VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

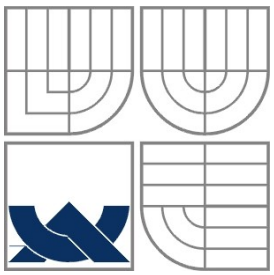
MULTIGRAMATIKY A SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA NICH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

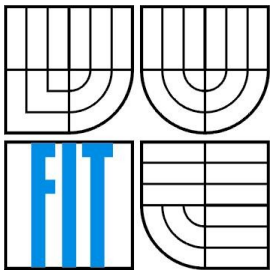
AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ KRAJÍČEK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MULTIGRAMATIKY A SYNTAKTICKÁ ANALÝZA ZALOŽENÁ NA NICH

MULTIGRAMMARS AND PARSING BASED ON THEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ KRAJÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. ALEXANDER MEDUNA

BRNO 2007

ZADÁNÍ

1. Seznamte se podrobně s multigramatikami a jejich vlastnostmi. Dle instrukcí vedoucího se seznamte detailně s řadou metod syntaktické analýzy.
2. Navrhněte metodu syntaktické analýzy založenou na multigramatikách. Porovnejte ji s vlastnostmi jiných metod syntaktické analýzy; diskutujte přednosti a nedostatky.
3. Navrhněte a implementujte program, který bude demonstrovat navrženou metodu v oblasti syntaktické analýzy přirozených jazyků. Testujte výsledný program.
4. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

LICENČNÍ SMLOUVA

Viz odkaz https://www.fit.vutbr.cz/info/szz/SmR-9-priloha3.pdf

Abstrakt

Úkolem této práce je seznámení s pragmaticky orientovaným výzkumem na poli teoretické informatiky a prezentovat návrhy aplikačních metod pro zvolené tematické oblasti. Konkrétní prostředek teoretické informatiky je reprezentován druhem generativního systému – multigramatikou a jednotlivé aplikační oblasti jsou zde zvoleny vzhledem k možnostem, které multigramatiky nabízí.

V návaznosti na poznatky dosažené Thompsonem (viz [9]), Lindenmayerem (viz [26]), Mandelbrotem (viz [8]) ale i výsledky ze studií dosažených Morneauem (viz [17]), které poukazují na souvislosti mezi přírodními zákonitostmi a matematickou disciplínou, zkoumáme aplikace multigramatik z pohledu dvou tematických oblastí: generativních L-systémů (zahrnuje dále aplikace z fraktálové grafiky a biomatematiky) a zpracování jazyků přirozených (zahrnuje dále návrh vhodného abstraktního jazyka). Zmíněny jsou také otázky společné s oblastí návrhu překladačů.

Klíčová slova

Multigramatika, rozšířená multigramatika, selektor, L-systém, selektivní L-systém, fraktál, želví geometrie, biomatematika, zpracování přirozených jazyků, syntaktická analýza, chart parser, token, lexém, morfém, stavba slov, závislostní struktura, frázová struktura, nejednoznačná gramatika, jazyky s jemným kontextem.

Abstract

This document deals with introduction focused on pragmatically oriented research at branch of theoretical computer science and with presentation of designed methods for chosen application topics.
At this study the theoretical subject is represented by kind of generative system – multisequential grammar and application topics are chosen according to possibilities supported by multisequential grammars.

In order to follow results published by Thompson (see [9]), Lindenmayer (see [26]), Mandelbrot (see [8]) and also studies published by Morneau (see [17]), which shows the relation between natural laws and human discipline – mathematics, we study the applications of multi-sequential grammars from two points of view: generative L-systems (which further includes applications of fractal geometry and biomathematics) and natural language processing (which further includes the design of proper abstract language). Some problems related to compiler construction are also mentioned.

Keywords

Multi-sequential grammar, multi-continues grammar, selector, L-system, selective L-system, fractal, turtle geometry, biomathematics, natural language processing, syntax analyse - parsing, chart parser, token, lexeme, morpheme, word syntax, dependency structure, phrase structure, ambiguity grammar, mildly context sensitive languages.

Citace

Jiří Krajíček: Multigramatiky a syntaktická analýza založená na nich, diplomová práce, Brno,
FIT VUT v Brně, 2007

Multigramatiky a syntaktická analýza založená na nich

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Prof. RNDr. Alexandra Meduny, CSc. Další informace mi poskytli Prof. Ing. Ivo Serba, CSc, Bc. Dana Lodrová, Bc. Petr Mikušek, Bc. Anh Le Hai. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Krajíček
10. 5. 2007

Poděkování

Tímto děkuji svému vedoucímu diplomové práce panu Prof. Alexandru Medunovi, Csc. za cenné rady a obětavý přístup, který mi věnoval i v rámci této práce.
Dále děkuji za štědrou podporu poskytnutou nadací „*Nadání Josefa, Marie a Zdeňky Hlávkových*“.

Obsah

1 Úvod

Teoretická informatika je prostředek k dosažení určitého cíle, tento cíl chápeme jako aplikaci. Samotná aplikace může být prezentována v různé rovině abstrakce, od čistě abstraktní až po zcela konkrétní. V jednom z případů pak můžeme za aplikaci považovat situaci, kdy pomocí prostředků teoretické informatiky rozvíjíme teoretickou informatiku samotnou např. konstrukcí důkazů nad otevřenými problémy této disciplíny. Na druhé straně existují také konkrétnější aplikace vyjádřeny problémy reálného světa, kde klademe důraz na praktické využití dosažených výsledků.

Tato práce se celkově zabývá souvislostmi mezi aparátem teoretické informatiky a jejich spojením v kontextu s popisem problémů na různých rovinách abstrakce – aplikacemi. Konkrétní prostředek teoretické informatiky je zde zastoupen specifických druhem generativního systému – multigramatikou a jednotlivé aplikační oblasti jsou pak zvoleny vzhledem k možnostem, které využití multigramatik nabízí. Pro snazší představu bychom mohli o této práci také hovořit pod označením: „*Multigramatiky a jejich aplikace*“. Vlastní aplikace jsou zde zastoupeny zejména dvěma základními tematickými oblastmi:

- generativní L-systémy (zahrnuje dále aplikace z fraktálové grafiky a biomatematiky),
- zpracování jazyků přirozených (zahrnuje dále návrh vhodného abstraktního jazyka).

