P_2)\big), \\ P_1 &= \{S_1 \to Q_2Q_2Q_2Q_2\overline{S_1},\ S_1 \to S_1,\ \overline{S_1} \to \overline{S_1},\ \overline{S_1} \to \varepsilon\}, \\ P_2 &= \{S_2 \to S_2,\ S_2 \to Q_1,\ S_1 \to aaaa,\ \overline{S_1} \to \varepsilon\}. \end{aligned}$$

Čtenář si může snadno ověřit, že $L_r(\sigma_5) = L_1$. Systém nedeterministicky provede posloupnost derivací:

$$\begin{aligned} (S_1, S_2) &\Rightarrow_r (S_1, Q_1) \Rightarrow_r (S_1, S_1) \Rightarrow_r (Q_2Q_2Q_2Q_2\overline{S_1}, aaaa) \Rightarrow_r (a^{16}\overline{S_1}, S_2) \\ &\Rightarrow_r (a^{16}\overline{S_1}, Q_1) \Rightarrow_r (S_1, a^{16}\overline{S_1}) \Rightarrow_r (Q_2^4\overline{S_1}, a^{16}) \\ &\Rightarrow_r (a^{64}\overline{S_1}, S_2) \Rightarrow_r (a^{64}\overline{S_1}, Q_1) \Rightarrow_r (S_1, a^{64}\overline{S_1}) \\ &\Rightarrow_r (Q_2^4\overline{S_1}, a^{64}) \Rightarrow_r (a^{256}\overline{S_1}, S_2) \Rightarrow_r (a^{256}, S_2) \end{aligned}$$

Jak demonstruje ukázka při odvozování pomocí PC gramatických systémů hrají významnou roli tzv. čekací pravidla, bez nich by systém σ_5 přijímal pouze prázdný jazyk. Odvozování probíhá zcela nedeterministicky, takže je více způsobů (kombinací pravidel), jak se dobrat ke stejnému řetězci. Systém σ_5 z ukázky je decentralizovaný a pracuje ve vracejícím módu. V dalším příkladu si demonstrujeme práci centralizovaného systému.

Příklad 3.0.6. Uvažujme jazyk $L_2 = \{a^i b^j c^k | i \neq j,\ i \neq k,\ j \neq k,\ i, j, k \geq 3\}$. Tento jazyk, obsahující řetězce s různým počtem výskytů terminálů $\{a, b, c\}$, rovněž není bezkontextový. Odpovídající gramatický systém může být:

$$\begin{aligned} \sigma_6 &= \big(\{S_1, S_2, S_3, S_4, A, B, C\}, a, b, c, Q_2, Q_3, Q_4, (S_1, P_1), (S_2, P_2), (S_3, P_3), (S_4, P_4)\big), \\ P_1 &= \{S_1 \to ABC,\ A \to Q_2,\ B \to Q_3,\ C \to Q_4,\ A \to A,\ B \to B, \\ &\qquad C \to C,\ S_2 \to a,\ S_3 \to b,\ S_4 \to c\}, \\ P_2 &= \{S_2 \to aS_2\}, \end{aligned}$$

$P_3 = \{S_3 \to aS_3\}$,
$P_4 = \{S_2 \to aS_4\}$.

Tento systém nedeterministicky provede posloupnost derivací:

$$\begin{aligned}(S_1, S_2, S_3, S_4) &\Rightarrow_{nr} (ABC, aS_2, bS_3, cS_4) \Rightarrow_{nr} (AQ_3C, aaS_2, bbS_3, ccS_4)\\ &\Rightarrow_{nr} (AbbS_3C, aaS_2, bbS_3, ccS_4) \Rightarrow_{nr} (Q_2bbS_3C, aaaS_2, bbbS_3, cccS_4)\\ &\Rightarrow_{nr} (aaaS_2bbS_3C, aaaS_2, bbbS_3, cccS_4)\\ &\Rightarrow_{nr} (aaaS_2bbS_3C, aaaaS_2, bbbbS_3, ccccS_4)\\ &\Rightarrow_{nr} (aaaS_2bbS_3Q_4, aaaaaS_2, bbbbbS_3, cccccS_4)\\ &\Rightarrow_{nr} (aaaS_2bbS_3cccccS_4, aaaaaS_2, bbbbbS_3, cccccS_4)\\ &\Rightarrow_{nr} (aaaS_2bbS_3c^6, a^6S_2, b^6S_3, c^6S_4) \Rightarrow_{nr} (a^4bbS_3c^6, a^7S_2, b^7S_3, c^7S_4)\\ &\Rightarrow_{nr} (a^4b^3c^6, a^8S_2, b^8S_3, c^8S_4).\end{aligned}$$

Pomocí derivací v nevracejícím módu jsme odvodili řetězec $a^4b^3c^6 \in L_2$, pro názornost zkusíme nyní odvodit stejný řetězec ve vracejícím módu.

$$\begin{aligned}(S_1, S_2, S_3, S_4) &\Rightarrow_{r} (ABC, aS_2, bS_3, cS_4) \Rightarrow_{r} (AQ_3C, aaS_2, bbS_3, ccS_4)\\ &\Rightarrow_{r} (AbbS_3C, aaS_2, S_3, ccS_4) \Rightarrow_{r} (Q_2bbS_3C, aaaS_2, bS_3, cccS_4)\\ &\Rightarrow_{r} (aaaS_2bbS_3C, S_2, bS_3, cccS_4) \Rightarrow_{r} (aaaabbS_3C, aS_2, bbS_3, ccccS_4)\\ &\Rightarrow_{r} (aaaabbS_3Q_4, aaS_2, bbbS_3, ccccS_4) \Rightarrow_{r} (a^4b^2S_3c^5S_4, a^2S_2, b^3S_3, S_4)\\ &\Rightarrow_{r} (a^4b^3c^5S_4, a^3S_2, b^4S_3, cS_4) \Rightarrow_{r} (a^4b^3c^6, a^4S_2, b^5S_3, c^2S_4).\end{aligned}$$

Pozorný čtenář si může všimnout, že každý ze symbolů z K se může objevit ve větné formě pouze jednou při odvozování. Moment zveřejnění tohoto symbolu určuje délku iterace příslušného terminálu. Protože v jednom kroku může být zveřejněn pouze jeden dotazovací symbol, nemůže nastat situace, kdy $|a| = |b|$, $|a| = |c|$ nebo $|b| = |c|$. Zároveň tímto získáváme další zajímavou vlastnost: $L_r(\sigma_6) = L_{nr}(\sigma_6) = L_2$ – tento systém generuje stejný jazyk ve vracejícím i nevracejícím módu.

Nyní se podívejme na otázku síly PC gramatických systémů. Nejprve klasifikujeme PC gramatické systémy do několika skupin. Označením CPC rozumíme centalizované PC gramatické systémy. Přidáme-li písmeno „N“ před uvedené označení, pak tím rozumíme PC gramatické systémy pracující v nevracejícím módu – takto dostáváme systémy typu NPC a NCPC. Pro rodiny jazyků generované PC gramatickými systémy použijeme označení X_nY, kde $X \in \{PC, CPC, NPC, NCPC\}$ a $Y \in \{REG, LIN, CF, CS, CS^\lambda\}$. X značí povolený typ PC systému, Y značí typ použitých pravidel – po řadě regulární, lineární, bezkontextová či kontextová. Zkratka CS^λ připouští pravidla libovolného typu. $n \in \mathbb{N}$ značí stupeň sytému, respektive počet koponent. Pokud není počet komponent omezen, píšeme ∞. Výsledky z [4] shrnuje následující věta.

Věta 3.0.7. *Generativní kapacitu PC gramatických systému lze popsat relacemi:*

(i) $Y_nCS^\lambda = RE, \forall n \wedge Y \in PC, CPC, NPC, NCPC,$

(ii) $NPC_\infty CF \subseteq PC_\infty CF,$

(iii) $MAT \subset PC_\infty CF$,

(iv) $CPC_\infty REG \subset MAT_{fin}$,

(v) $Y_n CS = Y_\infty CS, \forall n \geq 1, Y \in \{CPC, NCPC\}$,

(vi) $CS = PC_1 CS = PC_2 CS \subset PC_3 CS = PC_\infty CS = RE$,

(vii) $CS = NPC_1 CS \subset NPC_2 CS = NPC_\infty CS = RE$.

Z věty 3.0.7 je na první pohled patrná velká generativní síla PC gramatických systémů. Za použití pouze bezkontextových pravidel dostáváme nástroj silnější, než jsou maticové gramatiky. Použitím kontextových pravidel můžeme ještě zvýšit sílu těchto systémů. Necentralizované systémy využívající kontextových pravidel o třech komponentách pracující ve vracejícím módu či systémy se dvěma komponentami pracující v nevracejícím módu jsou dostatečně silné k popisu rekurzivně spočetných jazyků.

Významnou vlastností výše zmíněných PC gramatických systémů je synchronizace podle hodin, které měří čas všem komponentám stejně. V každém hodinovém cyklu musí každá komponenta využít právě jedno pravidlo (pokud právě neprobíhá komunikace). Pokud tyto hodiny odstraníme, dostáváme *nesynchronizovaný PC gramatický systém*. To sice vede ke zjednodušení systému, neboť je možné vypustit čekací pravidla tvaru $A \to A$, $A \in N$, avšak v [4] je ukázáno, že desynchronizace zároveň snižuje generativní sílu.

3.1 PC gramatické systémy komunikující příkazem

V doposud zmiňovaných PC gramatických systémech komunikaci vždy inicioval příjemce zprávy tím, že ve své větné formě zavedl dotazovací symbol. Tento způsob komunikace se označuje jako *komunikace na žádost*. Neméně matematicky zajímavý je i obrácený případ, kdy komunikaci iniciuje odesilatel. Takovou komunikaci budeme nazývat *komunikace příkazem*, systémy tuto komunikaci využívající označíme *CCPC*.

Definice 3.1.1. PC gramatický systém komunikující příkazem [4] je uspořádaná n-tice $\sigma = \big(N, T, (S_1, P_1, R_1), \ldots, (S_n, P_n, R_n)\big)$, $n \geq 1$, kde N je abeceda *neterminálů*, T je abeceda *terminálů* a (S_i, P_i, R_i), $1 \leq i \leq n$ jsou *komponenty* systému, kde $S_i \in N$ je *počáteční neterminál*, P_i je množina *přepisovacích pravidel* nad $N \cup T$ a $R_i \subseteq (N \cup T)^*$ je *selektor jazyka* i-té komponenty.

Definice 3.1.2. Konfigurací CCPC gramatického systému[4] myslíme uspořádanou n-tici $(x_1, x_2, \ldots, x_n)$, kde $x_i \in N \cup T$, $1 \leq i \leq n$.

U systémů komunikujících příkazem je vypuštěna abeceda dotazovacích symbolů, roli komunikačního prostředku hraje selektor jazyka, jenž je definován pro každou komponentu zvlášť. Ten funguje jako filtr – určuje, o jaké zprávy má příjemce zájem. Příjemce při výměně informací obdrží pouze takové větné formy (zprávy), které spadají do jazyka specifikovaného jeho selektorem. Způsob odvozování je dosti odlišný od klasického PC gramatického systému, proto si jej rozebereme podrobněji.

Definice 3.1.3. Derivační krok provedený CCPC gramatickým systémem σ s n komponentami[4], značený $\Rightarrow$, je definován:

$$(x_1, \ldots, x_n) \Rightarrow (y_1, \ldots, y_n), \text{ pokud } \forall i : 1 \leq i \leq n \text{ platí } x_i \Rightarrow^* y_i \text{ a zároveň}$$
$$\text{neexistuje takové } z \in (N \cup T)^*, \text{ že } y_i \Rightarrow z_i.$$

Definice 3.1.4. Nechť $\delta(x_i, j)$ je funkce dvou proměnných, definovaná pro $1 \leq i, j \leq n$ jako

$$\delta(x_i, j) = \begin{cases} \varepsilon & \text{pro} x_i \notin R_j \vee i = j, \\ x_i & \text{pro} x_i \in R_j \wedge i \neq j \end{cases}$$

Dále řekneme, že pro $1 \leq j \leq n$ je sekvence

$$\triangle(j) = \delta(x_1, j)\delta(x_2, j) \ldots \delta(x_n, j)$$

celková zpráva přijímaná j-tou komponentou. Rovněž pro $1 \leq j \leq n$ zavedeme *celkovou zprávu odesílanou* j-tou komponentou jako

$$\delta(j) = \delta(x_j, 1)\delta(x_j, 2) \ldots \delta(x_j, n).$$

Komunikační krok, značený $(x_1, \ldots, x_n) \vdash (y_1, \ldots, y_n)$, je pro $1 \leq i \leq n$ definován

$$y_i = \begin{cases} \triangle(i) & \text{pro } \triangle(i) \neq \varepsilon, \\ x_i & \text{pro } \triangle(i) = \varepsilon \wedge \delta(i) = \varepsilon, \\ S_i & \text{pro } \triangle(i) = \varepsilon \wedge \delta(i) \neq \varepsilon. \end{cases}$$

Derivační krok CCPC gramatického systému je již pro nás povědomý – každá komponenta provede na své vlastní větné formě ukončující derivaci. Tím je na rozdíl od systémů komunikujících na žádost vyloučeno, že systém se zasekne během derivačního kroku, neboť je vždy možné provést derivaci délky 0. Výměna informací probíhá decentralizovaně v komunikačním kroku. Každá komponenta může posílat zprávy všem ostatním (nikoliv sama sobě), respektive může obdržet zprávy od všech ostatních komponent. Pro každou komponentu systému se sestaví celková zpráva. Větné formy všech ostatních komponent (nikoliv větná forma příjemce) jsou porovnány se selektorem jazyka (filtrem) příjemce, a pokud splňují zadaný tvar, jsou přidány do celkové zprávy, jež se po prověření všech komponent odešle příjemci. Jinak řečeno, příjemce obdrží celkovou zprávu vytvořenou konkatenací všech větných forem ostatních komponent, které prošly jeho filtrem. Filtr může být tvořen například regulárním výrazem. Příjemce po obdržení neprázdné celkové zprávy nahradí svoji větnou formu obsahem zprávy. Pokud je komponenta odesilatelem zprávy, avšak žádnou zprávu sama nepřijímá, je její větná forma v komunikačním kroku nahrazena počátečním neterminálem. V posledním případě, kdy není komponenta ani příjemcem ani odesilatelem, zůstává její větná forma nezměněna. Podívejme se nyní, jak se kombinují komunikační a derivační kroky během odvozování.

Definice 3.1.5.

$$L(\sigma) = \Big\{ w \in T^* \mid (S_1, \ldots, S_n) \Rightarrow (x_1^{(1)}, \ldots, x_n^{(1)}) \vdash (y_1^{(1)}, \ldots, y_n^{(1)}) \Rightarrow (x_1^{(2)}, \ldots, x_n^{(2)}) \vdash (y_1^{(2)}, \ldots, y_n^{(2)}) \Rightarrow \ldots \Rightarrow (y_1^{(s)}, \ldots, y_n^{(s)}),\ s \geq 1, w = x_1^{(s)} \Big\}.$$

Jazyk generovaný CCPC gramatickým systémem je podle definice 3.1.5 sekvence derivačních a komunikačních kroků, které se pravidelně střídají. Pozornému čtenáři jistě neunikne, že systém se může zacyklit v případě, že se v komunikačním kroku nepošle ani jedna zpráva – to lze ovšem snadno detekovat a o zkoumané větné formě lze prohlásit, že nespadá do generovaného jazyka. Za povšimnutí stojí i fakt, že sekvence končí derivačním

nikoli komunikačním krokem. Stejně jako u klasických PC systémů platí, že jazyk generovaný první komponentou (*master*) je rovněž jazykem generovaným celým systémem. Pro lepší porozumění, jak CCPC gramatický systém pracuje, si na příkladu ukážeme činnost takového systému.

Příklad 3.1.6. Uvažujme jazyk $L_{cc} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$, který je kontextový. Odpovídající CCPC gramatický systém je definován jako

$$\sigma_7 = \big(\{S_1, \overline{S_1}, S_2, S_3, X, Y, M, T, U, V, W\}, \{a, b, c, d\}, (S_1, P_1, R_1), (S_2, P_2, R_2), (S_3, P_3, R_3)\big),$$
$$P_1 = \{S_1 \to S_1,\ S_1 \to NM,\ S_1 \to N\overline{S_1}M,\ \overline{S_1} \to X\overline{S_1},\ \overline{S_1} \to \overline{S_1}Y,$$
$$\overline{S_1} \to XY|X|Y,\ T \to a,\ U \to b,\ V \to c,\ W \to d\},$$
$$P_2 = \{S_2 \to S_2,\ X \to a,\ Y \to b,\ N \to T,\ M \to U\},$$
$$P_3 = \{S_3 \to S_3,\ X \to c,\ Y \to d,\ N \to V,\ M \to W\},$$
$$R_1 = \{Ta^*b^*U,\ Vc^*d^*W\},$$
$$R_2 = \{NX^*Y^*M\},$$
$$R_3 = \{NX^*Y^*M\}.$$

Systém nedeterministicky provede posloupnost derivací

$$(S_1, S_2, S_3) \Rightarrow (NXYYM, S_2, S_3) \vdash (S_1, NXYYM, NXYYM) \Rightarrow (S_1, TabbU, VcddW)$$
$$\vdash (TabbUVcddW, S_2, S_3) \Rightarrow (aabbbccddd, S_2, S_3).$$

Dostáváme se k otázce síly CCPC gramatických systémů. Zavedeme si označení $CCPC_nX$ pro rodiny jazyků generované CCPC gramatickými systémy o nejvýše n komponentách, kde $X \in \{REG, CF\}$ udává typ použitých pravidel. Pro CF uvažujme pouze použití bezkontextových pravidel neobsahujících ε. Pokud počet komponent není omezen, píšeme $CCPC_\infty X$. V [4] byly publikovány výsledky shrnuté větou 3.1.7.

Věta 3.1.7. *Generativní kapacitu PC gramatických systémů lze popsat relacemi:*

(i) $REG = CCPC_1REG = CCPC_2REG \subset CCPC_3REG = CCPC_\infty REG = CS,$

(ii) $CF = CCPC_1CF \subset CCPC_2CF = CCPC_\infty CF = CS.$

Získaná hierarchie rodin jazyků je pouze dvouúrovňová, překvapivě stejnou sílu dostaneme užitím regulárních či bezkontextových pravidel. K popisu kontextových jazyků nám stačí systémy o třech komponentách využívající pravě lineárních pravidel či systémy o dvou komponentách využívající bezkontextová pravidla neobsahující ε. Pokud povolíme ε-pravidla, dostáváme další zajímavé výsledky:

(i) $CS^\varepsilon \subseteq CCPC_3REG^\varepsilon$,

(ii) $CS^\varepsilon \subseteq CCPC_2CF^\varepsilon$.

Tedy přidáním ε-pravidel do CCPC gramatických systémů dostáváme nástroj silnější než samotné kontextové jazyky. Důkazy inkluzí jsou k nahlédnutí v [2].

4 Zvýšení generativní síly bezkontextových metod užitím CD gramatických systémů

Velikým cílem stojícím na počátku výzkumu gramatických systémů bylo zvýšení generativní síly bezkontextových gramatik, respektive možnost zpracovat kontextové jazyky za pomoci bezkontextových pravidel. Pro analýzu kontextových jazyků lze samozřejmě použít kontextové gramatiky, avšak této možnosti se dále věnovat nebudeme. Naopak se pokusíme využít toho, že na zpracování bezkontextových jazyků bylo navrženo mnoho relativně jednoduchých metod (např. LL prediktivní syntaktická analýza, LR syntaktická analýza, precedenční syntaktická analýza, metody obecné syntaktické analýzy...), ze kterých můžeme vycházet a pokusíme se zvýšit jejich generativní sílu. Pro detailní seznámení s těmito metodami může čtenář využít [1] nebo česky psanou alternativu [3].