Každá tematická oblast je nejprve popsána na obecné úrovni a jsou zmíněny metody, které jsou v ní využity. Následuje pak zavedení aparátu multigramatik do dané oblasti a demonstrace získaných výsledků. Obsahově je v 2. kapitole zmíněna teorie muligramatik a jejich příslušnost ke třídě jazyků. V následné 3. kapitole pak zmiňujeme vybrané tematické oblasti a jejich souvislosti s multigramatikami. Kapitola 4. již popisuje první tematickou oblast generativních L-systémů podrobněji a zavádí nad ní aplikace multigramatik. Podobně tak postupujeme i v nadcházející kapitole 5., kde se zabýváme zmíněnou tématickou oblastí zpracování přirozených jazyků a využitím multigramatik zde zejména jako prostředku ve fázi syntaktické analýzy. Každá z popisných kapitol pak obsahuje v závěru svá zhodnocení dosažených výsledků a porovnání s ostatními dosud užívanými přístupy. V poslední 6. kapitole jsou zmíněny implementační aspekty a výstupy softwarových produktů, které byly navrženy pro potřeby demonstrace dosažených výsledků v každé tematické oblasti (jedná se zejména o zpracování přirozených jazyků a využití selektivních L-systémů).

V návaznosti na semestrální projekt zde došlo k rozšíření kapitol 1. – 5. o nové poznatky a 6. kapitola byla sestavena na základě vývoje a dokumentace demonstračního software pro aplikace.

2 Teorie multigramatik

V oblasti teoretické informatiky multigramatiky představují druh paralelního generativního systému, který byl zaveden G. Rozenbergem v první polovině osmdesátých let minulého století (viz [1]).

Cílem multigramatik je snaha o zvýšení vyjadřovací síly při zachování jednoduchého tvaru bezkontextových pravidel v paralelním přepisování. Z pohledu průběhu derivace lze tyto gramatiky považovat také za případ tzv. gramatiky řízených. Podrobněji princip multigramatik uvedeme v následujících definicích a posléze demonstrujeme některé příklady.

2.1 Definice multigramatik

Definice 2.1

Multigramatika G je pětice $G = (V, T, P, S, K)$, kde V, T a S mají stejný význam jako v obecné gramatice. P je konečná množina pravidel tvaru:

$$a \rightarrow x, \text{ kde } a \in V \text{ a } x \in V^*.$$

K je konečná množina selektorů tvaru:

$$X_1\textbf{\textit{active}}(Y_1)\, X_2 \ldots X_n\, \textbf{\textit{active}}(Y_n)\, X_{n+1};$$

kde n je kladné celé číslo,

pro všechna $i = 1, \ldots, n+1$, $X_i \in \{Z^* : Z \subseteq \mathrm{V}\}$,

pro všechna $j = 1, \ldots, n$, $Y_j \subseteq V$ a $Y_j \neq \emptyset$.

G provádí *derivační krok* ve tvaru:

$$u_1 a_1 u_2 a_2 u_3 \ldots u_n a_n u_{n+1} \Rightarrow u_1 x_1 u_2 x_2 u_3 \ldots u_n x_n u_{n+1}$$

jestliže K obsahuje selektor $X_1\textbf{\textit{active}}(Y_1)\, X_2 \ldots X_n\, \textbf{\textit{active}}(Y_n)\, X_{n+1}$ splňující:

pro všechna $i = 1, \ldots, n+1$ $u_i \in X_i$

pro všechna $j = 1, \ldots, n$, $a_j \in Y_j$ a $a_j \rightarrow x_j \in P$

Jazyk generovaný G, $L(G)$, je definován $L(G) = \{ w: S \Rightarrow^* w \text{ a } w \in T^* \}$.

Definice 2.2

Rozšířená multigramatika G je pětice $G = (V, T, P, S, K)$,

kde V, T, P a S mají stejný význam jako v *multigramatice*.

K je konečná množina selektorů tvaru:

$$X_1\textbf{\textit{active}}(Y_1)\, X_2 \ldots X_n\, \textbf{\textit{active}}(Y_n)\, X_{n+1},$$

kde n je kladné celé číslo,

pro všechna $i = 1, ..., n + 1$, $X_i \in \{Z^* : Z \subseteq V\}$

pro všechna $j = 1, ..., n$, $Y_j \in \{ Z^+ : Z \subseteq V$ a $Z \neq \varnothing\}$.

Pro každý řetězec $v \in V^+$, kde $v = a_1...a_{|v|}$ s $a_i \in V$ pro $i = 1, ..., |v|$, definujeme jazyk

ContinuousRewriting$(v) \subseteq V^*$ následující ekvivalencí pro každé $z \in V^*$,

$$z \in \textit{ContinuousRewriting}(v)$$

tehdy a jen tehdy $a_i \rightarrow x_i \in P$ pro všechna $i = 1, ..., |v|$, a $z = x_1...x_{|v|}$.

G provádí derivační krok tvaru:

$$u_1y_1u_2y_2\,u_3 \ldots u_ny_nu_{n+1} \Rightarrow u_1z_1u_2z_2\,u_3 \ldots u_nz_nu_{n+1};$$

jestliže *K* obsahuje $X_1\textbf{\textit{active}}(Y_1)\,X_2 \ldots X_n\,\textbf{\textit{active}}(Y_n)\,X_{n+1}$ takové, že

pro všechna $i = 1, ..., n$, $y_i \in Y_i$ a $z_i \in$ *ContinuosRewriting*(y_i)

pro všechna $i = 1, ..., n + 1$, $u_i \in X_i$.

Jazyk generovaný *G*, *L*(*G*), je definován $L(G) = \{ w: S \Rightarrow^* w$ a $w \in T^*\}$.

Jak plyne z definic 2.1 a 2.2, *rozšířené multigramatiky* nám na rozdíl od multigramatik poskytují možnost vyjádřit aktivní část selektoru regulární množinou, tedy množinou řetězců, které lze popsat regulárním výrazem. Přičemž pro každý symbol z tohoto řetězce musí existovat pravidlo, jež obsahuje tento symbol na své levé straně (podmínka z jazyka *ContinuousRewriting*).

2.2 Příslušnost ke třídě jazyků

Lze dokázat, že pro každý jazyk přijímaný Turingovým strojem lze sestrojit multigramatika (rozšířená multigramatika), která tento jazyk generuje a naopak pro každý jazyk generovaný multigramatikou (rozšířenou multigramatikou) lze sestrojit Turingův stroj, který tento jazyk přijímá. Třída jazyků generovaných multigramatikami tedy odpovídá třídě jazyků rekurzivně vyčíslitelných (třída 0) dle Chomského klasifikace jazyků.

Lemma 2.1

Pro každou gramatiku frázové struktury *H*, existuje ekvivalentní multigramatika *M*, respektive rozšířená multigramatika *M'*.

Z pohledu teoretické informatiky je dále podstatná redukce složitosti v popisu multigramatiky při zachování síly s Turingovým strojem. Z tohoto důvodu jsou zaváděna další omezení multigramatik na počet neterminálů, počet a tvar selektorů.

Touto redukcí se zabýval zejména Prof. Meduna ve svých několika odborných článcích. Uveďme si jen přehledově podstatné poznatky, které z nich plynou.

1. Je dokázáno, že multigramatiky s 6 neterminály jsou ekvivalentní s Turingovým strojem (viz [2]).
2. Je dokázáno, že rozšířené multigramatiky s 6 neterminály jsou ekvivalentní s Turingovým strojem (viz [3]).
3. Je dokázáno, že multigramatiky se 2 homogenními selektory, kde každý selektor má jen 2 aktivní části jsou ekvivalentní s Turingovým strojem (viz [4]).
4. Dále je dokázáno, že rozšířené multigramatiky s 5 homogenními selektory (také 5 neterminálů), kde každý selektor má jen 2 aktivní části jsou ekvivalentní s Turingovým strojem (viz [5]).

Pozn.: homogenní selektor je takový selektor, jehož všechny aktivní části jsou identické (jsou popsány stejnou regulární množinou, respektive regulárním výrazem).

2.3 Příklady

Pozn.: namísto popisu selektoru prostřednictvím regulárních množin, budeme selektory popisovat z pohledu pragmatického přístupu (parseru) skrze regulární výrazy, ostatní notace bude zachována.

Příklad 2.1

Generujme multigramatikou jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$, vidíme, že platí: $L \notin$ CFL, ale $L \in$ CSL.
Sestrojíme multigramatiku $G = (V, T, P, S, K)$, kde

$V = \{a, b, c\}$,
$T = V$,
$P = \{$ 1: $S \rightarrow abc$,
2: $S \rightarrow \varepsilon$,
3: $a \rightarrow aa$,
4: $b \rightarrow bb$,
5: $c \rightarrow cc$ $\}$
K = {*active*(***S***), *a** *active*(***a***)*b** *active*(***b***)*c** *active*(***c***)}.

Ukázka paralelní derivace v G pro generování $a^3b^3c^3$:

$$S \Rightarrow abc\ [1] \Rightarrow aabbcc\ [3, 4, 5] \Rightarrow aaabbbccc\ [3, 4, 5].$$

Příklad 2.2

Generujme rozšířenou multigramatikou jazyk $L = \{a^n\ b^n\ c^n \mid n = 2(i - 1)$, kde i = *počet provedených derivačních kroků*$\}$, opět vidíme, že platí: $L \notin$ CFL, ale $L \in$ CSL.

Sestrojíme multigramatiku $G' = (V, T, P, S, K)$, kde

$V = \{a, b, c\}$,

$T = V$,

$P = \{$ 1: $S \rightarrow abc$,

2: $a \rightarrow aa$,

3: $b \rightarrow bb$,

4: $c \rightarrow cc$ $\}$

$K = \{active(\boldsymbol{S}), active(\boldsymbol{a}+)\ active(\boldsymbol{b}+)\ active(\boldsymbol{c}+)\}$.

Ukázka paralelní derivace v G' pro $i = 3$, tedy $a^4b^4c^4$:

$S \Rightarrow abc$ [1] $\Rightarrow aabbcc$ [2, 3, 4] $\Rightarrow aaaabbbbcccc$ [2, 2, 3, 3, 4, 4].

3 Tematické oblasti pro aplikace multi-gramatik

3.1 Motivace

Naše okolí nás obklopuje řadou problémů, které zkoumáme a jejichž řešení bychom rádi znali. Proto vynakládáme při dostupných znalostech značná úsilí k jejich rozřešení. Na vyšší (abstraktní) úrovni je problém zpravidla formálně popsán a v okamžiku, kdy se nám podaří (například při užití matematického aparátu) řešení nalézt a podložit jej (dokázat), lze přistoupit k vlastní aplikaci teorie, tak abychom poskytli nástroj na problém původní, který stál na samém počátku našeho úsilí. Těžko si lze tedy představit svět, kde bychom při snaze řešit problém nepřistoupili také k vlastní aplikaci (potírání problémů reálného světa). Z tohoto důvodu zavádíme tzv. *vlastní aplikace teorií*, ve snaze zavést řešení na problémy, které nás obklopují. Teorie je zde pomocníkem, nástrojem, jehož prostředky chceme využít. Z pohledu teorie zde však není cesta k teorii, teorie je cesta, ale není to cíl, nýbrž jen prostředek k dosažení cíle.

Pozn.: z filozofického hlediska lze považovat pojem „*řešení problémů*“ za skutečnost, která již existuje, není ji tedy třeba vynalézat, ale objevovat. Jinak řečeno, lze přepokládat, že odpovědi na otázky, které si v souvislosti s řešením problémů pokládáme, nás obklopují. Hranice mezi problémem a jeho řešením je tvořena jen vlastním omezením umět „*pozorovat*“ prostředí kolem nás. Je tedy nejprve nutné dokázat chápat okolní podměty a příčiny i důsledky všech dějů. Přičemž proces správného pochopení je také závislí na stavu, rozpoložení našeho vnímání. Abychom zachovali tyto předpoklady, buďme vždy upřímní zejména sami k sobě a skromní k ostatním („*always be honest to your self and humble to the others*“). Dokud nebudeme schopni „*správně*“ porozumět své podstatě, svému okolí (a to nejen z fyzického úhlu pohledu), lze jen těžko předpokládat, že se nám v komplexním kontextu dostane vhodné odpovědi.

3.2 Výběr tematických oblastí a jejich popis

Z obecného pohledu nachází generativní systémy uplatnění v řadě odborných oblastí. Jedná se zejména o výstavbu překladačů, příbuznou oblast zpracování jazyků (v obecnějším slova smyslu), počítačové grafiky, biomatematiky a dalších oborů, které například vyžadují požadavky na formální modely pro návrh, analýzu a specifikaci.

Silný vyjadřovací prostředek ze skupiny generativních systémů představují právě uvedené multigramatiky. Neformálně můžeme říci, že výhoda multigramatik vychází z předpokladu *jednoduchých* bezkontextových pravidel doplněných o sekundární formální řídící prostředek pro průběh derivace a v konečném důsledku schopnost generovat jazyky přijímané Turingovým strojem. Kombinují se zde výhody tzv. *řízených gramatik* (viz [6]) se systémy tzv. *paralelních gramatik* (viz [7]). Podrobnosti k těmto otázkám jsou diskutovány také v kapitole 5.2.3 část V. Na základě zde zmíněných vlastností a také z důvodu omezeného rozsahu této práce byly zvoleny tyto aplikace multigramatik:

- **L-systémy** (zavedení tzv. *selektivních L-systémů*), viz kapitola 4.
- **zpracování přirozených jazyků** (aplikace v syntaktické rovině), viz kapitola 5.