4.1 Transformace homogenního gramatického systému a zisk determinismu

Zisk determinismu v obecné rovině je téma komplikované a rozsáhlé. Naši pozornost na něj musíme upírat, jestliže chceme zpracovávat jazyky generované nedeterministickými matematickými modely i na počítači. V následujících úvahách se omezíme na CD gramatické systémy. V první části uvedeme transformaci pro homogenní systémy pracující pouze v módu $=k$. Následně tyto poznatky zobecníme i pro hybridní CD gramatické systémy. Před samotnou transformací je nutné zavést několik nových pojmů, některé definovat znovu jiným způsobem a jiné pojmy upřesnit. Předpokládáme, že se čtenář již setkal s definicí bezkontextové gramatiky (dále jen BKG) a stejně tak derivačního kroku touto gramatikou provedeného. Pojem sekvence derivačních kroku není zajisté nic nového, my si jej nyní rozšíříme pro potřeby gramatických systémů. Bez újmy na obecnosti budeme pracovat pouze s nejlevějšími derivacemi, pro přehlednost píšeme pouze $\Rightarrow$ místo zavedeného $\Rightarrow_{lm}$.

Definice 4.1.1. Nechť $G = (N, T, S, P)$ je BKG. Pokud G může provést sekvenci derivačních kroků $x_1A_1x_2A_2x_3 \dots x_nA_nx_{n+1} \Rightarrow^n x_1y_1x_2y_2x_3 \dots x_ny_nx_{n+1}[p_1, p_2, \dots, p_n]$, kde $A_i \in N$, $x_i, y_i \in (N \cup T)^*$, $1 \leq i \leq n$ pro nějaké $n \in \mathbb{N}$, pak A_1 je řídící neterminál sekvence derivačních kroků, značeno $\overrightarrow{A_1}$. Dále řekneme, že derivace probíhá podle $\overrightarrow{A_1}$. Pokud nějaká konfigurace u derivuje konfiguraci v podle $\overrightarrow{A_1}$ v n krocích za použití pravidel $p_1, \dots, p_n$, pak píšeme $u \Rightarrow v[\overrightarrow{A_1} \,|\, p_1, \dots, p_n]$.

Řídícím neterminálem sekvence derivačních kroků je podle definice 4.1.1 vždy nejlevější neterminál přepsaný během odvozování. V klasické LL gramatice je řídícím neterminálem vždy nejlevější neterminál větné formy. Definice explicitně připouští zobecnění – nalevo od řídícího neterminálu se mohou vyskytovat jiné neterminály, které nebyly přepsány během derivačního kroku, tedy řídící neterminál nemusí být nejlevějším neterminálem ve větné formě před derivací. Tohle zobecnění bude velice užitečné při práci s vícekrokovými derivacemi komponent ($=k$) gramatických systémů.

Gramatické systémy si můžeme rozdělit do dvou základních skupin podle jejich chování, respektive podle toho, zdali obsahují pravidla specifického tvaru.

Definice 4.1.2. Nechť $\sigma = (N, T, S, P_1, \ldots, P_n)$ je CD gramatický systém. Řekněme, že gramatický systém *diverguje*, pokud pro nějakou komponentu platí $A \Rightarrow^f_{P_i} xAyAz \in P_i \vee A \Rightarrow^f_{P_i} xB^mAC^ly \in P_i$, $A, B, C \in N$, $f \in D$, $x, y, z \in (N \cup T)^*$, $1 \leq i \leq n$, $m, l \in \mathbb{N}$, v opačném případě systém *konverguje*.

Divergující systém je takový systém, v jehož větných formách může neomezeně narůstat počet stejných neterminálů. Jako příklad poslouží libovolný systém s pravidlem tvaru $A \to AA \in P_i$. Pravidla tvaru $A \to BA$ a $A \to AB$ narušují konvergenci konvergenci systému pro mód práce $=k$, $k \geq 2$. U konvergujících systémů se počet stejných neterminálů ve větné formě v průběhu odvozování nemění, nebo snižuje. Situace, kdy se z větné formy obsahující $A \in N$ odvodí větná forma obsahující A^2, je u konvergujících systémů vyloučena, naopak u divergujících je zcela běžná. Počet jiných neterminálů se může zvyšovat libovolně u obou systémů, například pravidlo tvaru $A \to BCD$ se může nacházet i v konvergujícím systému, pokud platí $BCD \not\Rightarrow AA$. Obdobná definice platí i pro HCD gramatické systémy.

Přestože je většině čtenářů důvěrně znám pojem *LL gramatika*, neuškodí uvést definici, z níž budeme v dalších úvahách vycházet. S tímto pojmem velice úzce souvisí pro nás klíčová struktura – LL tabulka, kterou se pokusíme rozšířit. Ještě před tím je nutné definovat množinu *First*, která nám pomůže sestrojit tyto struktury korektně.

Definice 4.1.3. Nechť $G = (N, T, S, P)$ je BKG. Pro každé $x \in (N \cup T)^*$ je definováno $First(x)$ [3] jako:

$$First(x) = \{a \mid a \in T, x \Rightarrow^* ay, y \in (N \cup T)^*\}.$$

Definice 4.1.4. Nechť $G = (N, T, S, P)$ je BKG. Řekněme, že G je LL gramatika, pokud pro každý neterminál $A \in N$ a každý terminál $a \in T$ existuje maximálně jedno pravidlo tvaru $A \to x_1x_2 \ldots x_n$, $x_i \in (N \cup T)^*$, $1 \leq i \leq n$, $n \in \mathbb{N}$ pro které platí $a \in First(x_1x_2 \ldots x_n)$ [3].

Definice 4.1.5. Nechť $G = (N, T, S, P)$ je LL gramatika. Pak LL tabulkou označujeme množinu $T_{LL} = \{\alpha(a, A)\}$, kde $\alpha(a, A) = p$, $a \in T$, $A \in N$, $p \in P$ je funkce dvou proměnných mapující pravidla gramatiky G do LL tabulky podle klíčů dvou klíčů – a, A.

Jednoduše řečeno, množina $First(x)$ obsahuje všechny takové terminály, které se nacházejí na prvním místě ve větných formách odvozených z x pomocí gramatiky G. LL gramatika je BKG, pro kterou platí, že pro každý neterminál A existuje nejvýše jedno pravidlo, pomocí kterého lze odvodit terminál a jako nejlevější symbol. Tedy v BKG běžná situace – $A \to ab \in P$, $A \to ac \in P$ – nemůže v LL gramatice nastat.

Na LL tabulku se můžeme podívat jako na množinu hodnot získaných z mapovací funkce α. Definiční obor této funkce je množina dvojic – terminál, neterminál. Z matematického

pohledu se jedná o relaci $\alpha : (T \times N) \to P$. Mapovací funkce na základě vstupního terminálu a nejlevějšího neterminálu větné formy určí, jaké pravidlo se má použít v dalším derivačním kroku. Její sémantiku lze popsat následovně: pokud je na vrcholu zásobníku neterminál A a příchozí lexéma je a, použije se pravidlo p. Výsledkem této funkce je právě jedno pravidlo, což je nedostačující z pohledu gramatických systémů a vícekrokových derivací. Zobecněním mapovací funkce značně rozšíříme možnosti LL tabulky a získáme vyšší generativní sílu. Poté budeme schopni pracovat s k-kroky dlouhou derivací, nebo dokonce i hybridními gramatickými systémy. Pro tyto účely nám poslouží mapovací funkce, jejímž výstupem je n-tice pravidel, matematicky zapsáno $\alpha : (T \times N) \to (p_1, p_2, \ldots, p_n)$, $p_i \in P$ pro $1 \leq i \leq n$, $n \in \mathbb{N}$. Po aplikaci této funkce na celou definiční množinu $(T \times N)$ získáváme zdrojovou tabulku, pro prediktivní syntaktický analyzátor. Již se nejedná o klasickou LL tabulku, a proto si pro ni vymezíme nový pojem – *Grouped Left-most Left-to-right* (dále jen GLL) tabulka.

Běžnou LL tabulku můžeme sestrojit pro libovolnou LL gramatiku, ale nikoliv pro libovolnou bezkontextovou gramatiku. Velice podobné je to s GLL tabulkou. Nelze ji vytvořit pro libovolný CD gramatický systém s bezkontextovými pravidly. Musí se jednat o LL gramatický systém. Podívejme se nyní na nově definované pojmy formálně.

Definice 4.1.6. Nechť σ je gramatický systém. Řekneme, že σ je LL gramatický systém, pokud pro každý neterminál $A \in N$ a každý terminál $a \in T$ existuje maximálně jedna hodnota mapovací funkce $\alpha(a, A)$, $\alpha : (T \times N) \to P$.

Definice 4.1.7. Nechť $\sigma = (N, T, S, P_1, P_2, \ldots P_n)$ je LL CD gramatický systém. Pak GLL tabulka (*Grouped Left-most Left-to-right*), značená T_{GLL}, je definována jako

$$T_{GLL} = \big\{\alpha(a, A)\big\},$$

kde $\alpha(a, A) = (p_1, p_2, \ldots, p_k) = p_1 \wedge p_2 \wedge \ldots \wedge p_k$ pro $a \in T$, $A \in N$, $p_i \in P_j$ a pro nějaká $1 \leq j \leq n$, $1 \leq i \leq k$, $k = |\text{mode}(\sigma)|$. Hodnota k, nazývající se stupeň systému, se vypočte:

$$|\text{mode}(\sigma)| = \begin{cases} 1 & \text{pro mode}(\sigma) \in \{t\} \\ k & \text{pro mode}(\sigma) \in \{\leq k, \geq k, =k \mid k \geq 1\}. \end{cases}$$

GLL tabulka se strukturálně velmi podobná klasické LL tabulce. Pro každý vstupní terminál $a \in T$ a neterminál $A \in N$ v tabulce nalezneme nejvýše jednu hodnotu. V LL tabulce je touto hodnotou právě jedno pravidlo, v GLL je touto hodnotou právě jedna n-tice pravidel $(p_1, \ldots, p_k)$. Z kolika pravidel je tvořena n-tice je dáno stupněm systému, respektive stupněm módu, ve kterém CD gramatický systém pracuje. Není překvapivé, že pro systémy provádějící derivace o délce k (pracující v módu $=k$), je stupeň $|k|$ právě hodnota konstanty k. Například pro systém pracující v módu $=2$ je stupeň $|\sigma|$ roven 2. Ne tolik intuitivní se může zdát fakt, že stupeň systému pracujícího v módu t je 1. Nyní je zřejmé, že všechny n-tice v tabulce budou stejně dlouhé. Ještě je nutné doplnit význam těchto n-tic. Pokud je n-tice tvořena pouze jedním pravidlem p, aplikuje se na větnou formu. Nelze-li toto pravidlo aplikovat, jedná se o syntaktickou chybu. Je-li n-tice tvořena více pravidly, postupně se aplikují všechna tato pravidla na větnou formu, a to zleva doprava, počínaje p_1 a konče p_k. Nelze-li některé pravidlo aplikovat na větnou formu, jedná o syntaktickou chybu. Úspěšně provedený derivační krok je pouze takový krok, kdy byla celá n-tice pravidel použita. V případě, že není v GLL tabulce žádná hodnota pro terminál $a \in T$ a neterminál $A \in N$, pak tato kombinace také indikuje syntaktickou chybu. Rozdíl mezi LL a GLL tabulkou ilustruje obrázek 4.1.1.

N \ T	a	b
S	$\alpha(a, S)$	
X		$\alpha(b, X)$
Y	$\alpha(a, Y)$	

N \ T	a	b
S	1	
X		2
Y	3	

N \ T	a	b
S	1,2	
X		2,3
Y	4,1	

Tabulka 4.1.1 Mapovací funkce, příklad klasické LL tabulky a GLL tabulky pro CD gramatický systém stupně 2.

Takto sestavenou tabulku je možné použít v klasické prediktivní syntaktické analýze shora dolů, jinak řečeno lze sestrojit deterministický zásobníkový automat, který bude přijat jazyk daný GLL tabulkou. Nyní jsme v bodě, kdy máme definován vstup transformace – LL gramatický systém – a výstup – GLL tabulka. Než přistoupíme k transformaci samotné, je nutné zavést pomocné množiny $Follow_f^{CD}$, $f \in D$ a $Empty$.

Definice 4.1.8. Nechť $G = (N, T, S, P)$ je BKG a $X_1X_2 \dots X_n \in (N \cup T)^*$ je řetězec. $Empty(X_1X_2 \dots X_n)$[3] definujeme následovně:

$$\begin{aligned} &\textbf{if } X_1X_2 \dots X_n \Rightarrow^* \varepsilon \textbf{ then} \\ &\qquad Empty(X_1X_2 \dots X_n) = \{\varepsilon\} \\ &\textbf{else } Empty(X_1X_2 \dots X_n) = \emptyset. \end{aligned}$$

Definice 4.1.9. Nechť $\sigma = (N, T, S, P_1, P_2, \dots, P_n)$ je CD gramatický systém. Pro každý neterminál $A \in N$ je definována množina $Follow_f^{CD}(A)$, kde $f \in D$, následovně:

$$\begin{aligned} Follow_f^{CD}(A) = \big\{ a \,|\, a \in T,\ & S \Rightarrow_{P_{i_1}}^f w_1 \Rightarrow_{P_{i_2}}^f \dots \Rightarrow_{P_{i_m}}^f xAay, \\ & x, y \in (N \cup T)^*,\ m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m \big\} \\ \cup \big\{ \$ \,|\, & S \Rightarrow_{P_{i_1}}^f w_1 \Rightarrow_{P_{i_2}}^f \dots \Rightarrow_{P_{i_m}}^f xA, \\ & x \in (N \cup T)^*,\ m \geq 1,\ 1 \leq i_j \leq n,\ 1 \leq j \leq m \big\} \end{aligned}$$

Množina Empty specifikuje, jestli je možné zadaný řetězec zcela vymazat pomocí ε-pravidel libovolné komponenty či nikoliv. Množina $Follow_f^{CD}(A)$ obsahuje všechny terminály, které se v nějaké větné formě odvoditelné systémem pracujícím v módu f mohou vyskytovat za neterminálem A. Získání těchto množin je nezbytnou prerekvizitou pro samotný převod CD gramatického sytému na GLL tabulku, proto zde v následujících řádcích uvádíme algoritmy, pomocí kterých tyto množiny zkonstruujeme. Začneme algoritmem pro množinu $First$, prezentovaným v [3].

Algoritmus 4.1.1: Množina $First$

Vstup: BKG $G = (N, T, S, P)$ nebo CD gramatický systém $\sigma = (N, T, S, P_1, \dots P_n)$
Výstup: množina $First(A)$ pro každý neterminál $A \in N$

1: $\forall a \in T$: $First(a) := \{a\}$
2: $\forall A \in N$: $First(A) := \emptyset$
3: **while** některá množina $First(A)$ změněna **do**
4: **if** $A \to X_1 X_2 \dots X_{k-1} X_k \dots X_n \in P$ **then**
5: $First(A) := First(A) \cup First(X_1)$
6: **if** $Empty(X_1 X_2 \dots X_{k-1}) = \{\varepsilon\}$ **then**
7: $First(A) := First(A) \cup First(X_k)$
8: **end if**
9: **end if**
10: **end while**

Sestrojení množiny $First$ je celkem jednoduchá úloha. $First(a \in T)$ teminálu a musí nutně obsahovat terminál a samotný, protože je možné jej odvodit derivací délky 0 z a: $a \Rightarrow^0 a$. Množina každého neterminálu $A \in N$ musí obsahovat množiny $First$ všech symbolů, které se mohou vyskytovat na nejlevější pozici v některé větné formě odvozené z A. Nebudeme-li uvažovat ε-pravidla, stačí nám prověřit všechna pravidla obsahující na levé straně A. Přidáním ε-pravidel se situace mírně zkomplikuje – nyní musíme do $First(A)$ zahrnout i celou množinu prvního symbolu, který nelze z větné formy odstranit. Jestliže můžeme odstranit všechny levější symboly, je jasné, že i tento symbol může být na první pozici v odvozované větné formě. Množina $First$ se vytváří totožně pro BKG i pro gramatické systémy a při její konstrukci se můžeme na gramatický systém dívat jako na jednu gramatiku obsahující pravidla všech komponent. Množina $Follow$ (pro BKG), respektive množina $Follow_f^{CD}$ (pro CD gramatické systémy), jsou již sestrojovány zcela odlišně. Zatímco se $Follow$ se pravidla ověřují samostatně, u $Follow_f^{CD}$ se musí ověřovat po n-ticích o délce dané zvolenou délkou derivačního kroku, tedy stupněm systému. Tuto úlohu neoptimalizovaně řeší následující algoritmus, které prochází všechny možné derivace z počátečního neterminálu až do takové úrovně zanoření, ve které se již nezmění žádná z vytvořených množin.

Algoritmus 4.1.2: Množina $Follow_f^{CD}$

Vstup: CD gramatický systém $\sigma = (N, T, S, P_1, \dots P_n)$ pracující v módu $f \in D$
Výstup: množina $Follow_f^{CD}$ pro každý neterminál $A \in N$

1: $\Omega := \{S\}$, $\Theta := \emptyset$
2: $\forall A \in N$: $Follow_f^{CD}(A) := \emptyset$
3: $Follow_f^{CD}(S) := \{\$\}$
4: **while** některá množina $Follow_f^{CD}(A)$ změněna **do**
5: **foreach** $w_k \in \Omega$ **do**
6: $\Phi := \left\{ w_D \,|\, w_k \Rightarrow_{P_i}^{f} w_D,\ f \in D,\ 1 \leq i \leq n \right\}$
7: **foreach** $w_D \in \Phi$ **as** $Y_1 Y_2 \dots Y_{N-1} Y_N$ **do**
8: **if** $Y_i \in N$ **for** $1 \leq i \leq N$ **then**
9: $Follow_f^{CD}(Y_i) := Follow_f^{CD}(Y_i) \cup First(Y_{i+1})$
10: **if** $Empty(Y_{i+1} \dots Yj) = \{\varepsilon\}$ **for** $1 \leq j \leq N-1$ **then**
11: $Follow_f^{CD}(Y_i) := Follow_f^{CD}(Y_i) \cup First(Y_{j+1})$
12: **else if** $Empty(Y_{i+1} \dots Y_N) = \{\varepsilon\}$ **then**
13: $Follow_f^{CD}(Y_i) := Follow_f^{CD}(Y_i) \cup Follow_f^{CD}(w_k)$
14: **end if**
15: **end if**
16: **if** $Y_N \in N$ **then**
17: $Follow_f^{CD}(Y_N) := Follow_f^{CD}(Y_N) \cup Follow_f^{CD}(w_k)$
18: **end if**
19: **if** $w_D \notin \Omega$ **then**
20: $\Theta := \Theta \cup \{w_D\}$
21: **end if**
22: **end foreach**
23: **end foreach**
24: $\Omega := \Theta$, $\Theta := \emptyset$
25: **end while**

Tento na první pohled složitě vypadající algoritmus je důležitou součástí transformace, a proto věnujeme jeho výkladu větší pozornost. Množina Ω obsahuje větné formy derivovatelné z počátečního neterminálu pomocí gramatického systému σ v módu $f \in D$. Ze začátku do této množiny umístíme S, protože pro každý takový systém platí $S \Rightarrow^0 S$. Zároveň víme, že za každou větnou formou následuje „$“, proto ho umístíme do $Follow_f^{CD}(S)$. Množina Θ obsahuje řetězce derivované systémem v následujícím kroku algoritmu. V každém kroku algoritmu aplikujeme na všechny větné formy v Ω dostupná pravidla, tedy simulujeme všechny možné derivace proveditelné pomocí některé z komponent systému. Výsledkem této operace pro jednu větnou formu z Ω je množina Φ. Každý řetězec z Φ si poté rozložíme na jednotlivé terminály a neterminály (řádek 7). Množinu $Follow_f^{CD}$ každého neterminálu z rozložené větné formy rozšíříme (řádek 9) o množinu $First$ následujícího symbolu v této řadě. Dále najdeme ve větné formě první takový symbol, který následuje za aktuálním symbolem (Y_i) a nelze jej vymazat pomocí ε-pravidel, a celou množinu $First$ toho symbolu zahrneme do množiny $Follow_f^{CD}$ aktuálního symbolu (řádek 11). Je nutné si uvědomit, že za posledním prvkem může následovat stejná lexéma, jako za celým řetězcem, proto musíme zahrnout $Follow_f^{CD}(w_k)$ do množiny $Follow_f^{CD}$ posledního symbolu větné formy (řádek 17), či každého symbolu, který se může stát posledním (řádek 13). Na řádku 19 vytváříme množinu větných forem k prověření v následujícím kroku. Na konci každého kroku nastane záměna

množiny prověřených větných forem Ω za množinu větných forem odvozených v posledním kroku Θ. Algoritmus končí v případě, že v předchozím kroku nedošlo k žádné změně ve vytvářených množinách $Follow_f^{CD}$ a prověřili jsme všechna pravidla. Je pravděpodobné, že některé větné formy budou prověřovány vícekrát, ale k zacyklení algoritmu nemůže dojít, neboť máme pouze konečný počet terminálů a neterminálů.