L-systémy mající využití v řadě oblastí včetně výtvarného umění – fraktálové grafiky[1] slouží také ke generování biologických struktur. V případě zpracování přirozených jazyků klademe důraz na rozpoznání syntaktické (popř. sémantické) správnosti vstupních vět (zpravidla ve formě elektronického textu) a celkový proces porozumění jazyku skrze strojové zpracování.

Rozsahově se v kapitole 4. se zabýváme popisem stávajících L-systémů a následně návrhem aplikace multigramatik - zavedení *selektivních L-systémů*. Podobně v kapitole 5. se zabýváme současnými přístupy pro zpracování jazyků přirozených a navazujeme návrhem vlastních metod při zaměření aplikace multigramatik na rovině syntaktického zpracování.

K demonstraci selektivního L-systému a analyzátoru větného kontextu na principu multigramatik, byly také sestrojeny demonstrační aplikace. Návrhové a implementační poznatky těchto softwarových produktů včetně ukázek výsledných výstupů jsou zmíněny v kapitole 6.

---

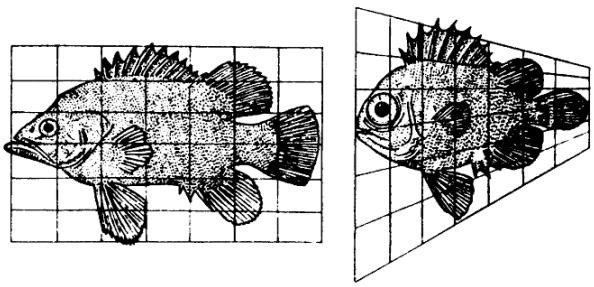
[1]) zakladatelem fraktálové grafiky je matematik Benoît B. Mandelbrot. Současný pojem fraktál je znám přibližně od roku 1975. Intuitivně řečeno, jedná se o matematické funkce pro popis obrazců vyplňujících prostor množením svých elementárních složek. Již dříve byly tyto obrazce v přírodě pozorovány, ovšem až s příchodem 20. století jsme byli schopni porozumět jejich matematickým zákonitostem.
Podrobnosti k těmto otázkám viz Mandelbrotova kniha *„The Fractal Geometry of Nature“* (viz [8]).

3.3 Přístup a vztah k vybraným oblastem

V motivační části 3.1 této kapitoly jsme zmínili některé nezbytné otázky pro poznání zákonů, které nás obklopují, tak abychom byli schopni nalézt vhodná řešení k existujícím problémům a zároveň dokázat tyto poznatky aplikovat. Nyní se budeme zabývat vztahy mezi uvedenými multigramatikami a vybranými tematickými oblastmi pro jejich aplikace.

Často až s dlouhým odstupem času jsme schopni chápat (přiznat) hloubku například matematických zákonitostí, které se v přírodních jevech nacházejí, ačkoli jsou tyto zákonitosti přítomny i bez našeho zásahu, tedy bez ohledu na náš dosavadní způsob jejich chápání. Jedním z matematiků, kteří poukazovali na souvislosti mezi přírodními jevy a lidskou disciplínou - matematikou byl biolog a matematik Prof. D'Arcy Thompson, který se tak stal zakladatelem nového oboru tzv. „*biomatematiky*". Ve své knize „*On Growth and Form*", (první vydání již v roce 1917, viz [9]) dokázal například souvislosti mezi matematickými transformacemi a proměnami napříč mezitřídními živočišnými druhy tedy, že rozmanitost tvarů je možné redukovat použitím transformací v různých geometriích; obr. 3.1.

$$u = (a_{10} + a_{11}x + a_{12}y)/(a_{00} + a_{01}x + a_{02}y)$$
$$v = (a_{20} + a_{21}x + a_{22}y)/(a_{00} + a_{01}x + a_{02}y)$$



Obr. 3.1 – ilustrace z Thompsonovy knihy, redukce tvarů transformace projektivní geometrie

Podobně jako Thompson, poukázal na souvislosti mezi přírodními jevy a matematickými zákony o řadu let později (1968) biolog Aristid Lindenmayer nalezením matematického formalismu (nazvaného Lindenmayerův systém: L-systém) pro popis progresí růstu (i degenerací) biologických struktur. Až později byl L-systém posouzen z pohledu teoretické informatiky a zařazen mezi gramatické systémy. Z tohoto pohledu jsme schopni uvažovat o možnostech aplikace multigramatik v L-systémech k dosažení (rozšíření) nových prostředků pro generační vývoj například zmiňovaných bio-struktur.

V případě druhé tematické oblasti – zpracováním přirozených jazyků pak uvažujeme o souvislostech mezi formálním popisem jazyka (zpravidla určitou generativní gramatikou) zejména na rovině syntaktického popisu ve snaze sestavit (posílit) prostředky pro strojový deterministický analyzátor jazyka prostřednictvím multigramatik. Zároveň tak využíváme vyjadřovací síly multigramatik k postihnutí celé rodiny jazyků přirozených (abstraktního návrhu nad jazyky), která by bylo s běžnými prostředky bezkontextových gramatik nedosažitelné.

4 Tematická oblast nad L-systémy

4.1 Popis současných L-systémů

V předchozí kapitole 3.2 již byl nastíněn původ L-systémů. Nyní si rozšíříme některé další podstatné souvislosti. Biolog a botanik Lindenmayer se původně zabýval růstem vzorů různých druhů řas jako je například bakterie *Anabaena catenula*. Lindenmayerův formální systém byl tedy nejprve využit k popisu prostých vícebuněčných organismů a vyjadřoval vztahy mezi sousedními rostlinnými buňkami. Později byl systém upraven (rozšířen) a využit i pro popis složitějších biologických struktur jako jsou například rostliny vyšších druhů a jiných větvení schopných forem.

Následně popíšeme L-systém formálně a jeho vlastnosti budeme demonstrovat na několika známých příkladech generovaní matematických struktur. V základní formě je L-systémů vyjádřen jen prostřednictvím abecedy symbolů, startovacího axiomu (řetězce) a množiny pravidel, viz definice 4.1.