Nyní už máme potřebné množiny a můžeme se podívat na samotnou transformaci CD gramatického systému na GLL tabulku. Následující transformační algoritmus je platný pro gramatické systémy pracující v módu $=k$.

Algoritmus 4.1.3: GLL Tabulka

Vstup: CD gramatický systém $\sigma = (N, T, S, P_1, \dots P_n)$ pracující v módu $f \in D$,
množina $First(x)$ pro každé $x \in (N \cup T)$,
množina $Follow_f^{CD}(X)$ pro každé $X \in N$

Výstup: GLL tabulka ve tvaru $T_{GLL} = \{\alpha(a \in T, A \in N)\}$

1: $\Omega := \{S\}$, $\Theta = \emptyset$
2: **while** $\Omega \neq \emptyset$ **do**
3: $w_k \in \Omega$, $\Omega := \Omega \setminus \{w_k\}$, $\Theta := \Theta \cup \{w_k\}$
4: $\Phi := \{(w_D, \vec{C}, r) \mid w_k \Rightarrow_{P_i}^k w_D[\vec{C} \mid r_1, r_2, \dots, r_j]$, $w_D \neq w_k$, $k = |\text{mode}(\sigma)|$,
$r = [r_1, r_2, \dots, r_j]$, $r_j \in P_i$, $1 \leq i \leq n$, $1 \leq j \leq \text{count}(\sigma)\}$
5: **foreach** $(w_D, \vec{C}, r) \in \Phi$ **do**
6: **if not** $w_D \Rightarrow_{P_i}^k T^*$ **then continue**
7: **foreach** $y \in First(\vec{C})$ **do**
8: **if** $y \in First(w_D)$ **then**
9: $\alpha(y, \vec{C}) := r$
10: **end if**
11: **end foreach**
12: **if** $Empty(w_D) = \varepsilon$ **or** $\vec{C} \to \varepsilon \in r$ **then**
13: **foreach** $y \in Follow_f^C D(\vec{C})$ **do**
14: $\alpha(y, \vec{C}) := r$
15: **end foreach**
16: **end if**
17: $\overline{w_D} := w_D - \{a \mid a \in T\}$, $\overline{w_D} \in N^*$
18: **if** $\overline{w_D} \notin \Theta$ **then**
19: $\Omega := \Omega \cup \{\overline{w_D}\}$
20: **end if**
21: **end foreach**
22: **end while**

Jádro algoritmu vychází z algoritmu 4.1.2 a mírně jej rozšiřuje. Množiny Ω a Θ obsahují speciální řetězce neterminálů – každý z těchto řetězců byl získán z některé větné formy odstraněním všech terminálů. První z jmenovaných množin obsahuje řetězce, které ještě nebyly testovány – na základě těch se posléze vybírají pravidla do GLL tabulky. Do druhé z jmenovaných množin se vkládají prověřené řetězce neterminálů, což zabraňuje zacyklení algoritmu a snižuje časovou náročnost. Do Ω umístíme počáteční neterminál, neboť $S \Rightarrow^0 S$. Dokud jsou v Ω přítomny nějaké řetězce, vyjmeme některý z nich (w_k), přidáme ho do Θ a sestrojíme pro něj množinu Φ. Φ je množina uspořádaných trojic, kde w_D je větná forma derivovaná z w_k za užití některé komponenty (P_i) gramatického systému σ

v módu $f \in D$. Výraz $|\text{mode}(\sigma)|$ reprezentuje stupeň práce, respektive délku derivačního kroku gramatického systému. $\vec{C}$ je řídící neterminál derivace a $r = [r_1, r_2, \ldots, r_n]$ je seznam všech pravidel užitých při derivování, výraz $\text{count}(\sigma)$ reprezentuje celkový počet pravidel v systému σ. Pravidla v r jsou seřazena ve stejném pořadí, v jakém byla aplikována na w_k. Pokud je z větné formy možné odvodit větu, přidáme do GLL tabulky pravidla následujícím způsobem:

(i) pokud je některý terminál $y \in First(\vec{C})$ přítomen v množině $First$ nejlevějšího odvozeného symbolu, vytvoříme pravidlo $\alpha(y, \vec{C})$, viz řádek 9,

(ii) pokud lze z w_D odvodit prázdný řetězec nebo byl řídící neterminál vymazán, pak pro každý terminál $y \in Follow_f^{CD}(\vec{C})$ (libovolný terminál, který může následovat za $\vec{C}$) vytvoříme pravidlo $\alpha(y, \vec{C})$, viz řádek 14.

Posledním krokem je vytvoření řetězce $\overline{w_D}$, který získáme z w_D odstranění všech terminálů. Jestliže jsme vznilký řetězec doposud nezkoumali, přidáme ho do Ω, což znamená, že bude prozkoumán v dalším kroku.

Algoritmus 4.1.3 je neoptimalizovaný a potenciálně nekonečný. Konečnost je zajištěna pouze pro konvergující systémy – u nich se větné formy nemohou neomezeně rozrůstat a konečnost algoritmu je garantována konečností množiny neterminálů. Zpracování divergujících systémů takto jednoduché není, a proto si na něj vymezíme zvláštní kapitolu.

Uvedli jsme, že vstupem transformace musí být LL gramatický systém. Splnění této podmínky nemusíme požadovat u všech gramatických systémů zcela striktně. Transformovat je možné i systémy, které nejsou LL, ale splňují podmínku následující větu:

Věta 4.1.10. *Nechť σ je CD či HCD gramatický systém a*

$$\overline{P} = \{p \,|\, p \in P_i,\ 1 \leq i \leq n\}.$$

Pokud $G = (N, T, S, \overline{P})$ je LL gramatika, pak σ může být převedeno na GLL tabulku.

Věta 4.1.10 říká, že můžeme převést právě takové systémy, u kterých pravidla všech komponent sjednocených do jedné množiny tvoří bezkontextovou LL gramatiku. To také znamená, že můžeme na gramatický systém aplikovat stejné metody, které se využívají pro zisk LL gramatiky z BKG – například vytýkání nebo odstranění levé rekurze.

V GLL tabulce může být pro každý řídící neterminál a každý terminál jediná n-tice pravidel, ale během převodu systému, který není LL, může nastat situace, kdy máme k dispozici několik n-tic pravidel, jež všechny obsahují tentýž řídící neterminál. Některé z nich povedou k větě jazyka, jiné nikoliv (systém se později zasekne a nebude moci pokračovat). Uvedenou situaci si demostrujeme na příkladu.

Příklad 4.1.11. Uvažujme systém

$$\sigma_8 = \big\{\{X, Y\}, \{a, b\}, XX, (P_1, =2), (P_2, =2)\big\}$$

s komponentami $P_1 = \{1\colon X \to aYb,\ 2\colon X \to \varepsilon\}$ a $P_2 = \{1\colon Y \to aXb,\ 2\colon Y \to \varepsilon\}$, jež generuje jazyk $L = \{a^n b^n a^n b^n \,|\, n \geq 0\}$. Nechť $\vec{X}$ je řídící neterminál, pak je možné derivovat podle $\vec{X}$ za použití pravidel $[1,1]$, $[1,2]$, $[2,1]$ nebo $[2,2]$, z toho první dvě možnosti lze použít, pokud aktuální lexéma je a, druhé dvě pro b či \$. Použijeme-li pravidla $[1,2]$, dostaneme větnou formu, která obsahuje pouze jeden neterminál Y, tedy nelze na ni aplikovat žádné další pravidlo a je tedy syntakticky chybná. Z toho důvodu do GLL tabulky

může přijít pouze n-tice pravidel $[1, 1]$. Tento příklad ukazuje, že u systému, jež nejsou LL, je nutné provést dopředu analýzu každé odvozené větné formy a vyřadit syntakticky chybné konstrukce. Jestliže se najde více n-tic pravidel, po jejichž aplikaci zůstává syntakticky správná konstrukce, jedná se zcela jistě o chybu v návrhu jazyka, takový jazyk nelze převést na GLL tabulku.

4.1.1 Ukázka transformace homogenního gramatického systému

Nyní známe veškerou teorii potřebnou k převodu CD gramatického systému na GLL tabulku. Je vhodné si celý převod demonstrovat na několika příkladech.

Příklad 4.1.12. Uvažujme kontextový jazyk $L_\phi = \{0^n1^n0^n \mid n \geq 0\}$ a odpovídající gramatický systém

$$\begin{aligned}
\sigma_9 &= \big(\{S, A, B, C, D\}, \{0, 1\}, S, P_1, P_2, P_3, P_4\big),\\
P_1 &= \{S \to S,\ S \to ABA\},\\
P_2 &= \{A \to 0C,\ B \to 1D\},\\
P_3 &= \{C \to 0A,\ D \to 1B\},\\
P_4 &= \{A \to \varepsilon,\ B \to \varepsilon,\ C \to \varepsilon,\ D \to \varepsilon\}.
\end{aligned}$$

Bez dalších důkazů prohlásíme $L(\sigma_9) = L_\phi$. Prvním krokem je vytvoření množiny $First$ pro každý symbol z $N \cup T$. Postupujeme přesně podle algoritmu 4.1.1:

i.	$First(0) := 0,\ First(1) := 1$	
ii.	$\forall X \in \{S, A, B, C, D\}$	$: First(X) := \emptyset$
iii.	$S \to S$	$: First(S) \leftarrow First(S)$
iv.	$S \to ABA$	$: First(S) \leftarrow First(A)$
v.	$A \to 0C$	$: First(A) \leftarrow First(0) = \{0\}$
		$: First(S) \leftarrow First(0) = \{0\}$
vi.	$B \to 1D$	$: First(B) \leftarrow First(1) = \{1\}$
vii.	$C \to 0A$	$: First(C) \leftarrow First(0) = \{0\}$
viii.	$D \to 1B$	$: First(D) \leftarrow First(1) = \{1\}$
ix.	$A\|B\|C\|D \to \varepsilon$	: žádná množina se nemění

Binární operací $A \leftarrow B$ zkracujeme formálně přesnější zápis $A := A \cup B$. Nyní potřebujeme sestrojit množinu $Follow_f^{CD}$ pro každý neterminál, respektive $Follow_{=3}^{CD}$, protože budeme využívat pouze derivace délky 2. Tuto množinu sestrojíme podle algoritmu 4.1.2:

i.	$\Omega := \{S\}$, $\Theta := \emptyset$
ii.	$\forall A \in N$: $Follow^{CD}_{=3}(A) := \emptyset$
iii.	$Follow^{CD}_{=3}(S) := \$$
iv.i	$w_k = S$
iv.ii	$\Phi = S, ABA$
iv.iii.i	$w_D = S$
iv.iii.ii	$S \in N$: $Follow^{CD}_{=3}(S) \leftarrow Follow^{CD}_{=3}(S) = \{\$\}$
iv.iii.iii	$w_D = ABA$
iv.iii.iv	$A \in N$: $Follow^{CD}_{=3}(A) \leftarrow First(B) = \{1\}$
iv.iii.v	$B \in N$: $Follow^{CD}_{=3}(B) \leftarrow First(A) = \{0\}$
iv.iii.vi	$A \in N$: $Follow^{CD}_{=3}(A) \leftarrow Follow^{CD}_{=3}(S) = \{1, \$\}$
iv.iii.vii	$\Theta \leftarrow ABA$
iv.iv	$\Omega = \{ABA\}$, $\Theta = \emptyset$
iv.v	$w_k = ABA$
iv.vi	$\Phi = \{0C1D0C, \varepsilon\}$
iv.vii.i	$w_D = 0C1D0C$
iv.vii.ii	$C \in N$: $Follow^{CD}_{=3}(C) \leftarrow First(1) = \{1\}$
iv.vii.iii	$D \in N$: $Follow^{CD}_{=3}(D) \leftarrow First(0) = \{0\}$
iv.vii.iv	$C \in N$: $Follow^{CD}_{=3}(C) \leftarrow Follow^{CD}_{=3}(ABA) = Follow^{CD}_{=3}(A) = \{1, \$\}$
iv.vii.v	$\Theta \leftarrow 0C1D0C$
iv.vii.vi	$w_D = \varepsilon$
iv.viii	$\Omega = \{0C1D0C\}$, $\Theta = \emptyset$
iv.ix	$w_k = 0C1D0C$
iv.x	$\Phi = \{00A11B00A, 010, \varepsilon\}$
iv.xi.i	$w_D = 00A11B00A$
iv.xi.ii	$A \in N$: $Follow^{CD}_{=3}(A) \leftarrow First(1) = \{1, \$\}$
iv.xi.iii	$B \in N$: $Follow^{CD}_{=3}(B) \leftarrow First(0) = \{0, \$\}$
iv.xi.iv	$A \in N$: $Follow^{CD}_{=3}(A) \leftarrow Follow^{CD}_{=3}(00A11B00A) = \{1, \$\}$
iv.xi.v	$\Theta \leftarrow 00A11B00A$
iv.xi.vi	$w_D = 010$
iv.xi.vii	$w_D = \varepsilon$
iv.xii	$\Omega = \{00A11B00A\}$, $\Theta = \emptyset$
v.	žádná množina$Follow^{CD}_{=3}$nezměněna $\Rightarrow$ konec

Výstupem z těchto algoritmů jsou množiny v následující podobě:

$$\begin{array}{lcllcl} First(S) & = & \{0\}, & Follow^{CD}_{=3}(S) & = & \{\$\}, \\ First(A) & = & \{0\}, & Follow^{CD}_{=3}(A) & = & \{1, \$\}, \\ First(B) & = & \{1\}, & Follow^{CD}_{=3}(B) & = & \{0\}, \\ First(C) & = & \{0\}, & Follow^{CD}_{=3}(C) & = & \{1, \$\}, \\ First(D) & = & \{1\}, & Follow^{CD}_{=3}(D) & = & \{0\}. \end{array}$$

Pro přehlednost si seřadíme pravidla nezávisle na jednotlivých komponentách

$$P = \left\{ \begin{array}{llll} 1: & S \to S, & 6: & D \to 1B, \\ 2: & S \to ABA, & 7: & A \to \varepsilon, \\ 3: & A \to 0C, & 8: & B \to \varepsilon, \\ 4: & B \to 1D, & 9: & C \to \varepsilon, \\ 5: & C \to 0A, & 10: & D \to \varepsilon \end{array} \right\}$$

a aplikujeme transformační algoritmus 4.1.3.

i.	$\Omega := \{S\}, \Theta := \emptyset$
ii.	$w_k = S, \Omega = \emptyset, \Theta = \{S\}$
iii.i	$S \Rightarrow_{P_1}^{=3} ABA[\vec{S} \,\vert\, 1,1,2]$
iii.ii	$\Phi = \left\{ \left(ABA, \vec{S}, [1,1,2] \right) \right\}$
iii.iii	$0 \in First(\vec{S}) \wedge 0 \in First(ABA)$: $\alpha(0, \vec{S}) := [1,1,2]$
iii.iv	$Empty(ABA) = \{\varepsilon\} \wedge \$ \in Follow_{=3}^{CD}(\vec{S})$: $\alpha(\$, \vec{S}) := [1,1,2]$
iii.v	$\overline{w_D} = ABA, \Omega = \{ABA\}$
iv.	$w_k = ABA, \Omega = \emptyset, \Theta = \{S, ABA\}$
v.i	$ABA \Rightarrow_{P_2}^{=3} 0C1D0C[\vec{A} \,\vert\, 3,4,3]$
v.ii	$ABA \Rightarrow_{P_4}^{=3} \varepsilon[\vec{A} \,\vert\, 7,8,7]$
v.iii	$\Phi = \left\{ \left(0C1D0C, \vec{A}, [3,4,3] \right), \left(\varepsilon, \vec{A}, [7,8,7] \right) \right\}$
v.iv	$0 \in First(\vec{A}) \wedge 0 \in First(0C1D0C)$: $\alpha(0, \vec{A}) := [3,4,3]$
v.v	$Empty(0C1D0C) = \emptyset$
v.vi	$\overline{w_D} = CDC, \Omega = \{CDC\}$
v.vii	$0 \in First(\vec{A}) \wedge \neg(0 \in First(\varepsilon))$
v.viii	$Empty(\varepsilon) = \{\varepsilon\} \wedge 1 \in Follow_{=3}^{CD}(\vec{A})$: $\alpha(1, \vec{A}) := [3,4,3]$
v.ix	$Empty(\varepsilon) = \{\varepsilon\} \wedge \$ \in Follow_{=3}^{CD}(\vec{A})$: $\alpha(\$, \vec{A}) := [7,8,7]$
vi.	$w_k = CDC, \Omega = \emptyset, \Theta = \{S, ABA, CDC\}$
vii.i	$CDC \Rightarrow_{P_3}^{=3} 0A1B0A[\vec{C} \,\vert\, 5,6,5]$
vii.ii	$CDC \Rightarrow_{P_4}^{=3} \varepsilon[\vec{A} \,\vert\, 9,10,9]$
vii.iii	$\Phi = \left\{ \left(0A1B0A, \vec{C}, [5,6,5] \right), \left(\varepsilon, \vec{C}, [9,10,9] \right) \right\}$
vii.iv	$0 \in First(\vec{C}) \wedge 0 \in First(0A1B0A)$: $\alpha(0, \vec{C}) := [5,6,5]$
vii.v	$Empty(0A1B0A) = \emptyset$
vii.vi	$\overline{w_D} = ABA, \Omega = \emptyset$
vii.vii	$0 \in First(\vec{C}) \wedge \neg(0 \in First(\varepsilon))$
vii.viii	$Empty(\varepsilon) = \{\varepsilon\} \wedge 1 \in Follow_{=3}^{CD}(\vec{C})$: $\alpha(1, \vec{C}) := [5,6,5]$
vii.ix	$Empty(\varepsilon) = \{\varepsilon\} \wedge \$ \in Follow_{=3}^{CD}(\vec{C})$: $\alpha(\$, \vec{C}) := [9,10,9]$
viii.	$\Omega = \emptyset$: konec

Výsledek převodu přehledně zobrazuje tabulka 4.1.2. Další příklad transformace, tentokrát kontextového jazyka $L = \{a^n b^m a^n b^m \mid n \geq 1\}$, je k nalezení v příloze A.

N \ T	**0**	**1**	$
S	1,1,2		1,1,2
A	3,4,3	7,8,7	3,4,3
B			
C	5,6,5	9,10,9	5,6,5
D			

Tabulka 4.1.2 GLL tabulka pro jazyk $L_\phi = \{0^n 1^n 0^n \mid n \geq 0\}$.