Definice 4.1

0L-systém je trojice $H = (V, P, w)$, kde V je konečná abeceda symbolů, P je konečná množina pravidel tvaru: $a \rightarrow x$, $a \in V$ a $x \in V^*$ a $w \in V^*$ je startující řetězec (axiom).

Během jednoho derivačního kroku jsou v L-systému přepsány paralelně všechny symboly, které jsou totožné se symbolem na levé straně užitého pravidla. Jedná se tedy o druh paralelních gramatik.

Definice 4.2

Přímá derivace ($\Rightarrow$): $a_1a_2 \ldots a_n \Rightarrow x_1x_2 \ldots x_n$, $n \geq 1$, jestliže $a_i \rightarrow x_i \in P$ pro všechna $i = 1, \ldots, n$. Derivace ($\Rightarrow^*$): reflexivní a tranzitivní uzávěr nad $\Rightarrow$.

Pokud uvážíme existenci jen jednoho pravidla pro každý symbol z abecedy, získáme tzv. deterministický L-systém (D0L).

Definice 4.3

Deterministický 0L systém (D0L): Pro každé $a \in V$, existuje jedno pravidlo tvaru: $a \rightarrow x \in P$.

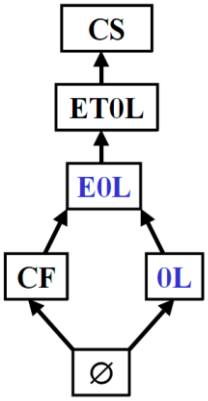
Pokud dále uvážíme případ, kdy nepožadujeme přítomnost axiomu v jazyce generového pomocí L-systému a chceme oddělit z abecedy symbolů množinu terminálů, dostáváme tak tzv. *rozšířený L-systém* (E0L).

Definice 4.4

Rozšířený 0L systém (E0L) je čtveřice G = (V, T, P, w), kde V, P, w mají identický význam jako v případě 0L systému a $T \subseteq V$.

Zatímco 0L systémy označujeme také jako tzv. bezkontextové L-systémy (schopnost generovat také jazyky bezkontextové), E0L mají schopnost generovat i jazyky kontextové. Formálně platí tento vztah: $L(0L) \subset L(E0L)$.

Zároveň L-systém je druhem gramatik, které generují jazyky vymykající se klasické Chomského hierarchii. Z tohoto pohledu lze uvažovat o pokrytí tříd v této hierarchii skrze L-systémy, viz obr. 4.1.



Obr. 4.1 – jazyky L-systému v kontextu s Chomského hierarchií

Pravidla gramatiky L-systému jsou aplikovány na větné formy iterativně počínaje aplikací nad startujícím řetězcem. V případě kdy dochází k opakované aplikaci téhož pravidla, hovoříme o stupni rekurze (zanoření). Počet těchto zanoření je zpravidla omezen, tak abychom dosáhli potřebné přesnosti (jemnosti) ve výsledné rozgenerované struktuře. Navíc na rozdíl od běžného pojetí, kde standardně požadujeme za konec derivace stav, kdy jsou všechny symboly větné formy zastoupeny jen terminály (tzv. „*úspěšná derivace*“), v případě *L*-systému není tento požadavek nutný a derivaci lze považovat za úspěšnou, pokud již větná forma (včetně tzv. neterminálů) prošla požadovaným stupněm derivačních kroků (bylo dosaženo potřebného rozgenerování).

Cílový řetězec (věta) generovaný L-systémem je zpravidla podroben sémantickému zpracování, kde požadujeme, aby ve výsledné větě existovaly právě takové symboly, které mají jistou sémantickou nosnost (představují akci, která se má provést při jejich výskytu ve větě).

S využití například tzv. „*želví geometrie*“ (viz [10]) jsme schopni generovanou větu sémanticky interpretovat dle významu jednotlivých symbolů a tím zároveň schopni získaný řetězec vyjádřit grafickým výstupem.

Pozn.: želví geometrie (známá již z programovacího jazyka Logo - 1967) je zastoupena symbolickou želvou, která se pohybuje po ploše. Přičemž má schopnost zanechávat za sebou ocasem stopu, nebo se jen volně posouvat. Dále je schopna natočení (zpravidla v jednom směru) o libovolný úhel.

Po potřeby želví geometrie v L-systému zpravidla předem definujeme jednotlivé významové symboly, které představují příkazy pro pohyb želvy v ploše. Například symbol '+' může být dán jako příkaz *„pravotočivé otočení o konstantní úhel*", nebo symbol '=' může vyjadřovat vykreslení úsečky konstantní délky, zatímco symbol '-' bude znamenat vykreslení téže úsečky, ale s poloviční délkou. Dále budou uvedeny některé příklady významných struktur, které jsme schopni pomocí L-systému generovat.

Příklad 4.1: Řasa – **Lindermayerův původní model růstu řasy**

$V = \{A, B\}$,

$T = V$,

$w = A$,

$P = \{A \rightarrow AB, B \rightarrow A\}$

Systém produkuje po jednotlivých derivačních krocích následující řetězce:

$n = 0 : A$

$n = 1 : AB$

$n = 2 : ABA$

$n = 3 : ABAAB$

$n = 4 : ABAABABA$

Příklad 4.2: **Fibonacciho řada**

$V = \{A, B\}$,

$T = V$,

$w = A$,

$P = \{A \rightarrow B, B \rightarrow AB\}$

Systém produkuje po jednotlivých derivačních krocích následující řetězce:

$n = 0 : A$

$n = 1 : B$

$n = 2 : AB$

$n = 3 : BAB$

$n = 4 : ABBAB$

$n = 5 : BABABBAB$

$n = 6 : ABBABBABABBAB$

$n = 7 : BABABBABABBABBABABBAB$

Pokud si vyjádříme délku každého z řetězců získaných v jednotlivých derivačních krocích, obdržíme tzv. *Fibonacciho posloupnost*, čili sekvenci čísel:

1 1 2 3 5 8 13 21 34 55 89….

Příklad 4.3.: **Cantorovo diskontinuum**

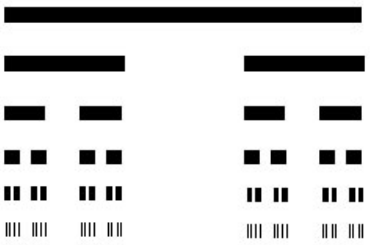
$V = \{A, B\}$,

$T = V$,

$w = A$,

$P = \{A \rightarrow ABA, B \rightarrow BBB\}$

Nechť A má význam „*kresli úsečku vpřed*“ a B má význam „*pohyb vpřed*“. Interpretací na generované formě pak dostáváme obrazec známý jako tzv. *Cantorova fraktálová množina*, viz obr. 4.2.



Obr. 4.2 – Cantorova fraktálová množina

Příklad 4.4.: **Kochova křivka – varianta jen s pravými úhly**

$V = \{F, +, -\}$,

$T = \{+, -\}$,

$w = F$,

$P = \{F \rightarrow F+F-F-F+F\}$

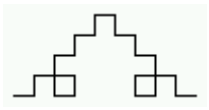
Nechť F má význam „*kresli úsečku vpřed*“, ‘+‘ má význam „*otočení želvy o 90° levotočivě*“, ‘−‘ má význam „*otočení želvy o 270° levotočivě, respektive o 90° pravotočivě*“. První startující úsečka (axiom) má počáteční úhel 0°. Systém produkuje po jednotlivých derivačních krocích následující řetězce a jejich interpretace:

$n = 0$:

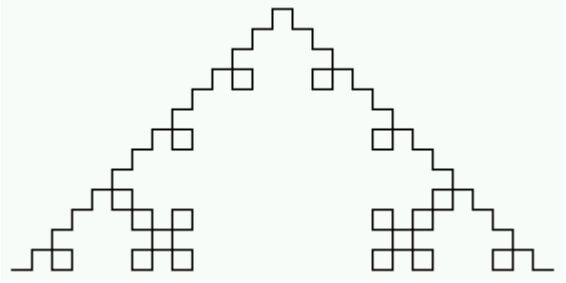
F

$n = 1$: 

$F+F-F-F+F$

$n = 2$: 

$F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F$

$n = 3$: 

F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+ F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F- F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F- F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+ F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F

Příklad 4.5: **Panroseovo pokrytí**

V = {1, 6, 7, 8, 9, +, −, [,]},

T = {+, −, [,]},

w = [7]++[7]++[7]++[7]++[7],

P = { 6 → 81++91−−−−71[−81−−−−61]++,

7 → +81−−91[−−−61−−71]+,

8 → −61++71[+++81++91]−,

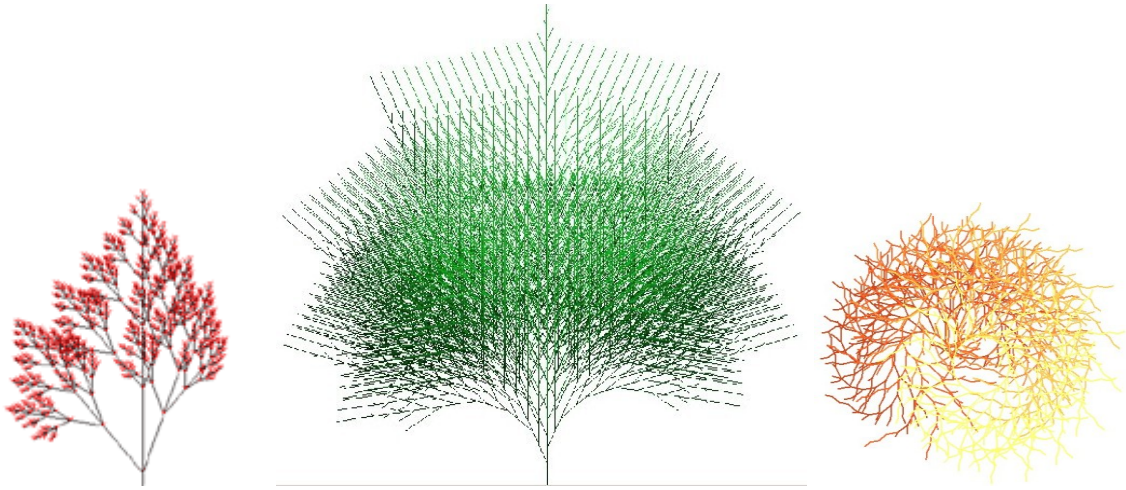
9 → −−81++++61[+91++++71]−−71,

1 → ε}

Význam symbolů ‘+‘ a ‘−‘ je identický jako v příkladě 4.4, ale úhel změny je zde nastaven na konstantu 36°. Zatímco symbol ‘**[**‘ vyjadřuje uložení aktuální pozice (souřadnice a úhel) na zásobník, symbol ‘**]**‘ pak představuje vyvolání záznamu na vrcholu zásobníku a nastavení aktuální pozice dle jeho obsahu. Systém pak produkuje po jednotlivých derivačních krocích interpretace dle obr. 4.3.

Obr. 4.3 – Panroseovo pokrytí generované *L*-systémem pro *n*=1, *n*=2, *n*=3

Následně demonstrujeme již jen přehledově některé další zajímavé struktury generované současnými L-systémy, viz obr 4.4 a 4.5.



Obr. 4.4 – vybrané interpretace L-systému pro 2D grafické prostředí

Obr. 4.5 – vybrané interpretace L-systému pro 3D grafické prostředí

Neformálně lze považovat modely generované L-systémy za organicky blízké (schopnost generovat přírodní struktury z mikro i makrosvěta). Mimo jiné struktury tvořené L-systémy vykazují soběpodobnost, což vedlo k jejich využití v oblasti fraktálové grafiky. Při aplikaci větší hloubky rekurze (rozgenerování) jsme zároveň schopni popsat růst i struktur více komplexních. Díky těmto vlastnostem nalezly L-systémy využití i v oblastech výtvarného umění, počítačové grafiky, modelování umělého života a biomatematiky.

4.2 Návrh metod nad L-systémy v aplikaci multigramatik

V této kapitole se budeme zabývat popisem metod založených na aplikaci multigramatik v L-systémech. Zaměříme se na zobecnění akcí produkovaných L-systémy, následně jejich rozšíření prostředky multigramatik a demonstrace získaných výsledků. Dále budou diskutována využití získaných poznatků z pohledu současných aplikací nad L-systémy.

4.2.1 Zavedení selektivního L-systému

Aplikací multigramatik v L-systémech budeme v základní formě rozumět zavedení tzv. *selektivního L-systému*. Nejprve si myšlenku tohoto systému popíšeme intuitivně a posléze formálně nadefinujeme. V kapitole 4.1 již byl zmíněn důležitý poznatek o L-systémech: L-systém je („*čistě*“) paralelním generativním systémem. Na druhé straně je v multigramatice, jakožto „*částečně*“ paralelním systému, aplikace jakéhokoli pravidla navíc podmíněna splněním kontextové podmínky selektoru, kterým lze postihnout obsah aktuální větné formy (přesněji viz příklady derivace v multigramatikách v kapitole 2.3). Neformálně lze říci, s využitím paralelních přepisů (možných jak v multigramatikách i L-systémech) a zavedením podmíněného (řízeného) derivačního kroku (podmínku představuje kontext požadovaný selektorem multigramatiky) získáme nový generativní systém: ***selektivní L-systém***.