4.2 Zpracování HCD gramatických systémů v módu $=k$

Zobecníme-li získané poznatky o homogenních gramatických systémech pracujících v módu $=k$, zjistíme, že lze podobným způsobem lze zpracovat i hybridní gramatické systémy, jejich všechny komponenty pracují v módu $=k$. V prvním kroku je potřeba upravit množinu $Follow_f^{CD}$, $f \in D$ tak, aby vyhovovala způsobu práce HCD gramatických systémů – zaveďme si tedy množinu $Follow^{HCD}$ formálně.

Definice 4.2.1. Nechť $\sigma = (N, T, P, S, (P_1, f_1), \ldots, (P_n, f_n))$ je HCD gramatický systém, kde $f_i \in {=k} \mid k \geq 1$, $1 \leq i \leq n$. Pro každý neterminál $A \in N$ je definována množina $Follow^{HCD}(A)$ následovně:

$$Follow^{HCD}(A) = \left\{ a \mid a \in T,\ S \Rightarrow_{P_{i_1}}^{f_{i_1}} w_1 \Rightarrow_{P_{i_2}}^{f_{i_2}} \ldots \Rightarrow_{P_{i_m}}^{f_{i_m}} w_m = xAay,\ x, y \in (N \cup T)^* \right\}$$
$$\cup \left\{ \$ \mid S \Rightarrow_{P_{i_1}}^{f_{i_1}} w_1 \Rightarrow_{P_{i_2}}^{f_{i_2}} \ldots \Rightarrow_{P_{i_m}}^{f_{i_m}} w_m = xA,\ x \in (N \cup T)^* \right\}$$

pro libovolné $m \geq 1$, $1 \leq i_j \leq n$, $1 \leq j \leq m$.

Jedinou zásadní změnou oproti množině $Follow_f^{CD}$ je to, že každá komponenta při odvozování pracuje ve svém vlastním módu, který je nezávislý na ostatních komponentách. Definice GLL tabulky 4.1.7 můžeme ponechat v nezměněné formě, neboť zde požadujeme pouze jednu hodnotu mapovací funkce α pro libovolné $a \in T$ a $A \in N$, což znamená, že v buňkách GLL tabulky může být libovolný počet pravidel. Můžeme tedy přistoupit k úpravě transformačních algoritmů.

1. Upravíme algoritmus 4.1.2 následovně:
   (i) Zaměníme všechny výskyty $Follow_f^{CD}$ za $Follow^{HCD}$,
   (ii) zavedeme si množinu Φ jako $\Phi = \left\{ w_D \mid w_k \Rightarrow_{P_i}^{f_i} w_D,\ f_i \in \{=k \mid k \geq 1\},\ 1 \leq i \leq n \right\}$.
2. Upravíme algoritmus 4.1.3 následovně:
   (i) Zaměníme všechny výskyty $Follow_f^{CD}$ za $Follow^{HCD}$,

(ii) zaměníme $|\text{mode}(\sigma)|$ za $|\text{mode}(P_i)|$ v definici množiny Φ, kde výraz $|\text{mode}(P_i)|$ reprezentuje délku derivačního kroku, respektive stupeň práce, komponenty P_i.

Po provedení těchto změn můžeme provést transformaci stejným způsobem. Implementace daných algoritmů se nijak neztížila, dokonce je řešení zobecněné pro HCD systémy implementačně méně náročné.

4.2.1 Ukázka transformace HCD gramatického systému v módu $=k$

Uvažujme kontextový jazyk $L_\mu = \{huh^r u\text{mi}(h) \mid h \in \{a,b,c\}^*, h^r = \text{reverse}(h)\}$, kde mi je morfismus definovaný jako:

$$\text{mi}\,(x \in \{a,b,c\}) = \begin{cases} c & \text{pro } x = a, \\ b & \text{pro } x = c, \\ a & \text{pro } x = b. \end{cases}$$

Tento jazyk je generován gramatickým systémem

$$\begin{aligned}
\sigma_{10} &= (\{S, H, \mathcal{H}, M, \mathcal{M}\},\ \{a,b,c,u\},\ S,\ (P_1,=1),(P_2,=2),(P_3,=2),\\
&\quad (P_4,=2),(P_5,=2),(P_6,=2))\,,\\
P_1 &= \{S \to HuM\}\,,\\
P_2 &= \{H \to u,\ M \to \varepsilon\}\,,\\
P_3 &= \{H \to a\mathcal{H}a,\ M \to c\mathcal{M}\}\,,\\
P_4 &= \{H \to b\mathcal{H}b,\ M \to a\mathcal{M}\}\,,\\
P_5 &= \{H \to c\mathcal{H}c,\ M \to b\mathcal{M}\}\,,\\
P_6 &= \{\mathcal{H} \to H,\ \mathcal{M} \to M\}\,.
\end{aligned}$$

Po seřazení pravidel a vytvoření množin a aplikaci algoritmů na vytvoření množin $First$ a $Follow^{HCD}$ dostáváme:

$$P = \left\{ \begin{array}{llll}
1: & S \to HuM, & 7: & M \to a\mathcal{M}, \\
2: & H \to u, & 8: & H \to c\mathcal{H}c, \\
3: & M \to \varepsilon, & 9: & M \to b\mathcal{M}, \\
4: & H \to a\mathcal{H}a, & 10: & \mathcal{H} \to H, \\
5: & M \to c\mathcal{M}, & 11: & \mathcal{M} \to M, \\
6: & H \to b\mathcal{H}b, & &
\end{array} \right\}$$

$$\begin{array}{llllll}
First(S) & = & \{a,b,c,u\}\,, & Follow^{HCD}(S) & = & \{\$\}\,, \\
First(H) & = & \{a,b,c,u\}\,, & Follow^{HCD}(H) & = & \{a,b,c,u\}\,, \\
First(\mathcal{H}) & = & \{a,b,c\}\,, & Follow^{HCD}(\mathcal{H}) & = & \{0\}\,, \\
First(M) & = & \{a,b,c\}\,, & Follow^{HCD}(M) & = & \{\$\}\,, \\
First(\mathcal{M}) & = & \{a,b,c\}\,, & Follow^{HCD}(\mathcal{M}) & = & \{\$\}\,.
\end{array}$$

Využijeme transformační algoritmus:

$\Omega := \{S\}$, $\Theta := \emptyset$
$S \Rightarrow^{=1}_{P_1} HuM[\overrightarrow{S} \mid 1]$:
$a \in First(\overrightarrow{S}) \wedge a \in First(HuM) : \alpha(a, \overrightarrow{S}) := [1]$
$b \in First(\overrightarrow{S}) \wedge b \in First(HuM) : \alpha(b, \overrightarrow{S}) := [1]$
$c \in First(\overrightarrow{S}) \wedge c \in First(HuM) : \alpha(c, \overrightarrow{S}) := [1]$
$d \in First(\overrightarrow{S}) \wedge d \in First(HuM) : \alpha(d, \overrightarrow{S}) := [1]$
$Empty(HuM) = \emptyset$
$\Omega = \{HM\}$, $\Theta = \{S\}$
$HM \Rightarrow^{=2}_{P_2} u[\overrightarrow{H} \mid 2, 3]$:
$u \in First(\overrightarrow{H}) \wedge u \in First(u) : \alpha(u, \overrightarrow{H}) := [2, 3]$
$HM \Rightarrow^{=2}_{P_3} a\mathcal{H}ac\mathcal{M}[\overrightarrow{H} \mid 4, 5]$:
$a \in First(\overrightarrow{H}) \wedge a \in First(a\mathcal{H}ac\mathcal{M}) : \alpha(a, \overrightarrow{H}) := [4, 5]$
$HM \Rightarrow^{=2}_{P_4} b\mathcal{H}ba\mathcal{M}[\overrightarrow{H} \mid 6, 7]$:
$b \in First(\overrightarrow{H}) \wedge b \in First(b\mathcal{H}ba\mathcal{M}) : \alpha(b, \overrightarrow{H}) := [6, 7]$
$HM \Rightarrow^{=2}_{P_5} c\mathcal{H}cb\mathcal{M}[\overrightarrow{H} \mid 4, 5]$:
$c \in First(\overrightarrow{H}) \wedge c \in First(c\mathcal{H}cb\mathcal{M}) : \alpha(c, \overrightarrow{H}) := [8, 9]$
$\Omega = \{\mathcal{HM}\}$, $\Theta = \{S, HM\}$
$\mathcal{HM} \Rightarrow^{=2}_{P_5} HM[\overrightarrow{\mathcal{H}} \mid 10, 11]$:
$a \in First(\overrightarrow{\mathcal{H}}) \wedge a \in First(HM) : \alpha(a, \overrightarrow{\mathcal{H}}) := [10, 11]$
$b \in First(\overrightarrow{\mathcal{H}}) \wedge b \in First(HM) : \alpha(b, \overrightarrow{\mathcal{H}}) := [10, 11]$
$c \in First(\overrightarrow{\mathcal{H}}) \wedge c \in First(HM) : \alpha(c, \overrightarrow{\mathcal{H}}) := [10, 11]$
$u \in First(\overrightarrow{\mathcal{H}}) \wedge u \in First(HM) : \alpha(u, \overrightarrow{\mathcal{H}}) := [10, 11]$
$Empty(HM) = \emptyset$
$\Omega = \emptyset$, $\Theta = \{S, HM, \mathcal{HM}\}$

Dostáváme následující GLL tabulku:

N ╲ T	**a**	**b**	**c**	**u**	**$**
S	1	1	1	1	
H	4,5	6,7	8,9	2,3	
$\mathcal{H}$					
M	10,11	10,11	10,11	10,11	
$\mathcal{M}$					

Tabulka 4.2.1 GLL tabulka pro jazyk $L_\mu = \{huh^r u\text{mi}(h) \mid h \in \{a, b, c\}^*\}$.

4.3 Zpracování HCD gramatických systémů s ukončujícími komponentami

Často se nám při návrhu gramatického systému hodí, mít k dispozici nějakou ukončující komponentu. Jedná se o výraznou úsporu z hlediska složitosti navrhovaného systému. Ukončující komponenty jsou značně nedeterministické. Uvedeme 2 způsoby, jak s nimi při syntaktické analýze zacházet – rozklad a plánování derivací. První z jmenovaných způsobů je jednoduchý, ale platný pouze pro omezené případy užití, a druhý univerzálnější, ale podstatně složitější.

V prvním uvažovaném způsobu komponenty rozložíme na jednotlivá pravidla a budeme plnit tabulku těmito pravidly stejně, jako by se jednalo o obyčejnou LL gramatiku. Jinak řečeno, rozložíme komponentu s módem t na n komponent pracujících v módu $=1$, kde n značí počet pravidel původní komponenty. Na takto upravený systém lze aplikovat algoritmus převodu ze 4.2. Z praktického hlediska platí, že aplikujeme v jednom kroku pouze jedno pravidlo na základě příchozí lexémy a horního neterminálu na zásobníku. Pozorný čtenář jistě může namítnout, že tento způsob odvozování v obecné rovině nesplňuje definici ukončující derivace, a bude mít pravdu. V nedeterministickém systému se sekvence derivačních kroků o délce 1 opravdu chová jinak, což demonstrujeme na příkladu 4.3.1.

Příklad 4.3.1. Uvažujme HCD gramatický systém:

$$\begin{aligned}
\sigma_{11} &= (\{S, A, B, C\}, \{a, b, c, m\}, S, (P_1, =1), (P_2, =2), (P_3, t)), \\
P_1 &= \{S \to ABC,\ B \to m\}, \\
P_2 &= \{B \to b,\ C \to c\}, \\
P_3 &= \{A \to a,\ C \to \varepsilon\}.
\end{aligned}$$

Systém provede posloupnost derivací následovně:

$$S \Rightarrow^{=1}_{P_1} ABC \Rightarrow^{t}_{P_3} aB \Rightarrow^{=1}_{P_1} am.$$

Odvozování probíhá zcela deterministicky, vždy máme k dispozici pouze jedno pravidlo, které lze využít. Jiná sekvence derivací ani není přípustná, takže sytém generuje triviální jazyk $L_\varphi = \{am\}$. Komponenta P_2 nemůže být nikdy využita. Teď uvažujme sekvence derivačních kroků řešených metodou rozkladu:

$$\begin{aligned}
&S \Rightarrow^{=1}_{P_1} ABC \Rightarrow^{t}_{P_3} aBC \Rightarrow^{=2}_{P_2} abc, \\
&S \Rightarrow^{=1}_{P_1} ABC \Rightarrow^{t}_{P_3} aBC \Rightarrow^{=2}_{P_2} amC \Rightarrow^{t}_{P_3} amc.
\end{aligned}$$

Nejen, že jsme získali dvě možnosti, jak provést odvození věty, ale navíc obě tyto možnosti vedou k syntakticky chybným řetězcům.

V jakých situacích lze tedy nasadit metodu rozkladu? Zcela bezpečné je to u systémů, kde množina neterminálů na levé straně pravidel ukončující komponenty P_t a množina neterminálů na levé straně pravidel libovolné komponenty P_i pracující v módu $=k$, $k \geq 2$, jsou disjunktní, formálně řečeno:

$$A \to u \in P_t \Longrightarrow A \to v \notin P_i,\ A \in N,\ u, v \in (N \cup T)^*,\ 1 \leq i \leq n.$$

Zatímco metodu rozkladu si později ukážeme podrobně na jazyku $\{a^n b^n c^n | n \geq 1\}$, metodu plánování derivací si nyní naznačíme spíše z teoretického hlediska. S nasazením ukončující derivace není problém z pohledu transformace, ale spíše z pohledu přijímání řetězců, neboť zde máme k dispozici pouze nejlevější vstupní lexému. Přepisovat odpovídající neterminály (ležící na levé straně pravidel ukončující komponenty) nemůžeme, neboť nejsme schopni vybrat vhodné pravidlo. Jejich zpracování jsme nuceni odložit na pozdější okamžik, ale zároveň musíme zajistit, aby nám je nepřepsala některá jiná komponenta.

Prvním krokem je sestrojení množiny neterminálů vyskytujících se na levé straně některého pravidla ukončující komponenty, tuto množinu si pro komponentu P_i označíme N^{t_i}. Poté rozšíříme abecedu neterminálů následovně:

$$\overline{N} = N \cup \{A^{t_i} \mid A \in N^{t_i}\},$$

kde $\overline{N}$ je naše nová množina neterminálů. Tento postup opakujeme pro všechny ukončující komponenty. Pro názornost uvádíme ukázku na systému z příkladu 4.3.1:

$$\begin{aligned} N^{t_3} &= \{A, C\}, \\ \overline{N} &= \{S, A, B, C, A^{t_3}, C^{t_3}\}. \end{aligned}$$

Dále upravíme všechna pravidla z P_i – obsahuje-li pravidlo na pravé straně neterminál A z abecedy N^{t_i}, zaměníme tento neterminál za neterminál A^{t_i}. Pokud by tedy ukončující komponenta P_3 obsahovala pravidlo $A \to BA$, pak jej nahradíme pravidlem $A \to BA^{t_3}$. Tím jsme umožnili rekurzivní odvozování pomocí ukončující komponenty. Nově zavedeme zcela klíčové omezení – z neterminálů z abecedy N^{t_i} může odvozovat pouze ukončující komponenta P_i. Z jejího pohledu jsou neterminály A a A^{t_i} totožné, tj. A^{t_i} může být dosazeno na levou stranu pravidla místo A. Tím jsme zabránili, aby nám některá komponenta pracující v módu $=k$, $k \geq 2$, přepisovala neterminály, ze kterých se má odvozovat pomocí ukončující derivace, avšak aktuálně nejsou na nejlevější pozici v odvozované větné formě. Poslední věc, kterou musíme zajistit, je záměna všech neterminálů z N^{t_i} za odpovídající neterminál A^{t_i} v celé větné formě. Tato záměna se provádí po provedení prvního kroku ukončující derivace. S ukázkou odvozování se opět se vracíme k příkladu 4.3.1.

$$S \Rightarrow_{P_1}^{=1} ABC \Rightarrow_{P_3}^{t} aBC^{t_3} \Rightarrow_{P_1}^{=1} amC^{t_3} \Rightarrow_{P_3}^{t} am.$$

Po těchto úpravách může být nasazen transformační algoritmus, výsledná GLL tabulka pro σ_{11} je v podobně 4.3.1. Pravidla pro neterminál B a terminály b, c by mohla být vynechána, neboť tato situace nemůže nikdy nastat.

N \ T	**a**	**b**	**c**	**m**	**$**
S	1				
A	5				
B		3,4		2	
C					
A^{t_3}					
C^{t_3}					6

Tabulka 4.3.1 GLL tabulka pro jazyk $L_\varphi = \{am\}$ s využitím metody plánování derivací.

4.3.1 Ukázka transformace HCD gramatického systému s ukončujícími komponentami

Praktickým příkladem demonstrujícím využití gramatických systémů je systém pro popis deklarace a inicializace proměnných různých typů. Systém bude přijímat konstrukce tvaru

$$typ_1\ typ_2 \dots typ_n : id_1\ id_2 \dots id_n : hodnota_1\ hodnota_2 \dots hodnota_n,$$

tedy takové deklarace proměnných, které obsahují stejný počet typů, identifikátorů a hodnot, přesněji řečeno $L_\kappa = \{typ^n : id^n : hodnota^n \mid n \geq 1, typ \in \{int, double, bool\}, hodnota \in \{v^{int}, v^{double}, v^{bool}\}\}$, což velmi nápadně připomíná známý kontextový jazyk $\{a^n b^n c^n \mid n \geq 1\}$. Například konstrukce „$int\ double\ bool : a\ b\ c : 5\ 6.1\ true$“ je syntakticky správná, zatímco konstrukce „$int\ bool : a\ b\ c : 5\ true$“ nebude přijata. Pro popis tohoto jazyka využijeme HCD gramatický systém:

$$\begin{aligned}
\sigma_{12} &= (N, T, TXLYV, (P_1, =2), (P_2, =2), (P_3, t)), \\
N &= \{X, \mathcal{X}, Y, \mathcal{Y}, T, L, V\} \\
T &= \left\{int, double, bool, v^{int}, v^{double}, v^{bool}, :\right\}, \\
P_1 &= \{X \to T\mathcal{X}L \mid :,\ Y \to \mathcal{Y}V \mid :\}, \\
P_2 &= \{\mathcal{X} \to TXL \mid :,\ \mathcal{Y} \to YV \mid :\}, \\
P_3 &= \left\{T \to int \mid double \mid bool,\ L \to id,\ V \to v^{int} \mid v^{double} \mid v^{bool}\right\}.
\end{aligned}$$

Tento systém vskutku přijímá jen věty obsahující stejný počet typů, identifikátorů a inicializačních hodnot, ale pořád je silně nedeterministický. S ukončující derivací se vypořádáme pomocí metody rozkladu. Sestrojíme množiny $First$ a $Follow^{HCD}$:

$$\begin{aligned}
First(T) &= \{int, double, bool\}, & Follow^{HCD}(T) &= \{:, int, double, bool\}, \\
First(L) &= \{id\}, & Follow^{HCD}(L) &= \{id, :\}, \\
First(V) &= \left\{v^{int}, v^{double}, v^{bool}\right\}, & Follow^{HCD}(V) &= \{\$\}, \\
First(X) &= \{:, int, double, bool\}, & Follow^{HCD}(X) &= \{id\}, \\
First(\mathcal{X}) &= \{:, int, double, bool\}, & Follow^{HCD}(\mathcal{X}) &= \{id\}, \\
First(Y) &= \{:\}, & Follow^{HCD}(Y) &= \left\{v^{int}, v^{double}, v^{bool}\right\}, \\
First(\mathcal{Y}) &= \{:\}, & Follow^{HCD}(\mathcal{Y}) &= \left\{v^{int}, v^{double}, v^{bool}\right\}.
\end{aligned}$$

Při transformaci na GLL tabulku narazíme na problém – system σ_{12} je divergující. Této problematice se budeme věnovat později do hloubky, nyní bez dalšího vysvětlování využijeme jednoduchý trik. Neterminály, které narušují konvergenci tohoto systému, tedy T, L, V, povolíme během transformace pouze jednou ve větné formě a aplikujeme transformační algoritmus. Posléze dostáváme tabulku 4.3.2, vytvořenou s následujícími pravidly:

1: $X \to T\mathcal{X}L$
2: $X \to :$
3: $Y \to \mathcal{Y}V$
4: $Y \to :$
5: $\mathcal{X} \to TXL$
6: $\mathcal{X} \to :$
7: $\mathcal{Y} \to YV$
8: $\mathcal{Y} \to :$
9: $T \to int$
10: $T \to double$
11: $T \to bool$
12: $L \to id$
13: $V \to v^{int}$
14: $V \to v^{double}$
15: $V \to v^{bool}$

N ╲ T	**int**	**double**	**bool**	**id**	v^{int}	v^{double}	v^{bool}	**:**	**$**
X	1,3	1,3	1,3					2,4	
$\mathcal{X}$	5,7	5,7	5,7					6,8	
Y									
$\mathcal{Y}$									
L	9	10	11						
S				12					
S					13	14	15		

Tabulka 4.3.2 GLL tabulka pro jazyk $L_\kappa = \{typ^n : id^n : hodnota^n \mid n \geq 1\}$.