Zatímco je během derivačního kroku v L-systému vybráno pravidlo, jehož levá strana se nachází v 1 až n výskytech ve větné formě (všechny výskyty jsou paralelně nahrazeny pravou stranou pravidla v situace při rozgenerování), v případě selektivního L-systému bude navíc výběr pravidla podmíněn jeho explicitním povolením na základě selektoru podobně jako je tomu u zmiňovaných multigramatik.

Definice 4.5

Selektivní L-systém G je pětice $G = (V, T, P, w, K)$, kde V, T, K mají stejný význam jako v multigramatice a w je počáteční řetězec (axiom – podobně jako u $E0L$-systému), kde $w \in V^*$. P je konečná množina pravidel tvaru: $a \rightarrow x$, kde $a \in V$ a $x \in V^*$.

Pro derivační krok v selektivním L-systému platí totéž co v případě multigramatiky s rozdílem zahájení derivace, kde na rozdíl od multigramatiky nezačínáme derivaci ze startujícího neterminálu „S“ ale z počátečního axiomu „w“.

Příklad 4.6: **ukázka derivace v selektivním L-systému**

$V = \{F, +, -, c, 4, 5, h\}$,
$T = \{+, -, c, 4, 5, h\}$,
$w = Fc45hFc45Fc45hFc45Fc45hFc45Fc45hF$,
$P = \{F \rightarrow F{-}hFc15F{+}F{+}F{-}hFc345FhF\}$,
$K = \{.^{*}h\langle F\rangle.^{*}\}$

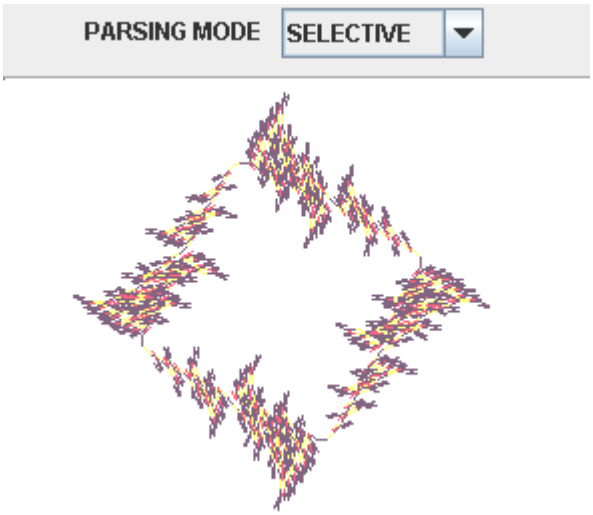
Zde zavedené symboly využívají notaci navrženou pro implementaci selektivního L-systému v jazyce Java. Pro přehled jen zmíníme některé nezbytné souvislosti. Základním vykreslovacím elementem je zde symbol *F*, symboly před jeho výskytem představují vždy parametry ovlivňující vlastnosti pro jeho vykreslení, dále pak symboly +, − mají stejný význam jako v příkladě 4.4 s výjimkou konstanty úhlu, která je nastavena na 140°. Symbol „*c*“ má význam „*ponatočení želvy o úhel, který je dekadicky číselně vyjádřen bezprostředně za výskytem tohoto symbolu*“. Symbol „*h*“ představuje pomocný znak pro zachycení vybraného kontextu ve větné formě. Syntaxe užitá pro zápis selektoru odpovídá syntaxi regulárního výrazu užité v programovacím jazyce Java (viz [11]). Aktivní část je v selektoru vyznačena pomocí špičatých závorek, tedy: <*active_part*>. Podobně jako v předchozích příkladech budeme číselně jednotlivé derivační kroky označovat kladnou celočíselnou proměnnou *n*. Další podrobnosti k syntaxi zápisu a významu jednotlivých symbolů jsou uvedeny v implementační kapitole 6.

Pro $n = 1$ je průběh derivace v selektivním L-systému následující:

$Fc45hFc45Fc45hFc45Fc45hFc45Fc45hF \Rightarrow$
$Fc45hF{-}hFc15F{+}F{+}F{-}hFc345FhFc45Fc45hF{-}hFc15F{+}F{+}F{-}hFc345FhF$
$c45Fc45hF{-}hFc15F{+}F\ {+}F{-}hFc345FhFc45Fc45hF{-}hFc15F{+}F{+}F{-}hFc345FhF$

Pozn.: paralelní nahrazení proběhlo jen v částech povolených selektorem $.^{*}h\langle F\rangle.^{*}$, tedy jen v případě takových výskytů symbolů *F* v axiomu, které byly bezprostředně předcházeny pomocným symbolem *h*. Symbol *h* zde jinak nepředstavuje pro grafickou interpretaci jiný význam. Na rozdíl od běžného L-systému byly tedy ve větné formě rozgerovány jen některé symboly *F*, zatímco jiné zůstaly v latentním stavu (bez přepisu). Výběr přepisovaných částí prostřednictvím selektoru označujeme obecně selekcí, odtud zavedené označení *selektivní L-systém*.

Pro n=3 dostáváme interpretací věty selektivního L-systému grafický výstup, viz obr. 4.6.



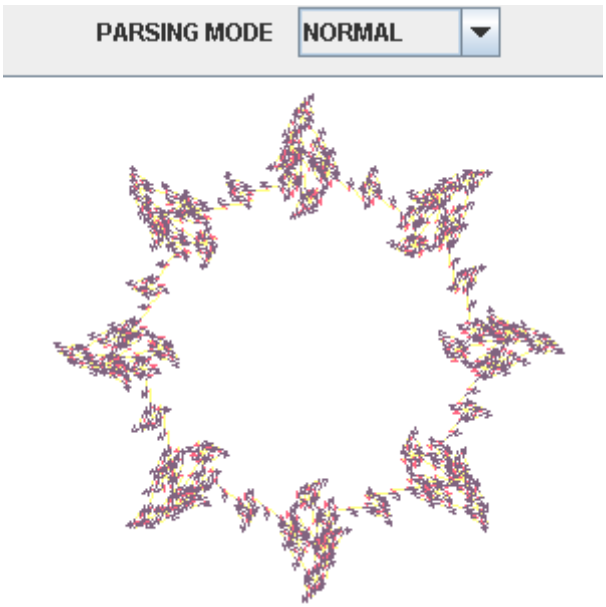
Obr. 4.6 – interpretace selektivního L-systému pro n = 3, viz příklad 4.6
(výstup z vytvořeného program pro demonstraci selektivního L-systému)