4.4 Transformace divergujícího HCD gramatického systému na GLL tabulku

V předchozí sekci jsme již naznačili, že transformaci lze aplikovat pouze na systémy, které si zachovávají stejný nebo nižší počet shodných neterminálů, tj. že nelze z A odvodit AA, B^mA či AB^m, $A, B \in N$, $m \in \mathbb{N}$. Tento problém je možné překlenout, pokud si zavedeme jistá omezení týkající se maximálního počtu neterminálů ve větné formě. Tato omezení uplatníme v každé iteraci algoritmu 4.1.3 na řetězec $\overline{w_D}$ v místě jeho vzniku. Tím znovu zajistíme konečnost algoritmu, protože množina neterminálů je konečná a jejich počet ve větné formě je omezen na nějaké $\zeta \in \mathbb{N}$. Na větnou formu $\overline{w_D}$ můžeme aplikovat omezení v podobě:

(i) odstraníme z $\overline{w_D}$ všechny výskyty řídícího neterminálu,

(ii) odstraníme z $\overline{w_D}$ výskyty všech neterminálů vyskytujících se na levé straně některého z užitých pravidel,

(iii) odstraníme z $\overline{w_D}$ všechny výskyty neterminálů přesahující předem stanovené ζ,

(iv) nechť každý neterminál se smí ve větné formě vyskytovat pouze tolikrát, kolikrát se vyskytuje na levé straně v užitých pravidlech – všechny nadbytečné výskyty z $\overline{w_D}$ odstraníme,

(v) z $\overline{w_D}$ odstraníme všechny výskyty přesahující ζ dané nejdelší možnou derivací podle některé z komponent. U CD systému je ζ rovno stupni systému, u HCD některou z komponent. Například pro HCD gramatický systém s komponentami pracujícími v módech =4, =3, =2 je ζ rovno 4. Pro komponenty pracující v módu t uvažujeme $\zeta = 1$.

Stanovíme-li si koeficient ζ předem, je nutné počítat s tím, že ζ určuje nejen rychlost konvergence, ale zároveň i přesnost algoritmu – při špatně zvoleném je možné, že některé větné formy vedoucí k platným větám mohou být vynechány. Je nutné doplnit, že ani jeden z uvedených způsobů nelze prohlásit za nejvhodnější pro všechny případy. Například (v) zajistí konečnost libovolného divergujícího systému a zároveň garantuje zachování celé množiny vět generovaných systémem, ale způsob (iv) je dostačující pro naprostou většinu námi diskutovaných případů a zajistí konvergenci mnohem rychleji. Je úkolem návrháře jazyka, aby vyhodnotil jakým způsobem nejlépe převést divergující GS na GLL tabulku.

4.4.1 Ukázka transformace divergujícího gramatického systému

Nyní si uvedeme příklad, jak by bylo možné řešit problém divergence u jazyka $L_\iota = \{a^{2n}b^{2n}c^{2n} \mid n \geq 1\}$, respektive u systému:

$$\sigma_{13} = (\{S, A, B, C, \mathcal{A}, \mathcal{B}, \mathcal{C}, \mathfrak{A}, \mathfrak{B}, \mathfrak{C}\}, \{a, b, c\}, S, (P_1, =1), (P_2, =3), (P_3, =3), (P_4, t)),$$
$$P_1 = \{1 : S \to \mathcal{ABC}\},$$
$$P_2 = \{2 : \mathcal{A} \to AA\mathfrak{A},\ 3 : \mathcal{B} \to BB\mathfrak{B},\ 4 : \mathcal{C} \to CC\mathfrak{C}\},$$
$$P_3 = \{5 : \mathfrak{A} \to AA\mathcal{A},\ 6 : \mathfrak{B} \to BB\mathcal{B},\ 7 : \mathfrak{C} \to CC\mathcal{C}\}$$
$$P_4 = \left\{ \begin{array}{l} 8 : A \to a,\ 9 : B \to b,\ 10 : C \to c,\ 11 : \mathcal{A} \to \varepsilon,\ 12 : \mathcal{B} \to \varepsilon, \\ 13 : \mathcal{C} \to \varepsilon,\ 14 : \mathfrak{A} \to \varepsilon,\ 15 : \mathfrak{B} \to \varepsilon,\ 16 : \mathfrak{C} \to \varepsilon, \end{array} \right\}$$

$$\begin{array}{lllllll}
First(S) & = & \{a\}, & Follow(S) & = & \{\$\}, \\
First(A) & = & \{a\}, & Follow(A) & = & \{a, b\}, \\
First(\mathcal{A}) & = & \{a\}, & Follow(\mathcal{A}) & = & \{b\}, \\
First(\mathfrak{A}) & = & \{a\}, & Follow(\mathfrak{A}) & = & \{b\}, \\
First(B) & = & \{b\}, & Follow(B) & = & \{b, c\}, \\
First(\mathcal{B}) & = & \{b\}, & Follow(\mathcal{B}) & = & \{c\}, \\
First(\mathfrak{B}) & = & \{b\}, & Follow(\mathfrak{B}) & = & \{c\}, \\
First(C) & = & \{c\}, & Follow(C) & = & \{c, \$\}, \\
First(\mathcal{C}) & = & \{c\}, & Follow(\mathcal{C}) & = & \{\$\}, \\
First(\mathfrak{C}) & = & \{c\}, & Follow(\mathfrak{C}) & = & \{\$\}.
\end{array}$$

Je zřejmé, že při aplikaci transformačního algoritmu bude neprozkoumaných větných forem stále přibývat. Pro vyřešení tohoto problému zvolíme postup číslo (iv) navrhovaný v sekci 4.1, tedy zavedeme omezení – každý neterminál se smí v řetězci vyskytovat pouze tolikrát, kolikrát se vyskytuje na levé straně v užitých pravidlech. Všechny nadbytečné výskyty odstraníme, což povede k maximálně jednomu výskytu A, B a C ve větné formě.

$$\Omega := \{S\},\ \Theta := \emptyset$$
$$S \Rightarrow^{=1}_{P_1} \mathcal{ABC}[\vec{S} \mid 1] :$$
$$\qquad a \in First(\vec{S}) \wedge a \in First(\mathcal{ABC}) : \alpha(a, \vec{S}) := [1]$$
$$\Omega := \{\mathcal{ABC}\},\ \Theta := \{S\}$$
$$\mathcal{ABC} \Rightarrow^{=3}_{P_2} AA\mathfrak{A}BB\mathfrak{B}CC\mathfrak{C}[\vec{\mathcal{A}} \mid 2, 3, 4] :$$
$$\qquad a \in First(\vec{\mathcal{A}}) \wedge a \in First(AA\mathfrak{A}BB\mathfrak{B}CC\mathfrak{C}) : \alpha(a, \vec{\mathcal{A}}) := [2, 3, 4]$$
$$\Omega := \{A\mathfrak{A}B\mathfrak{B}C\mathfrak{C}\},\ \Theta := \{S, \mathcal{ABC}\}$$
$$A\mathfrak{A}B\mathfrak{B}C\mathfrak{C} \Rightarrow^{=1}_{P_4} a\mathfrak{A}B\mathfrak{B}C\mathfrak{C}[\vec{A} \mid 8] :$$
$$\qquad a \in First(\vec{A}) \wedge a \in First(a\mathfrak{A}B\mathfrak{B}C\mathfrak{C}) : \alpha(a, \vec{A}) := [8]$$
$$\vdots$$

V uvedené ukázce je podstatné to, že před vložením větné formy $AA\mathfrak{A}BB\mathfrak{B}CC\mathfrak{C}$ do množiny Ω, ji zkrátíme na $A\mathfrak{A}B\mathfrak{B}C\mathfrak{C}$. Po dokončení transformace dostaneme tabulku:

N ╲ T	a	b	c	$
S	1			1
A	8			
$\mathcal{A}$	2,3,4	11		
$\mathfrak{A}$	5,6,7	14		
B		9		
$\mathcal{B}$			12	
$\mathfrak{B}$			15	
C			10	
$\mathcal{C}$				13
$\mathfrak{C}$				16

Tabulka 4.4.1 GLL tabulka pro jazyk $L_\iota = \{a^{2n}b^{2n}c^{2n} \mid n \geq 1\}$.

4.5 Syntaktický analyzátor založený na GLL tabulce

V sekci 4.1 při zavedení GLL tabulky jsme již naznačili, jak bude probíhat syntaktická analýza. Nyní se podrobně podíváme na algoritmus syntaktické analýzy založené na GLL, který nám poskytne pro zadaný systém a libovolný vstupní řetězec odpověď na otázku, jestli řetězec spadá do jazyka generovaného daným systémem. Stejně jako u klasické LL analýzy budeme využívat rozšířený deterministický zásobníkový automat jako matematický model pro zpracování řetězců. Na základě nejlevější lexémy vstupní větné formy a vrchního neterminálů na zásobníku budeme z GLL tabulky vybírat pravidla. Ty aplikujeme na větnou formu na zásobníku a tím simulujeme práci gramatického systému a zároveň konstrukci abstraktního syntaktického stromu. Předpokládejme, že již máme sestrojený lexikální analyzátor, jenž nám rozdělí vstupní větu na jednotlivé lexémy. Nejprve uvedeme GLL analyzátor po formální stránce – popsáný algoritmem 4.5.4.

Gramatické systémy se vždy začínaly odvozovat z počátečního neterminálu a ani zde tomu není jinak, proto v prvním kroku vložíme axiom na zásobník. Po zisku první lexémy vstupní věty a horního neterminálu na zásobníku si vyhledáme v GLL tabulce seznam pravidel, která použít. Je zřejmé, že není-li nalezeno žádné pravidlo, nemáme k dispozici žádný způsob jak pokračovat a musíme práci na větné formě ukončit s chybou. Tato část se nijak zásadně neliší od klasické LL syntaktické analýzy, nyní nastává klíčová fáze – expanze, kde už jsou rozdíly výrazné. Každé z nalezených pravidel aplikujeme na větnou formu. Nelze-li některé pravidlo aplikovat z důvodu chybějícího neterminálu na zásobníku, opět nelze pokračovat a je nutné skončit s chybou. Na rozdíl od LL analýzy tu provádíme několik expanzí v jednom kroku čímž simulujeme derivace délky k. Další fází je porovnávání – dokud se shoduje vrchol zásobníku se vstupní lexémou, vytlačujeme symboly ze zásobníku a čteme další části vstupní věty. Probublá-li nám znak $ na vrchol zásobníku, pak jsme z odvozováním hotovi a lze vynést závěr. Je-li přečtena celá věta, pak ji zcela jistě můžeme přijmout, v opačném případě ji zamítneme. Protože jsme ze zásobníku vytlačili všechny symboly shodné se vstupní větou, očekáváme na vrcholu neterminál. Není-li tam, tj. je-li tam terminál či je zásobník prázdný, pak vstupní věta obsahuje neplatnou lexému a opět končíme s chybou. S neterminálem na vrcholu zásobníku a novou vstupní lexémou můžeme opět přejít k vyhledání pravidel v GLL tabulce a celý proces expanze/porovnání zopakovat.

Algoritmus 4.5.4: GLL syntaktický analyzátor

Vstup: Věta w, gramatický systém $\sigma = (N, T, S, \ldots)$ a GLL tabulka T_{GLL}
Výstup: Odpověď na otázku $w \in L(\sigma)$

```
1:  Stack := S                                              ▷ Zásobník
2:  Token := přečti další symbol z w                        ▷ Aktuální vstupní lexéma
3:  while true do
4:      TopNonterminal := najdi vrchní neterminál na zásobníku
5:      Rules := pomocí Token a TopNonterminal získej seznam pravidel z T_GLL
6:      if Rules = ∅ then
7:          return false                                    ▷ Žádné pravidlo → syntaktická chyba
8:      end if
9:      foreach Rule ∈ Rules do
10:         if levá strana Rule je na zásobníku Stack then
11:             nahraď na zásobníku Stack levou stranu pravidla Rule pravou stranou
12:         else
13:             return false                                ▷ Chybí neterminál → syntaktická chyba
14:         end if
15:     end foreach
16:                                                         ▷ Expanze ↑ / ↓ porovnání
17:     while vrchol zásobníku Stack = Token do
18:         if Token = $ then
19:             return true                                 ▷ Věta přijata
20:         end if
21:         Stack.pop()
22:         Token := přečti další symbol z w
23:     end while
24:     if vrchol zásobníku Stack ∈ T ∨ Stack = {ε} then
25:         return false                                    ▷ Neočekávaná lexéma → syntaktická chyba
26:     end if
27: end while
```

4.6 Generativní síla syntaktické analýzy založené na GLL tabulce

Na závěr kapitoly věnované zpracování kontextových jazyků pomocí GLL je vhodné rozvinout téma generativní kapacity této struktury a podívat, co jsme pomocí ní získali a naopak, kde ji využít nelze. V následujících řádcích budeme zkratkou $L(LL)$ označovat rodiny jazyků generovaných LL gramatikou pomocí některé LL tabulky a zkratkou $L(GLL)$ budeme označovat rodiny jazyků generovaných HCD gramatickém systémem pomocí některé GLL tabulky.

V prvním kroku se pokusíme ověřit, že GLL obsahuje všechny jazyky z LL. Uvažujme libovolný HCD gramatický systém s právě jednou komponentou a derivací délky =1. Výběr komponenty u takového systému odpadá a derivační kroky jsou prováděny stejně jako obyčejnou BKG. Převedeme-li takový systém na GLL tabulku, bude naprosto totožná s klasickou LL tabulkou, takže můžeme konstatovat, že GLL má nejméně takovou generativní sílu, jako klasická LL.

Už věta 2.0.5, respektive věta 2.1.5, ukazují, že síla CD a HCD gramatických systémů je mnohem větší než síla BKG. Fakt, že lze jazyk generovat systémem, ovšem neznamená, že ho lze pokrýt odpovídající GLL tabulkou, ale už četné příklady uvedené výše ukazují, že množina jazyků analyzovatelných pomocí GLL tabulky je větší, než množina jazyků analyzovatelných pomocí klasické LL tabulky. Pomocí GLL tabulky lze kromě LL jazyků popsat i jistou podmnožinu kontextových jazyků. Tyto kontextové jazyky musí spadat do třídy deterministických jazyků – podmínka plynoucí z užití deterministického zásobníkového automatu jako syntaktického analyzátoru. Z praktického hlediska, jak ilustruje následující příklad, to takový problém nečiní, neboť ve velké části případů je možné drobnou změnou syntaxe determinismus získat.

Příklad 4.6.1. Uvažujme jazyky $L_{ndet} = \{ww \mid w \in \{a,b\}^+\}$ a $L_{det} = \{wuw \mid w \in \{a,b\}^+\}$. V obou případech se jedná o replikaci, zatímco v prvním případě čistou (obsahující pouze terminály „a“ a „b“), v druhém případě je mezi replikované řetězce vložen symbol „u“. Oba jazyky je možné převést pomocí zmíněných transformačních algoritmů na GLL tabulky. Transformace v pořádku proběhne pro oba jazyky a dostaneme dvě velice podobně vypadající GLL tabulky. Problém se projeví až při samotné syntaktické analýze – zatímco analyzátor pro L_{det} bude fungovat přesně, analyzátor pro L_{ndet} nepřijme žádný řetězec – bude pouze akumulovat symboly na zásobníku a po nalezení symbolu \$ prohlásí libovolný řetězec za syntakticky chybný. Jinak řečeno, na zásobníku během procesu syntaktické analýzy vytvoří řetězec $wwww$ a po zpracování ww stejný řetězec na zásobníku zbude.

Musíme konstatovat, že GLL tabulku lze využít pouze pro zpracování deterministických kontextových jazyků. Obdobný poznatek platí i pro bezkontextové jazyky – GLL nelze použít pro zpracování nedeterministických bezkontextových jazyků (bezkontextové jazyky, pro něž neexistuje žádný deterministický zásobníkový automat, který by je přijímal). Ačkoliv lze sestrojit GLL tabulku i pro nedeterministický bezkontextový jazyk, nelze jej přijímat pomocí žádného GLL analyzátoru. Příkladem takového jazyka může být $L_{ndb} = \{xy \mid x \in \{a,b\},\ y = reverse(x)\}$.

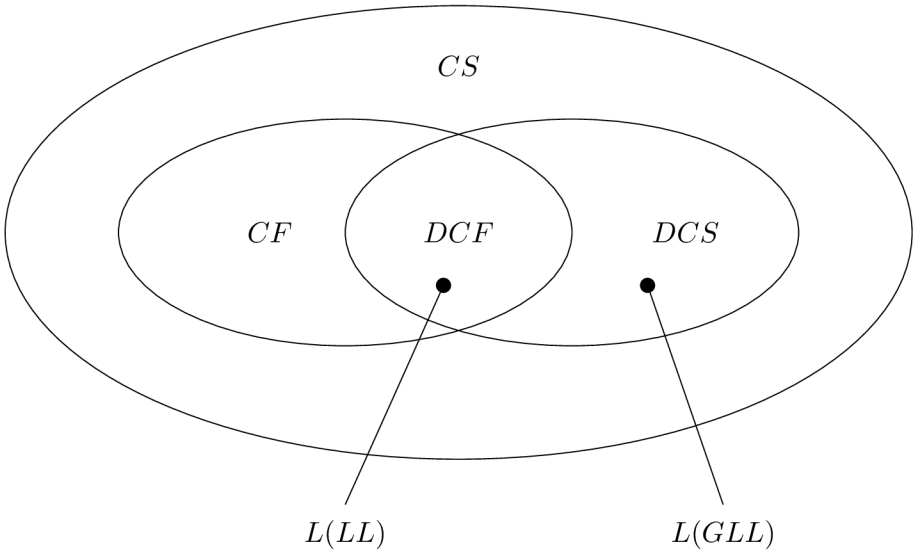
Na závěr uvádíme shrnutí rozdílů v generativní síle GLL oproti LL:

(i) $L(LL) \subset CF$,

(ii) $L(LL) \subseteq DCF \subset L(GLL) \subseteq DCS \subset CS$,

(iii) $DCF = HCD_1(=k) \subset HCD_2(=k) \subseteq HCD_r(=k) \subseteq HCD_\infty(=k) \subseteq MAT$, $\forall(k,r) : k \geq 2 \wedge r \geq 3$,

kde DCF je třída deterministických bezkontextových jazyků a DCS je třída deterministických kontextových jazyků. $HCD_n(=k)$ popisuje rodiny jazyků generovaných HCD gramatickými systémy s maximálně n komponentami v módu $=k$. Obrázek 4.6.1 zobrazuje uvedené relace graficky.



Obrázek 4.6.1 Hierarchie rodin jazyků CS, DCS, CF, DCF a oblast užití LL, respektive GLL, syntaktických analyzátorů.

5 Snížení složitosti užitím gramatických systémů

Druhým velkým cílem stojícím za výzkumem gramatických systémů, je snížení složitosti. Na tento cíl se můžeme podívat ze dvou pohledů:

(i) snížení počtu použitých pravidel a zrychlení výpočtu,

(ii) zjednodušení gramatiky za účelem snazšího porozumění a jednoduššího návrhu.

Oba body lze uspokojivě pokrýt pomocí nějakého CD gramatického systému. Začneme s klasickou BKG gramatikou a pokusíme se vytvořit systém, který tuto gramatiku nahradí s užitím menšího počtu pravidel. Posléze navrhneme způsob převodu tohoto systému na syntaktický analyzátor s užitím známých metod pro bezkontextové jazyky.

Příklad 5.0.1. Uvažujme LL gramatiku

$$
\begin{aligned}
G_{LL1} &= (N, T, PROGRAM, P),\\
N &= \{PROGRAM, STAT, STATLIST, E, \overline{E}, T, \overline{T}, V, \overline{V}, U, \overline{U}, F\},\\
T &= \{begin, end, read, write, id, if, (,), =, <, >, ==, !=, +, -, *, /, ||, \&\&, ;\},\\
P &= \{PROGRAM \to begin\ STATLIST,\\
&\quad STATLIST \to STAT; STATLIST \mid end,\\
&\quad STAT \to if(E)\ STAT \mid id = E \mid read\ id \mid write\ E,\\
&\quad E \to T\overline{E},\ \overline{E} \to ||T\overline{E} \mid \&\&T\overline{E} \mid \varepsilon,\\
&\quad T \to U\overline{T},\ \overline{T} \to\ <U\overline{T} \mid >U\overline{T} \mid ==U\overline{T} \mid !=U\overline{T} \mid \varepsilon,\\
&\quad U \to V\overline{U},\ \overline{U} \to +V\overline{U} \mid -V\overline{U} \mid \varepsilon,\\
&\quad V \to F\overline{V},\ \overline{V} \to *F\overline{V} \mid /F\overline{V} \mid \varepsilon,\\
&\quad F \to (E) \mid id\}.
\end{aligned}
$$

Jedná se o gramatiku popisující jednoduchý programovací jazyk nabízející příkazy pro vstup a výstup dat, příkaz větvení, příkaz přiřazení a vyhodnocování aritmetických výrazů. Tato gramatika je vhodná pro některou metodu prediktivní syntaktické analýzy pracující shora dolů. Sestavit z ní LL tabulku je triviální úkol. Pravidla pro vyhodnocování aritmetických výrazů jsou na první pohled nesrozumitelná, a přestože pracujeme pouze s několika málo operátory, máme pro ně mnoho pravidel. Precedence jednotlivých operátorů, které jsou definovány přímo gramatikou, mohou čtenáři na první pohled uniknout. Intuitivně rozumíme, že $prec(\{*, /\}) > prec(\{+, -\}) > prec(\{<, >, ==, !=\}) > prec(\{||, \&\&\}) >$

$prec(\{=\})$, kde $prec(\zeta)$ je priorita operátorů z množiny ζ, ovšem s jistotou to můžeme prohlásit až po chvíli bádání nad zadanou gramatikou. Zde máme pouze 5 prioritních úrovní, ale reálné programovací jazyky jich mají 15 a více. Navíc naše gramatika počítá s tím, že operátory jsou pouze levě asociativní, což v praxi nenastává příliš často. Na zpracování operátorů je mnohem vhodnější precedenční syntaktická analýza zdola nahoru, než prediktivní shora dolů. Ovšem řídící konstrukce se realizují velice obtížně v precedenční syntaktické analýze, a proto je vhodným přístupem zkombinovat obě metody. Zkusme nyní na základě této gramatiky navrhnout gramatický systém, který bude vhodný pro tuto kombinaci dvou zcela odlišných metod.

$$
\begin{aligned}
\sigma_{14} &= (N, T, PROGRAM, P_1, P_2),\\
N &= \{PROGRAM, STATLIST, STAT, E\},\\
T &= \{begin, end, read, write, id, if, (,), =, <, >, ==, !=, +, -, *, /, ||, \&\&, ;\},\\
P_1 &= \{PROGRAM \to begin\ STATLIST,\\
&\quad STATLIST \to STAT; STATLIST \mid end,\\
&\quad STAT \to if(E)\ STAT \mid id = E \mid read\ id \mid write\ E\},\\
P_2 &= \{E \to E + E \mid E - E \mid E * E \mid E/E \mid E\&\&E \mid E \,||\, E \mid\\
&\quad E < E \mid E > E \mid E == E \mid E!= E \mid (E) \mid id\}.
\end{aligned}
$$

Dostáváme gramatický systém o dvou komponentách. První komponenta bude pracovat shora dolů a druhá komponenta zdola nahoru. Necháme-li celý systém pracovat v módu $f \in \{\geq 1, =1, t\}$, pak platí $L(G_{LL1}) = L_f(\sigma_{14})$. Zároveň dochází k výraznému zjednodušení – zredukovali jsme počet pravidel z 27 na 19 a úplně zmizela ε-pravidla. První komponenta systému popisuje řídící konstrukce jazyka, zatímco druhá přehledně popisuje zpracování aritmetických výrazů. Precedence jednotlivých operátorů už dále nejsou součástí gramatiky, ale budou definovány až jako součást metody syntaktické analýzy – konkrétně precedenční tabulkou. Další nespornou výhodou je redukce počtu neterminálů (z 12 na 3), což vede k zmenšení LL tabulky. LL tabulka z prvního příkladu bude mít 252 buněk, zatímco součet buněk LL tabulky a precedenční tabulky druhého systému činí pouze 232. Můžeme tedy prohlásit, že paměťová náročnost se nezvýšila.

Může zde vyvstat otázka, zdali je možné systém σ_{14} transformovat na GLL tabulku. Nejedná se o LL gramatický systém kvůli levé rekurzi v pravidlech komponenty P_2. GLL analyzátor také pracuje shora dolů, takže by bylo zapotřebí se levé rekurze zbavit, což je samozřejmě možné, ale dopracovali bychom s k výsledku velice blízkému G_{LL1}. Vzhledem k faktu, že se pohybujeme na úrovni bezkontextových struktur, není nasazení GLL vhodné. Problém pojmeme ze zcela odlišného pohledu, komponenty systému σ_{14} izolujeme a v dalších úvahách je budeme vnímat jako samostatné gramatiky. Zároveň budeme předpokládat, že již máme sestrojený syntaktický analyzátor pro každou komponentu zvlášť. Analyzátor pro P_1 implementuje LL prediktivní syntaktickou analýzu, zatímco analyzátor pro P_2 implementuje precedenční syntaktickou analýzu. Systém σ_{14} je pouze zástupcem celé rodiny, na kterou se prezentovaná metoda převodu vztahuje. Do několika bodů shrneme omezení, která musí být dodržena libovolným systémem, aby bylo možné metodu aplikovat.

Věta 5.0.2. *CD Gramatický systém σ je převoditelný na LL prediktivní analyzátor s precedencemi, pokud platí:*

(i) *σ obsahuje alespoň 2 komponenty.*

(ii) *σ obsahuje právě jednu komponentu typu P_υ a libovolný počet komponent typu P_ψ.*

(iii) *Komponenta P_υ tvoří LL gramatiku – neobsahuje žádnou dvojici pravidel $A \to x$, $A \to y$ pro kterou platí $First(x) \cap First(y) \neq \emptyset$, kde $A \in N$ a $x, y \in (N \cup T)^*$.*

(iv) *Komponenty P_ψ jsou precedenční gramatiky, tedy (1) neobsahují žádná ε-pravidla a (2) na pravé straně se v žádném pravidle nevyskytují 2 po sobě jdoucí neterminály.*

Máme-li k dispozici systém splňující zadaná kriteria, zbývá vyřešit ještě několik praktických problémů k propojení obou metod syntaktické analýzy.

(i) Jak zvolit aktivní komponentu?

(ii) Jak dlouho má aktivní komponenta pracovat?

(iii) Jakým způsobem předávat komunikovanou větnou formu mezi komponentami?

Abychom byli schopni odpovědět na položené otázky, na chvíli se vrátíme do teoretické roviny. Pro systém σ_{14} můžeme říct, že obě komponenty budou pracovat v módu t a jsme z teoretického pohledu hotovi – komponenta P_1 bude pracovat dokud to bude možné, poté předá aktivitu P_2, ta po provedení ukončující derivace opět aktivuje P_1, atd. – komponenty budou pravidelně střídat. I v módech ≥ 1 a $=1$ se komponenty v práci na větné formě budou střídat, ale bude docházet k situaci, kdy je několikrát po sobě za aktivní komponentu vybrána P_1, respektive P_2, což znamená jistou režii navíc. Bez újmy na obecnosti se můžeme omezit pouze na nejlevější derivace, generovaný jazyk se nezmění, pouze se komponenty budou střídat častěji. Pozorný čtenář už tuší, že se komponenty budou střídat, kdykoliv komponenta P_1 vygeneruje neterminál E, který leží na levé straně pravidla z P_2, avšak sama komponenta P_1 tento neterminál na levé straně žádného pravidla nemá. Takový neterminál bude patřit do množiny $Common^{LSD}$ systému σ_{14}. Neterminály vyskytující se na levých stranách obou komponent zařadíme do množiny $Common$. Obě tyto množiny si záhy zavedeme formálně.

Definice 5.0.3. Nechť σ je gramatický systém s komponentami P_i a P_j. Množina společných neterminálů těchto komponent, značená $Common(P_i, P_j)$, množina levě rozdílných (*left side distinct*) společných neterminálů těchto komponent, značená $Common(P_i, P_j)^{LSD}$ a množina společných neterminálů přítomných na levé straně obou těchto komponent (*both left sides*), značená $Common(P_i, P_j)^{BLS}$, jsou definovány:

$$
\begin{aligned}
Common(P_i, P_j) &= \big\{A \in N \mid A \to x \in P_j \wedge B \to yAz \in P_i,\ B \in N,\ x, y, z \in (N \cup T)^*\big\},\\
Common^{LSD}(P_i, P_j) &= \big\{A \in N \mid A \to x \in P_j \wedge B \to yAz \in P_i,\ A \to v \notin P_i,\\
&\qquad B \in N,\ x, y, z, v \in (N \cup T)^*\big\},\\
Common^{BLS}(P_i, P_j) &= Common(P_i, P_j) \setminus Common^{LSD}(P_i, P_j).
\end{aligned}
$$

Všechny tyto množiny obsahují pouze neterminály, ale nejsou rozhodně identické. Množina $Common$ obsahuje libovolné neterminály, které se nachází v pravidlech jak první, tak druhé komponenty, v druhé komponentě alespoň jedno na levé straně. Množina $Common^{LSD}$ obsahuje neterminály, které se nachází v první komponentě pouze na pravé straně a v druhé komponentě na levé straně. Množina $Common^{BLS}$ obsahuje neterminály, které se nachází

na levých stranách obou komponent. Pro gramatický systém σ_{14} mají tyto množiny podobu: $Common(P_1, P_2) = \{E\}$, $Common^{LSD}(P_1, P_2) = \{E\}$ a $Common^{BLS}(P_1, P_2) = \emptyset$. Můžeme si všimnout, že žádná z těchto množin není komutativní grupou, tedy sestrojíme-li množiny $Common(P_i, P_j)$ a $Common(P_j, P_i)$, dostaneme rozdílný výsledek. Pro naše účely budeme vždy využívat jako první parametr metodu pracující shora dolů a jako druhý parametr metodu pracující zdola nahoru. Dále budeme potřebovat množinu $Follow$. S několika jejími podobami jsme se již setkali, zde se budeme držet té nejběžnější, kterou uvádí Meduna v [3].

Definice 5.0.4. Nechť $G = (N, T, S, P)$ je BKG. Pro každé $A \in N$ definujeme množinu $Follow(A)$ jako

$$Follow(A) = \{y \in T \mid S \Rightarrow^* xAyz,\, x, z \in (N \cup T)^*\} \cup \{\$ \mid S \Rightarrow^* xA, x \in (N \cup T)^*\}.$$

Tato množina se samozřejmě využije k vytvoření LL tabulky pro komponentu P_1, ale my ji navíc využijeme k propojení obou metod syntaktické analýzy. V tomto bodě je vhodné se zaměřit na komunikaci. Začneme u LL tabulky, kterou si rozšíříme o symbol $\ggg P_i$. Zavedeme si LL tabulku jako $T_{LL} = \big\{\alpha(a \in T, A \in N)\big\}$, kde $\alpha(a, A) = p \vee \ggg P_i$, přičemž P_i je libovolná komponenta gramatického systému, kromě aktivní komponenty (pro náš příklad $P_i = P_2$). Připustit v tomto místě aktivní komponentu nedává smysl, protože by mohlo dojít k zacyklení analyzátoru. Sémantika $\ggg P_i$ je definována následovně:

(i) Odstraň nejvrchnější neterminál A ze zásobníku,

(ii) inicializuj komponentu P_i,

(iii) předej řízení P_i.

Aktivovaná komponenta P_i má samozřejmě k dispozici stejnou vstupní lexému a začíná syntaktickou analýzu vždy od začátku. Pracuje přesně podle své precedenční tabulky, která bude rovněž o jeden symbol bohatší, konkrétně $\lll$. Precedenční tabulku si definujeme jako $T_{prec} = \big\{\beta(a \in T, A \in N)\big\}$, kde $\beta(a, A) = \delta$, $\delta \in \{<, =, >, \lll, \varepsilon\}$. ε reprezentuje prázdné pole. Sémantika $\lll$ je také jednoduchá:

(i) Prověřovaná část programu je syntakticky správná, na vrcholu zásobníku se nyní nachází komunikovaný řetězec.

(ii) Pokud je to žádoucí, vrať komunikovaný řetězec komponentě, která o něj požádala (předala aktivitu precedenčnímu analyzátoru).

(iii) Předej řízení volající komponentě.

Předávat řetězec nazpět není nutné v každém případě – pokud chceme pouze odpověď na otázku syntaktické správnosti, řetězec můžeme zahodit. Mohou nastat situace, kdy volající komponenta potřebuje na základě podoby řetězce vykonat nějakou přídavnou akci. V takovém případě je nutné řetězec umístit na vrchol zásobníku volající komponenty ještě před tím, než se jí vrátí řízení.

Ještě zbývá vyřešit, jak nové symboly rozmísit do tabulek. Začneme precedenční tabulkou, zde umístíme symbol $\lll$ do všech polí na průsečíku vstupní lexémy a a dna zásobníku \$, která vyhovují podmínce $\exists A \in Common^{LSD}(P_i, P_j)$: $a \in Follow(A)$, kde P_i je volající

komponenta a P_j je komponenta pracující zdola nahoru. Dále do všech prázdných polí na průsečíku a a libovolného terminálu b kromě otevíracích závorek, která splňují výše uvedenou podmínku, umístíme znak $>$ symbolizující redukci. Komponenta P_j tedy ukončí práci na větné formě, pokud narazila na terminál, který může ve své větné formě odvodit volající komponenta hned za komunikovaným řetězcem. Aktivní komponenta se vzdává aktivity až v momentě, kdy již není možné pokračovat. Situace, kdy je stejný terminál odvoditelný aktivní i volající komponentou je vyloučena.

Do LL tabulky komponenty P_i umístíme pro každý neterminál $A \in N$ a vstupní lexému $a \in T$ symbol $\ggg P_j$ následovně:

(i) $\alpha(a, A) = \ggg P_j$, pokud $A \in Common^{LSD}(P_i, P_j)$ a zároveň $a \in First^{P_j}(A)$ pro nějakou komponentu P_j pracující zdola nahoru, kde $First^{P_j}(A)$ je množina $First(A)$ vytvořená pouze z pravidel komponenty P_j.

(ii) $\alpha(a, A) = \ggg P_j$, pokud $A \in Common^{BLS}(P_i, P_j)$, $a \in First^{P_j}(A)$ a zároveň platí $a \notin First^{P_i}(A)$, jinak $\alpha(a, A) = p \in P_i$.

Zde má znovu přednost aktivní komponenta, a pokud sama může odvodit terminál a, udělá to, v opačném případě předá řízení jiné komponentě, která to může provést. Na první pohled může situace, kdy terminál a lze odvodit dvěma komponentami pracujícími zdola nahoru, způsobit problém. Tato situace nemůže nikdy v LL systému nastat, a přestože nepracujeme s LL systémem, je vyloučena kvůli bodu (iii) věty 5.0.2.

Ze zavedených rozšíření je zřejmý i způsob práce takového systému. Komponenta využívající prediktivní syntaktickou analýzu se chová jako master. Nutně musí začínat zpracování větné formy. Pod sebou má libovolný počet podřízených komponent pracujících metodou precedenční syntaktické analýzy. Master pracuje samostatně dokud může, poté vybere podle LL tabulky odpovídající akci – buď větu odmítne (je syntakticky chybná), nebo nechá podproblém vyřešit některou z podřízených komponent. Podřízené komponenty nemohou komunikovat přímo mezi sebou.

Vraťme se k našemu příkladu a k vytvoření tabulek. Definici množiny $Common^{LSD}$ pro komponenty P_1 a P_2 odpovídá pouze neterminál E. Pouze E je odvoditelné komponentou P_1 a zároveň na levé straně pravidla je přítomno pouze v P_2, takže $Common^{LSD}(P_1, P_2) = \{E\}$. Pro tento neterminál má množina $Follow$ pouze 2 prvky, tedy $Follow(E) = \{), ;\}$. Do precedenční tabulky druhé komponenty přidáme na průsečík dna zásobníku \$ a prvků z $Follow(E)$ symbol $\lll$. Výsledek zobrazuje tabulka 5.0.1 pracující s pravidly ve tvaru

$$P_1 = \left\{ \begin{array}{ll} 1: & PROGRAM \to begin\ STATLIST, \\ 2: & STATLIST \to STAT; STATLIST, \\ 3: & STATLIST \to end, \\ 4: & STAT \to if(E)\ STAT, \end{array} \quad \begin{array}{ll} 5: & STAT \to id = E, \\ 6: & STAT \to read\ id, \\ 7: & STAT \to write\ E \end{array} \right\}$$

Zajímavostí je, že v tabulce vůbec nemáme vyplněný sloupec pro \$ – ten totiž nikdy nebude následovat ve větné formě za E, jeho funkci nahrazují „;“ a „)“. Tyto dvě lexémy budou vždy indikovat redukci. Pouze v případě, že nastala syntaktická chyba zůstane pole prázdné a pro kombinaci „(“ (vrchní symbol na zásobníku) a „)“ (vstupní lexéma) musí být přítomno $=$, aby bylo možné vytvářet rovnoměrné závorkové struktury. Fakt, že $Common^{BLS} = \emptyset$ nám ulehčuje tvorbu LL tabulky. Před jejím vytvořením se ještě hodí

uvést množinu $First^{P_2}(E) = \{id, (\}$. LL tabulku budeme vytvářet podle stejného algoritmu, který platí pro BKG, pouze na průsečíku E se symboly z $First^{P_2}(E)$ předáme řízení pomocí $\ggg P_2$. Výsledek zobrazuje tabulka 5.0.2.

T ╲ T		**Vstupní token**													
		+	−	*	/	==	!=	<	>	&&	||	*id*	;	(	)
Stack	+	>	>	<	<	>	>	>	>	>	>	<	>	<	>
	−	>	>	<	<	>	>	>	>	>	>	<	>	<	>
	*	>	>	>	>	>	>	>	>	>	>	<	>	<	>
	/	>	>	>	>	>	>	>	>	>	>	<	>	<	>
	==	<	<	<	<	>	>	>	>	>	>	<	>	<	>
	!=	<	<	<	<	>	>	>	>	>	>	<	>	<	>
	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>
	>	<	<	<	<	>	>	>	>	>	>	<	>	<	>
	&&	<	<	<	<	<	<	<	<	>	>	<	>	<	>
	||	<	<	<	<	<	<	<	<	>	>	<	>	<	>
	id	>	>	>	>	>	>	>	>	>	>		>		>
	(	<	<	<	<	<	<	<	<	<	<	<		<	=
	)	>	>	>	>	>	>	>	>	>	>		>		>
	$	<	<	<	<	<	<	<	<	<	<	<	⋘	<	⋘

Tabulka 5.0.1 Precedenční tabulka pro komponentu P_2 systému σ_{14}. Pro přehlednost byly vynechány zcela prázdné řádky a sloupce.

N ╲ T		**Vstupní token**										
		begin	*end*	*read*	*write*	*if*	*id*	(	)	+ . . . ||	;	$
Stack	*PROGRAM*	1										
	STATLIST		3	2	2	2	2					
	STAT			6	7	4	5					
	E						$\ggg P_2$	$\ggg P_2$				

Tabulka 5.0.2 Rozšířená LL tabulka pro komponentu P_1 systému σ_{14}.

Příklad 5.0.5. Nyní uvažujme scénář, kdy se do našeho systému přidá ještě jedna komponenta, tedy systém bude vypadat:

$$
\begin{aligned}
\sigma_{15} &= (N, T, PROGRAM, P_1, P_2, P_3),\\
N &= \{PROGRAM, STAT, E, \overline{E}, F\},\\
T &= \{begin, end, read, write, id, if, none, (,), [,], <<, >>, <> +, -, *, /, ;\},\\
P_1 &= \{PROGRAM \to begin\ STATLIST,\\
&\quad STATLIST \to STAT; STATLIST \mid end,\\
&\quad STAT \to if(\overline{E})\ STAT \mid id = \overline{E} \mid read\ id \mid write\ \overline{E},\\
&\quad \overline{E} \to E \mid [F], F \to none\},
\end{aligned}
$$

$$P_2 = \{E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid id\},$$
$$P_3 = \{E \to -E,\ F \to F << F \mid F >> F \mid F <> F \mid (F) \mid id\}.$$

Kromě toho nám přibylo pravidlo $F \to none$, tedy z neterminálů F může odvozovat master i komponenta P_3. Nebudeme zabředávat do sémantických detailů operací třetí komponenty a přikročíme k podstatnějším zajímavostem. V systému přibylo pravidlo $\overline{E} \to [F]$. Hranaté závorky v tomto pravidle jsou vynuceny, bez nich by nebylo možné systém převést, neboť bychom nemohli vybrat správné pravidlo z důvodu, že $First(F) = First(E)$. Tímto opatřením zachováváme vlastnosti LL systému (pomineme-li levou rekurzi). Sestavme si množiny *Common* a *First*:

$$\begin{aligned}
Common^{LSD}(P_1, P_2) &= \{E\},\\
Common^{BLS}(P_1, P_2) &= \emptyset,\\
Common^{LSD}(P_1, P_3) &= \{E\},\\
Common^{BLS}(P_1, P_3) &= \{F\},\\
First^{P_2}(E) &= \{id, (\},\\
First^{P_3}(E) &= \{-\},\\
First^{P_3}(F) &= \{id, (\}.
\end{aligned}$$

S pomocí těchto množin můžeme sestrojit LL tabulku – výsledek zobrazuje tabulka 5.0.3, přičemž přepisovací pravidla jsou zavedena následovně:

$$P_1 = \left\{ \begin{array}{ll}
1: & PROGRAM \to begin\ STATLIST, \\
2: & STATLIST \to STAT; STATLIST, \\
3: & STATLIST \to end, \\
4: & STAT \to if(\overline{E})\ STAT, \\
5: & STAT \to id = \overline{E},, \\
6: & STAT \to read\ id, \\
7: & STAT \to write\ \overline{E}, \\
8: & \overline{E} \to E, \\
9: & \overline{E} \to [F], \\
10: & F \to none.
\end{array} \right\}$$

N ╲ T		**Vstupní token**											
		begin	*end*	*read*	*write*	*if*	*id*	*none*	(	)	[	]	−
Stack	*PROGRAM*	1											
	STATLIST		3	2	2	2	2						
	STAT			6	7	4	5						
	$\overline{E}$						8		8		9		
	E						$\ggg P_2$		$\ggg P_2$				$\ggg P_3$
	F						$\ggg P_3$	10	$\ggg P_3$				

Tabulka 5.0.3 Rozšířená LL tabulka pro komponentu P_1 systému σ_{15}. Prázdné sloupce byly pro přehlednost vynechány.

6 Implementace aplikace

Doposud jsme si z teoretického hlediska osvětlili gramatické systémy, s jejich znalostí jsme rozšířili několik známých metod syntaktické analýzy a navrhli proces transformace těchto systémů na struktury vhodné pro deterministickou analýzu generovaného jazyka. V této kapitole se pokusíme proces transformace plně automatizovat, navrhneme a implementujeme aplikaci, respektive veřejnou knihovnu, která ze zadaného CD či HCD gramatického systému vytvoří GLL tabulku. Rovněž sestrojíme GLL syntaktický analyzátor navržený v sekci 4.5.

Klíčové algoritmy transformace jsme již prezentovali na množinách v sekci 4.1. Zde jsme používali matematické modely na vysoké úrovni abstrakce. Pro implementaci je zajisté vhodné využít takové prostředí, které alespoň některé z nich nativně podporuje. Zároveň můžeme předpokládat, že převod jazyka do tabulky je operace, která se provádí pouze jednou pro zadaný jazyk, tedy rychlost transformace není klíčovým faktorem. Z uvedených důvodů byl vybrán jako prostředek implementace programovací jazyk Python. Tento jazyk již v základu podporuje práci s množinami, slovníky, seznamy a dalšími informatickými a matematickými modely. Jedná se o převážně interpretovaný jazyk, výpočet bude sice pomalejší než s užitím C či C++, ale z pohledu lidského pozorovatele se může jednat o nepatrný rozdíl. Skriptovací jazyk navíc umožní uživateli jednoduší specifikace problému.

6.1 Transformace na GLL

Nyní se stručně podíváme na modul `grammarSystem.py`, koncipovaný jako knihovna umožňující definici gramatických systémů v jazyce Python a jejich následný převod na GLL tabulku.

6.1.1 Struktura

Základní algoritmy transformace zapouzdříme do třídy `AbstractGrammarSystem`. Tato třída je schopna vytvářet množiny $First$, $Empty$ a $Follow$, respektive $Follow_f^{CD}$, $finD$, a zároveň generovat GLL tabulku. Tato třída zapouzdřuje společné prvky kooperačně distribuovaných gramatických systémů. Nepředpokládá se, že bude využívána koncovými uživateli – pro ně jsou zde připraveny podtřídy `CD` a `HCD`. Tyto třídy poskytují uživatelské rozhraní pro práci s CD či HCD gramatickými systémy, přizpůsobené pro konkrétní typ systému.

6.1.2 Vstupy

Gramatické systémy k transformaci se specifikují přímo v jazyce Python. Třídy `CD` a `HCD` mají pouze jediný konstruktor pomocí nějž je zadána abeceda neterminálů, abeceda terminálů a počáteční neterminál či kolekce neterminálů, ze které se začíná derivovat. U třídy `CD` je navíc posledním parametrem konstruktoru mód práce systému. Komponenty systému se přidávají metodou `addComponent` zpracovávající variabilní počet parametrů. U třídy `HCD` je prvním parametrem této funkce mód práce komponenty. Ostatními parametry jsou jednotlivá pravidla, každé pravidlo je instance třídy `Rule`. Konstruktor této třídy očekává nejméně jeden argument, kterým je jediný neterminál $A \in N$ tvořící levou stranu. Pokud není zadán další parametr, jedná se o ε-pravidlo „$A \to \varepsilon$“, v opačném případě další parametry tvoří pravou stranu pravidla. Specifikace takového CD gramatického systému může vypadat následovně:

```
System = CD({'S', 'X', 'Y'}, {'a', 'b'}, 'S', "=2")
System.addComponent(Rule('S', 'Y'), Rule('Y', 'X'),
                    Rule('X', 'a', 'X', 'b'))
System.addComponent(Rule('X', 'Y'), Rule('Y'))
```

V tomto případě se jedná o systém generující jazyk $L = \{a^n b^n \mid n \ mod \ 2 = 0, n \geq 0\}$, tedy jazyk, který obsahuje pouze sudý počet terminálů „a“ a „b“. Systém má právě 2 komponenty a pracuje v módu =2. Posledním pravidlem druhé komponenty je právě ε-pravidlo. Podobně můžeme sestrojit HCD gramatický systém:

```
System = HCD({'S', 'H'}, {'a', 'b'}, ['S', 'S', 'S'])
System.addComponent('=3', Rule('S', 'a', 'H', 'b'))
System.addComponent('=3', Rule('H', 'a', 'S', 'b'))
System.addComponent('t', Rule('S'), Rule('H'))
```

Jedná se o systém se třemi komponentami pracujícími v módech =3, =3 a t. Systém začíná pracovat z počáteční konfigurace SSS a generuje věty jazyka $L = \left\{(a^n b^n)^3 \mid n \geq 0\right\}$. Příklady specifikace dalších jazyků, o nichž jsme se již zmínili v teoretické části jsou k nalezení v přiložených zdrojových textech.

6.1.3 Množina Empty

V první řadě je potřeba implementovat množinu *Empty*, tato množina je prerekvizitou k vytváření *First* i *Follow*. Máme-li zadaný řetězec a $G = (N, T, S, P)$ je BKG, pak zjistit, zdali může být řetězec vymazán pomocí G, je triviální úlohou. Nejprve sestrojíme množinu *Empty* pro každý neterminál.

(i) $\forall(A \in N)$: $Empty(A) := \emptyset$,

(ii) $\forall(A \in N)$: $Empty(A) := \{\varepsilon\}$, pokud $A \to \varepsilon \in P$

(iii) $\forall(A \in N)$: $Empty(A) := \{\varepsilon\}$, pokud $A \to x_1 \ldots x_n \in P$, kde $x_i \in N \wedge$ $Empty(x_i) = \{\varepsilon\}$, pro $1 \leq i \leq n$, $n \in \mathbb{N}$.

Ve třech jednoduchých krocích jsme získali *Empty* pro každý neterminál, poté *Empty* řetězce $u = x_1 \ldots x_n$ vypočteme jako

$$Empty(u) := \{\varepsilon\}, \text{ pokud } Empty(x_i) = \{\varepsilon\} \text{ pro } 1 \leq i \leq n, \text{ jinak } \emptyset.$$

U gramatických systémů není situace tak jednoduchá, neboť se nám do výpočtů ještě mísí derivace délky k. Ukončující derivace nám situaci nijak neztíží, pro systém využívající pouze ukončujících derivací můžeme použít výše uvedený způsob pro získání *Empty*. Obsahuje-li systém pracující v módu $=k$ nějakou komponentu s ε-pravidlem, komponenta může neterminál vymazat z větné formy pouze pokud může provést ještě dalších $k-1$ derivačních kroků. Například komponenta $P_\kappa = \{B \to a, C \to \varepsilon\}$ pracující v módu $=2$ může vymazat C jen pokud je ve větné formě přítomno B nebo další C. Komponenta totiž může pracovat pouze na větné formě obsahující BC, B^2 či C^2, tedy B ani samotné C nemůže být nikdy vymazáno. Pozorný čtenář už tuší, že pomocí komponenty pracující v módu $=k$ lze vymazat libovolné kombinace neterminálů $A \in N$, pro které existuje pravidlo $A \to \varepsilon \in P_\kappa$. Jinak řečeno, větnou formu vymazáváme po n-ticích, nikoliv po jednotlivých neterminálech.

Na aplikační úrovni si nejprve spočítáme pro každou komponentu množinu n-tic neterminálů odstranitelných z větné formy, což jsou právě kombinace těch neterminálů, pro něž existuje v komponentě ε-pravidlo. Tuto funkcionalitu zajišťuje metoda `empty`. Množinu *Empty* pro zadaný řetězec poté určí metoda `isEmpty` se znalostí všech n-tic, které mohou být odstraněny. Tato metoda nejprve ověří, jestli je celá n-tice obsažena ve větné formě a případně ji vymaže. Postup se opakuje, dokud nedostaneme ε nebo nějakou větnou formu, která nemůže být vymazána.

Při výpočtu samozřejmě uvažujeme kombinace s opakováním, tedy pro $\{A, B\}$, dostáváme 3 možnosti – AA, AB, BB. Počet kombinací roste v závislosti na počtu ε-pravidel a délce derivačního kroku. Například pro komponentu s 3 ε-pravidly o délce kroku $=3$ je počet n-tic 10, pro $=4$ to je 15 a pro $=5$ to činí 21. Mnohé z prověřovaných situací nemohou nikdy během odvozování nastat. Pokud si je dopředu návrhář systému vědom, že může být vymazáno pouze několik málo kombinací, může přímo zadat množinu odstranitelných kombinací pomocí atributu `Empty` a ušetřit tak výpočetní čas. To lze provést například následujícím způsobem:

```
System.Empty = [['A2', 'B2', 'C2'],['A3', 'B3', 'C3']]
```

6.1.4 First, Follow, GLL a Φ

Implementace množiny *First* je provedena přesně podle algoritmu 4.1.1 metodou `first`. K nalezení prvního neodstranitelného neterminálu ve větné formě použijeme metodu `FirstNotEraseAble` využívající výše zmíněnou operaci `isEmpty`. Sestrojení množiny $Follow^{CD}$ v metodě `follow` provedeme také přesně podle dříve prezentovaného algoritmu 4.1.2, jediná náročná operace je sestrojení množiny Φ na řádku . Tuto množinu v mírně rozšířené podobě využívá i algoritmus 4.1.3 pro sestavení GLL. Proto je vhodné ji implementovat samostatně a výsledek sdílet. Množina Φ je výsledkem simulace všech možných derivačních kroků se zadanou počáteční větnou formou s užitím všech komponent. Tuto simulaci provádí třída `Derivation` po zavolání metody `phi`. V simulaci se prochází počáteční větná forma zleva doprava, tedy využívá se nejlevější derivace. Simulace se provádí v několika na sobě závislých krocích – počet těchto kroků se odvíjí od stupně komponenty. V prvním kroku se vybere z větné formy nejlevější neterminál, na který může být aplikováno pravidlo nějaké komponenty. Na tento neterminál se aplikují všechna pravidla se stejnou levou stranou, z každé jednotlivé aplikace získáme instanci třídy `RuleApplication`. U každé aplikace si pamatujeme kromě výsledné větné formy i komponentu, pomocí níž byla provedena. Na začátku dalšího kroku se podíváme, jestli již počet provedených kroků odpovídá stupni požadovaného komponentou. Pokud ano, přidáme aplikaci do množiny Φ,

v opačném případě najdeme v současné větné formě aplikace nejlevější neterminál, který lze přepsat pomocí dříve využité komponenty. Na ten opět aplikujeme všechna použitelná pravidla a dostáváme novou množinu aplikací. Samotná implementace je mírně rozsáhlejší, zvědavý čtenář jej může nalézt v přílohách. Klíčové je v tomto bodě ještě zmínit metodu `GLL`, která vytváří samotnou GLL tabulku. Aby bylo možné zpracovat i divergující systémy je nutné na závěr každého kroku této metody zredukovat počet neterminálů ve větné formě pomocí `reduce`. Maximální počet jednotlivého neterminálu ve větné formě se stanovuje podle stupně jednotlivých komponent v metodě `rescrictions`, ale uživatel je může změnit následovně:

```
System.restrictions()
System.Restrictions['A'] = 5
```

Je-li použit parametr `MaxDerivation`, pak se maximální povolený počet jednotlivých neterminálů ve větné formě odvíjí pouze od komponenty s nejvyšším stupněm:

```
System.restrictions(MaxDerivation=True)
System.Restrictions['C'] = 5
```

Po zadání základních parametrů systému bude celý výpočet z pohledu uživatele vypadat následovně:

```
System.empty()
System.first()
System.follow()
System.restrictions()
System.GLL()
```

Třída při výpočtu si automaticky generuje závislosti a spočítá pouze potřebné množiny, které doposud nebyly stanoveny. To umožňuje využít zkrácený zápis

```
System.GLL()
```

vedoucí ke stejnému výsledku.

6.1.5 Výstup

Výsledek transformace, tedy GLL tabulka, je dostupný přes atribut `formatedGLLTable`. Tato tabulka se odkazuje na serializovaná pravidla, jež jsou výstupem metody `serializeRules`. Pravidla v tabulce jsou pouze ve formě řetězce, separovaná pomocí čárky. Samotná tabulka je instance datového typu `DataFrame`[1]. K jednotlivým buňkám tabulky lze přistoupit pomocí metody `get_value` . Jednotlivé částí systému je možné vypsat na standardní výstup pomocí metody `print`. Pomocí parametrů lze specifikovat části systému, které se mají vypsat.

(i) `All` tisk všech dostupných informací, stejný efekt je docílen vynecháním parametru.

(ii) `System` zadaný systém ve formě n-tice.

(iii) `Empty` množina *Empty*.

[1] Datový typ z knihovny `pandas` šířené pod BSD licencí, volně dostupné z http://pandas.pydata.org/pandas-docs/stable/overview.html.

(iv) `First` množina *First*.

(v) `Follow` množina *Follow*.

(vi) `Restrictions` omezení na počet neterminálů ve větné formě.

(vii) `Alpha` GLL tabulka jako množina $T_{GLL} = \alpha(a \in T, A \in N)$.

(viii) `GLL` formátovaná GLL tabulka.

Tisk informací k několika různým systémům může potom vypadat takto:

```
System1.print()
System2.print('All')
System3.print('First', 'Follow', 'Empty')
System4.print('System', 'GLL')
```

6.1.6 Zisk klasické LL tabulky

Na základě výsledků konstatovaných v sekci 4.6 je zřejmé, že není problém analyzovat pomocí GLL tabulky všechny jazyky analyzovatelné pomocí LL tabulky. Za použití stejných algoritmů a metod můžeme převádět LL gramatiky na LL tabulky – jednoduše vytvoříme systém pracující v módu =1, jenž obsahuje pouze jednu komponentu. Tento systém má stejnou generativní kapacitu jako BKG a derivace délky 1 také garantuje, že v každé buňce tabulky bude maximálně 1 pravidlo.

6.2 GLL analyzátor

Syntaktický analyzátor založený na GLL tabulce je implementován třídou `GLLParser`. V konstruktoru očekává gramatický systém (σ), ze kterého získá potřebné informace – abecedu terminálů, abecedu neterminálů, serializovaná pravidla, GLL tabulku a počáteční neterminál. Po zavolání metody `analyze` se provede analýza řetězce přesně podle algoritmu 4.5.4, jejímž výsledkem je odpověď na otázku: „Spadá zadaný řetězec do jazyka generovaného pomocí σ?“ Pro zobrazení celé posloupnosti prováděných derivací je nutné použít příznak `Verbose`. Roli lexikálního analyzátoru zde plní třída `Tokenizer`, který rozdělí vstup na tokeny pomocí regulárních výrazů. Očekává se, že jednotlivé symboly budou ve vstupní větě separovány bílými znaky. Plná implementace těchto tříd je k dispozici v přílohách, použity mohou být následujícím způsobem:

```
parser.analyze('a a b b c c', Verbose=True)
print(parser.getError())
```

7 Závěr

V první části práce jsme si z teoretického pohledu představili gramatické systémy jako další matematický model vhodný pro popis formálních jazyků. Diskutovali jsme jejich strukturu, generativní kapacitu a způsob práce. Rozčlenili jsme si je do dvou základních větví – sekvenčně pracujících CD, respektive HCD, gramatických systémů a paralelních PC gramatických systémů. Představili jsme dva základní cíle a zároveň důvody, proč se těmto strukturám věnovat, tj. zvýšení síly syntaktické analýzy při zachování bezkontextových pravidel a snížení složitosti navrhovaných gramatik.

Zmínili jsme známé bezkontextové metody syntaktické analýzy založené na deterministických zásobníkových automatech. Velkou míru pozornosti jsme věnovali odstranění nedeterminismu z CD gramatických systémů. Byly diskutovány algoritmy, které tento proces umožní do jisté míry automatizovat. Celá snaha vyústila v rozšíření LL tabulky do nové struktury – GLL tabulky, která simuluje chování gramatického systému. Pro tuto strukturu jsme sestrojili syntaktický analyzátor. Po zobecnění poznatků jsme došli k závěru, že je možné obdobným způsobem převádět i HCD gramatické systémy. Zavedli jsme si ukončující derivace, diskutovali jsme jejich využití v gramatických systémech a nastínili způsob jejich převodu do GLL tabulky. Průběžně jsme sestrojili GLL tabulky pro několik známých kontextových jazyků. Na závěr kapitoly věnující se zvýšení generativní síly jsme odhalili oblast využití uvedených metod a jejich limity.

Poté jsme přikročili k druhému dílčímu cíli, tedy ke snížení složitosti gramatik. Navrhli jsme gramatický systém propojující prediktivní syntaktickou analýzu s precedenční. Diskutovali jsme možnost využití více podřízených precedenčních gramatik kooperujících s LL gramatikou a uvedli jsme možnosti řízení těchto gramatik rozšířenou LL tabulkou.

V jazyce Python jsme implementovali uvedené algoritmy převodu, což vyústilo ve vznik modulu pro popis CD gramatických systémů a jejich převod na GLL tabulku. Vytvořený nástroj dokáže rovněž automatizovaně převádět LL gramatiky na LL tabulky. Posléze jsme sestrojili syntaktický analyzátor a s jeho užitím provedli analyzovali několik vět z různých kontextových jazyků.

Podařilo se nám rozšířit repertoár bezkontextových metod syntaktické analýzy a některé metody propojit. Přestože pořád zůstáváme v rovině bezkontextových matematických modelů, můžeme uvedenými metodami zpracovávat i vybrané kontextové jazyky.

Práce se pouze dotýká široké oblasti gramatických systémů, ve které je mnoho prostoru pro další bádání. Snaží se poskytnout základní přehled znalostí z této oblasti, či jít příkladem, v naději, že na ni bude navázáno. Především oblast PC gramatických systému se stává zajímavá s rozšiřujícími možnostmi paralelního zpracování.

Literatura

[1] Aho, A. V.: *Compilers, Principles, Techniques & Tools*, 2nd ed. Boston: Addison Wesley, 1994, ISBN 0-301-48681-1.

[2] Lucian, I., Salomaa, A.: *2-testability and relabelings produce everything*. Turku: Turku Centre for Computer Science, 1996, ISBN 9516507522.

[3] Meduna, A.: *Automata and languages: theory and applications*. London: Springer, 2000, ISBN 1-85233-074-0.

[4] Rozenberg, G., Salomaa, A., aj.: *Handbook of formal languages: background and application*. Berlin: Springer-Verlag, 1997, ISBN 3-540-60648-3.

[5] Rozenberg, G., Salomaa, A., aj.: *Handbook of formal languages: word, language, grammar*. Berlin: Springer-Verlag, 1997, ISBN 3-540-60420-0.

[6] Takita, B.: Is C++ context-free or context-sensitive? 2013.
Dostupné z: http://stackoverflow.com/questions/14589346/is-c-context-free-or-context-sensitive.

[7] Trevor, J.: Python is not context free. 2012.
Dostupné z: http://trevorjim.com/python-is-not-context-free.

[8] Černá, I., Kretínský, M., Kučera, A.: Automaty a formální jazyky I. 2002, verze 1.3.

Přílohy

Příloha A

Transformace kontextového jazyka

Uvažujme jazyk kontextový $L_\lambda = \{a^n b^m a^n b^m \mid n, m \geq 1\}$. Je dán gramatický systém

$$\begin{aligned}
\sigma_{16} &= (\{S, X, Y, Z, V\}, \{a, b\}, S, P_1, P_2, P_3, P_4, P_5, P_6, P_7), \\
P_1 &= \{S \to S,\ S \to aXbZaXbZ\}, \\
P_2 &= \{X \to aY\}, \\
P_3 &= \{Y \to aX\}, \\
P_4 &= \{X \to \varepsilon,\ Y \to \varepsilon\}, \\
P_5 &= \{Z \to bV\}, \\
P_6 &= \{V \to bZ\}, \\
P_7 &= \{Z \to \varepsilon,\ V \to \varepsilon\}.
\end{aligned}$$

Pro tento systém platí relace $L_{=2}(\sigma_{16}) = L_\lambda$. Sestrojíme množiny $First$ a $Follow^{CD}_{=2}$:

$$\begin{array}{lllllll}
First(S) & = & \{a\}, & Follow^{CD}_{=2}(S) & = & \{\$\}, \\
First(X) & = & \{a\}, & Follow^{CD}_{=2}(X) & = & \{b\}, \\
First(Y) & = & \{a\}, & Follow^{CD}_{=2}(Y) & = & \{b\}, \\
First(Z) & = & \{b\}, & Follow^{CD}_{=2}(Z) & = & \{a, \$\}, \\
First(V) & = & \{b\}, & Follow^{CD}_{=2}(V) & = & \{a, \$\}.
\end{array}$$

Uvažujme pravidla očíslovaná následujícím způsobem:

$$P = \left\{ \begin{array}{ll}
1: & S \to S, \\
2: & S \to aXbZaXbZ, \\
3: & X \to aY, \\
4: & Y \to aX, \\
5: & X \to \varepsilon,
\end{array} \quad \begin{array}{ll}
6: & Y \to \varepsilon, \\
7: & Z \to bV, \\
8: & V \to bZ, \\
9: & Z \to \varepsilon, \\
10: & V \to \varepsilon.
\end{array} \right\}$$

Na která aplikujeme transformační algoritmus:

$S \Rightarrow^{=2}_{P_1} aXbZaXbZ[\vec{S} \mid 1, 2]$:
$a \in First(\vec{S}) \wedge a \in First(aXbZaXbZ)$: $\alpha(a, \vec{S}) := [1, 2]$
$Empty(aXbZaXbZ) = \emptyset$
$\Omega = \{XZXZ\}$, $\Theta = \{S\}$

$XZXZ \Rightarrow^{=2}_{P_2} aYZaYZ[\overrightarrow{X} \mid 3,3]$:
$a \in First(\overrightarrow{X}) \wedge a \in First(aYZaYZ)$: $\alpha(a, \overrightarrow{X}) := [3,3]$
$Empty(aYZaYZ) = \emptyset$
$XZXZ \Rightarrow^{=2}_{P_5} XbVXbV[\overrightarrow{Z} \mid 7,7]$:
$b \in First(\overrightarrow{Z}) \wedge b \in First(XbVXbV)$: $\alpha(b, \overrightarrow{Z}) := [7,7]$
$Empty(XbVXbV) = \emptyset$
$XZXZ \Rightarrow^{=2}_{P_7} XX[\overrightarrow{Z} \mid 9,9]$:
$b \in First(Z)$, ale $b \notin First(XX)$
$Empty(XX) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(a, \overrightarrow{Z}) := [9,9]$
$Empty(XX) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(\$, \overrightarrow{Z}) := [9,9]$
$XZXZ \Rightarrow^{=2}_{P_4} ZZ[\overrightarrow{X} \mid 5,5]$:
$a \in First(X)$, ale $a \notin First(ZZ)$
$Empty(ZZ) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{X}$: $\alpha(b, \overrightarrow{X}) := [5,5]$
$\Omega = \{YZYZ, XVXV, XX, ZZ\}$, $\Theta = \{S, XZXZ\}$

$YZYZ \Rightarrow^{=2}_{P_3} aXZaXZ[\overrightarrow{Y} \mid 4,4]$:
$a \in First(\overrightarrow{Y}) \wedge a \in First(aXZaXZ)$: $\alpha(a, \overrightarrow{Y}) := [4,4]$
$Empty(aXZaXZ) = \emptyset$
$YZYZ \Rightarrow^{=2}_{P_5} YbVYbV[\overrightarrow{Z} \mid 7,7]$:
$b \in First(\overrightarrow{Z}) \wedge b \in First(YbVYbV)$: $\alpha(b, \overrightarrow{Z}) := [7,7]$
$Empty(YbVYbV) = \emptyset$
$YZYZ \Rightarrow^{=2}_{P_5} YY[\overrightarrow{Z} \mid 9,9]$:
$b \in First(Z)$, ale $b \notin First(YY)$
$Empty(YY) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(a, \overrightarrow{Z}) := [9,9]$
$Empty(YY) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(\$, \overrightarrow{Z}) := [9,9]$
$YZYZ \Rightarrow^{=2}_{P_4} ZZ[\overrightarrow{Y} \mid 6,6]$:
$a \in First(Y)$, ale $a \notin First(ZZ)$
$Empty(ZZ) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{Y}$: $\alpha(b, \overrightarrow{Y}) := [6,6]$
$\Omega = \{XVXV, YVYV, XX, YY, ZZ\}$, $\Theta = \{S, XZXZ, YZYZ\}$

$XVXV \Rightarrow^{=2}_{P_3} aYVaYV[\overrightarrow{X} \mid 3,3]$:
$a \in First(\overrightarrow{X}) \wedge a \in First(aYVaYV)$: $\alpha(a, \overrightarrow{X}) := [3,3]$
$Empty(aYVaYV) = \emptyset$
$XVXV \Rightarrow^{=2}_{P_6} XbZXbZ[\overrightarrow{V} \mid 8,8]$:
$b \in First(\overrightarrow{V}) \wedge b \in First(XbZXbZ)$: $\alpha(b, \overrightarrow{V}) := [8,8]$
$Empty(XbZXbZ) = \emptyset$
$XVXV \Rightarrow^{=2}_{P_7} XX[\overrightarrow{V} \mid 9,9]$:
$b \in First(V)$, ale $b \notin First(XX)$
$Empty(XX) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(a, \overrightarrow{V}) := [9,9]$
$Empty(XX) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(\$, \overrightarrow{V}) := [9,9]$
$XVXV \Rightarrow^{=2}_{P_4} VV[\overrightarrow{X} \mid 6,6]$:
$a \in First(X)$, ale $a \notin First(VV)$
$Empty(VV) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{X}$: $\alpha(b, \overrightarrow{X}) := [6,6]$
$\Omega = \{YVYV, XX, YY, ZZ, VV\}$, $\Theta = \{S, XZXZ, YZYZ, XVXV\}$

$YVYV \Rightarrow^{=2}_{P_2} aXVaXV[\overrightarrow{Y} \mid 4,4]$:

$a \in First(\overrightarrow{Y}) \wedge a \in First(aXVaXV)$: $\alpha(a, \overrightarrow{Y}) := [4, 4]$

$Empty(aXVaXV) = \emptyset$

$YVYV \Rightarrow^{=2}_{P_6} YbZYbZ[\overrightarrow{V} \mid 8, 8]$:

$b \in First(\overrightarrow{V}) \wedge b \in First(YbZYbZ)$: $\alpha(b, \overrightarrow{V}) := [8, 8]$

$Empty(YbZYbZ) = \emptyset$

$YVYV \Rightarrow^{=2}_{P_7} YY[\overrightarrow{V} \mid 10, 10]$:

$b \in First(V)$, ale $b \notin First(YY)$

$Empty(YY) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(a, \overrightarrow{V}) := [10, 10]$

$Empty(YY) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(\$, \overrightarrow{V}) := [10, 10]$

$YVYV \Rightarrow^{=2}_{P_4} VV[\overrightarrow{Y} \mid 6, 6]$:

$a \in First(Y)$, ale $a \notin First(VV)$

$Empty(VV) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{Y}$: $\alpha(b, \overrightarrow{Y}) := [6, 6]$

$\Omega = \{XX, YY, ZZ, VV\}$, $\Theta = \{S, XZXZ, YZYZ, XVXV, YVYV\}$

$XX \Rightarrow^{=2}_{P_2} aYaY[\overrightarrow{X} \mid 3, 3]$:

$a \in First(\overrightarrow{X}) \wedge a \in First(aYaY)$: $\alpha(a, \overrightarrow{X}) := [3, 3]$

$Empty(aYaY) = \emptyset$

$XX \Rightarrow^{=2}_{P_4} \varepsilon[\overrightarrow{X} \mid 5, 5]$:

$a \in First(X)$, ale $a \notin First(\varepsilon)$

$Empty(\varepsilon) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{X}$: $\alpha(b, \overrightarrow{X}) := [5, 5]$

$\Omega = \{YY, ZZ, VV\}$, $\Theta = \{S, XZXZ, YZYZ, XVXV, YVYV, XX\}$

$YY \Rightarrow^{=2}_{P_3} aXaX[\overrightarrow{Y} \mid 4, 4]$:

$a \in First(\overrightarrow{Y}) \wedge a \in First(aXaX)$: $\alpha(a, \overrightarrow{Y}) := [4, 4]$

$Empty(aXaX) = \emptyset$

$YY \Rightarrow^{=2}_{P_4} \varepsilon[\overrightarrow{Y} \mid 6, 6]$:

$a \in First(Y)$, ale $a \notin First(\varepsilon)$

$Empty(\varepsilon) = \{\varepsilon\} \wedge b \in Follow^{CD}_{=2}\overrightarrow{Y}$: $\alpha(b, \overrightarrow{Y}) := [6, 6]$

$\Omega = \{ZZ, VV\}$, $\Theta = \{S, XZXZ, YZYZ, XVXV, YVYV, XX, YY\}$

$ZZ \Rightarrow^{=2}_{P_5} bVbV[\overrightarrow{Z} \mid 7, 7]$:

$b \in First(\overrightarrow{Z}) \wedge b \in First(bVbV)$: $\alpha(b, \overrightarrow{Z}) := [7, 7]$

$Empty(bVbV) = \emptyset$

$ZZ \Rightarrow^{=2}_{P_7} \varepsilon[\overrightarrow{Z} \mid 9, 9]$:

$b \in First(Z)$, ale $b \notin First(\varepsilon)$

$Empty(\varepsilon) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(a, \overrightarrow{Z}) := [9, 9]$

$Empty(\varepsilon) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{Z}$: $\alpha(\$, \overrightarrow{Z}) := [9, 9]$

$\Omega = \{VV\}$, $\Theta = \{S, XZXZ, YZYZ, XVXV, YVYV, XX, YY, ZZ\}$

$VV \Rightarrow^{=2}_{P_6} bZbZ[\overrightarrow{V} \mid 8, 8]$:

$b \in First(\overrightarrow{V}) \wedge b \in First(bZbZ)$: $\alpha(b, \overrightarrow{V}) := [8, 8]$

$Empty(bZbZ) = \emptyset$

$VV \Rightarrow^{=2}_{P_7} \varepsilon[\overrightarrow{V} \mid 10, 10]$:

$b \in First(V)$, ale $b \notin First(\varepsilon)$

$Empty(\varepsilon) = \{\varepsilon\} \wedge a \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(a, \overrightarrow{V}) := [10, 10]$

$Empty(\varepsilon) = \{\varepsilon\} \wedge \$ \in Follow^{CD}_{=2}\overrightarrow{V}$: $\alpha(\$, \overrightarrow{V}) := [10, 10]$

$\Omega = \emptyset$, $\Theta = \{S, XZXZ, YZYZ, XVXV, YVYV, XX, YY, ZZ, VV\}$

Transformace systému σ_{16} je už mírně delší, než předchozí uvedené příklady, ale algoritmicky pořád dobře zvládnutelné. Tabulka A.0.1 zobrazuje výslednou podobu GLL tabulky.

N ╲ T	**a**	**b**	**$**
S	1,2		
X	3,3	5,5	
Y	4,4	6,6	
Z	9,9	7,7	9,9
V	10,10	8,8	10,10

Tabulka A.0.1 GLL tabulka pro jazyk $L_\lambda = \{a^n b^m a^n b^m \mid n, m \geq 1\}$.

U systému lze vypozorovat jistou zajímavost. Systém generuje jazyk $\{a^n b^m a^n b^m\}$ pro libovolné $n, m \geq 1$, tj. nejkratší odvoditelný řetězec je „$abab$“, nelze odvodit ani „aa“ ani „bb“, nebo dokonce ε. Tohle omezení hraje významnou roli v syntaktické analýze s použitím deterministického zásobníkového automatu. Řetězec o nejmenší možné délce 4 je garantován použitím pravidla $S \to aXbZaXbZ$ komponenty P_1. Vyvstává otázka, zdali nelze využít pravidlo $S \to XZXZ$. Poté by platilo $n, m \geq 0$, takže rozšíříme množinu přijímaných řetězců. Na první pohled se to může zdát jako rozumné řešení, ale uvažujme situaci, kdy vstupní věta neobsahuje žádný symbol „a“. V takovém případě systém provede posloupnost derivací

$$S \Rightarrow_{P_1}^{=2} XZXZ \Rightarrow_{P_4}^{=2} ZZ.$$

Na zásobníku nám zbude ZZ, tedy budeme přijímat řetězec z jazyka $\overline{L_\lambda} = \{b^m b^m\}$, přičemž platí $\overline{L_\lambda} \subset L_\lambda$ pro $n, m \geq 0$. Jazyk $\overline{L_\lambda}$ vykazuje stejné chování jako jazyk ww, tedy replikace, a skutečně $\overline{L_\lambda}$ patří do třídy nedeterministických jazyků. Přestože tento jazyk lze generovat gramatickým systémem, nelze jej přijímat GLL analyzátorem. Jednoduše řečeno zobecněním $n, m \geq 0$ se dostáváme z oblasti kontextových jazyků zpracovatelných pomocí deterministického zásobníkového automatu, tedy i GLL analyzátoru.

Daný jazyk lze rozšířit tak, aby platilo $n \geq 0 \wedge m \geq 1$ nebo $n \geq 1 \wedge m \geq 0$ a zároveň zachovat determinismus. Stačí si vybrat jeden terminál, který bude fungovat jako zarážka a druhý může být vymazán. Uvažujeme-li první uvedenou variantu, pak je možné X vymazat a je garantován pouze minimální počet terminálů „b“. Tohoto efektu jednoduše docílíme záměnou druhého pravidla komponenty P_1 za pravidlo $S \to XbZXbZ$.

Příloha B

Obsah přiloženého paměťového media

Přiložené paměťové médium obsahuje zdrojové texty práce, instalační pokyny a zdrojové texty aplikace v jazyce Python, která implementuje uvedené principy. Součástí paměťového media jsou soubory a adresáře:

- `readme.txt` – instalační pokyny a návod k obsluze aplikace,
- `readme.pdf` – instalační pokyny a návod k obsluze aplikace ve formátu PDF,
- `thesis` – adresář s písemnou zprávou ve formátu PDF,
- `thesis_src` – adresář se zdrojovými texty písemné zprávy pro systém LaTeX,
- `src` – adresář se zdrojovými kódy aplikace v jazyce Python:
  - `grammarSystem.py` – implementace tříd pro popis a transformaci gramatických systémů,
  - `derivation.py` – implementace vícekrokových derivací a vytvoření množiny Φ,
  - `gllParser.py` – implementace GLL syntaktického analyzátoru a lexikálního analyzátoru založeného na regulárních výrazech,
  - `systemRepository.py` – ukázka definic několika gramatických systémů v jazyce Python s podporou modulu `grammarSystem.py`,
  - `gsPresenter.py` – grafické uživatelské rozhraní pro zobrazení gramatických systémů a simulování přijímaní řetězců,
  - `gsPresenterDesign.py` – definice vhledu grafického uživatelské rozhraní využitého v modulu `gsPresenter.py`.