Pro srovnání uvedeme situaci provedení derivace při užití E0L-systému, tedy bez souvislosti s množinou selektorů K. Pro n = 1 je průběh derivace v E0L-systému následující (přepsány jsou nyní bez rozdílu všechny výskyty symbolů F v axiomu):

$Fc45hFc45Fc45hFc45Fc45hFc45Fc45hF \Rightarrow$

$F{-}hFc15F{+}F{+}F{-}hFc345FhFc45hF{-}hFc15F{+}F{+}F{-}hFc345FhFc45F{-}hFc15F{+}F{+}F{-}hFc$
$345FhFc45hF{-}hFc15F{+}F{+}F{-}hFc345FhFc45F{-}hFc15F{+}F{+}F{-}hFc345FhFc45hF{-}hFc15$
$F{+}F{+}F{-}hFc345Fh\ c45F{-}hFc15F{+}F{+}F{-}hFc345FhFc45hF{-}hFc15F{+}F{+}F{-}hFc345FhF$

Pro n=3 dostáváme interpretací věty E0L-systému grafický výstup, viz obr. 4.7.



Obr. 4.7 – srovnávací interpretace *E0L*-systému pro n = 3, viz příklad 4.6
(výstup z vytvořeného program pro demonstraci selektivního L-systému)

Porovnáním výsledku dosaženého interpretací nad E0L-systémem (obr 4.7) a výsledku získaného interpretací nad selektivním L-systémem (obr 4.6) získáme přehled nad provedenou selekcí, které jsme zavedením selektorů z množiny K chtěli dosáhnout.

Pozn.: takto provedená selekce zároveň představuje efektivní nástroj pro zanesení jistého prvku plánované nepravidelnosti (asymetrie) do původního symetrického L-systému. Tato vlastnost je zejména vhodná pro popis nesymetrických jevů, které se v přírodních jevech vyskytují.

4.2.2 Aplikační aspekty selektivního L-systému

Schopnost paralelního zpracování v selektivních L-systémech nám dovoluje uvažovat o návrhu algoritmů, které by vedly na snížení časové složitosti v porovnání se sekvenčním přístupem. Navíc s rozšířením HW architektur pro podporu paralelních aplikací se z budoucího vývoje ukazuje tento požadavek na paralelní algoritmizaci úloh jako věc nezbytná.

Dále zmíníme některé další aspekty, které souvisejí s abstrakcí zavedenou nad akcemi prováděných v selektivních L-systémech. Dosud jsme význam *akce* v selektivním *L*-systému chápali jako aplikaci pravidla nad větnou formou (derivační krok) povoleného některým selektorem. Nyní význam *akce* zobecníme o *další operace*, které mohou být provedeny nad větnou formou (nepůjde tedy jen o *operaci nahrazení* na základě pravidla povoleného selektorem).

Z pragmatického hlediska budeme uvažovat následující operace:

- *mocnina řetězce*,
- *konkatenace s jiným řetězcem*,
- *reverze řetězce*, na rozdíl od ostatních operací vyžaduje tato operace rozšíření selektivního L-systému na povolení přítomnosti řetězce i na levé straně pravidel, viz definice 4.6
- *vyjmutí vybraného řetězce z větné formy*

Definice 4.6

Rozšířený selektivní L-systém G' je pětice $G' = (V, T, P, w, K)$, kde V, T, K, w mají stejný význam jako v *selektivním L-systému* a P je konečná množina pravidel tvaru: $a \rightarrow x$, kde $a, x \in V^*$.

Každou ze zmíněných operací je možno simulovat pomocí upraveného pravidla selektivního (popř. rozšířeného) L-systému. Demonstrace simulace zmíněných operací:

- *mocnina řetězce*, např.: $F \rightarrow F^k$, kde k je celočíselná kladná konstanta
- *konkatenace s jiným řetězcem*, např.: $F \rightarrow F \,.\, y$, kde $y \in V^*$ a '.' je operace konkatenace
- *reverze řetězce*, např.: $y \rightarrow reverse(y)$, kde $y \in V^*$ a operace *reverse*() je převrácení řetězce
- *vyjmutí vybraného řetězce z větné formy*, např.: $F \rightarrow \varepsilon$

Příklad 4.7: ukázka derivace v rozšířeném selektivním L-systému s využitím akce: *operace převrácení řetězce ve větné formě*. Nechť je dán následný rozšířený selektivní L-systém:

$V = \{F, c, 8, 5, h\}$,
$T = \{c, 8, 5, h\}$,
$w = Fc85hFc85Fc85hF$,
$P = \{1p: 85 \rightarrow reverse(85)\}$ – pravidlo pro simulaci reverse, rozšířený tvar,
$K = \{.*<1p>h.*\}$, kde $1p$ je reference na číslo prvního pravidla

Význam užitých symbolů je identický s významem uvedeným v příkladě 4.6. Pro $n = 1$ je průběh derivace v rozšířeném selektivním L-systému následující:

$$Fc85hFc85Fc85hF \Rightarrow Fc58hFc85Fc58hF.$$

Jak můžeme z výsledku derivace vidět, došlo na základě akce povolené selektorem k převrácení hodnoty úhlu pro natočení želvy jen v krajních výskytech větné formě, zatímco prostřední hodnota úhlu zůstala zachována, což vede při následné grafické interpretaci ke změně tvaru v generované geometrické struktuře. Selektory, které povolují operaci reverzi řetězce - simulované pravidlem, označujeme jako tzv. *reverzní selektory*.

S využitím výše zmíněných operací jsme schopni zavést modifikace nad větnou formou a vyjádřit tak požadované deformace v generované struktuře následně reprezentované grafickým výstupem. Z pragmatického hlediska budeme však během derivace v selektivním L-systému uvažovat kombinace aplikací běžných přepisovacích pravidel s pravidly pro simulace zavedených operací. Zpravidla pak bude mít derivace následující průběh:

1. proveď aplikace běžných přepisovacích pravidel povolených selektory do požadovaného stupně zanoření
2. na výsledné větné formě proveď dále aplikaci simulačních pravidel (pro zavedené operace) povolených selektorem
3. proveď grafickou interpretaci výsledného řetězce například pomocí želví geometrie

Zavedením tohoto průběhu derivace je možné snadno definovat změny základní generované struktury v požadovaný výsledek. Praktické výstupy jsou demonstrovány v implementační kapitole 6.

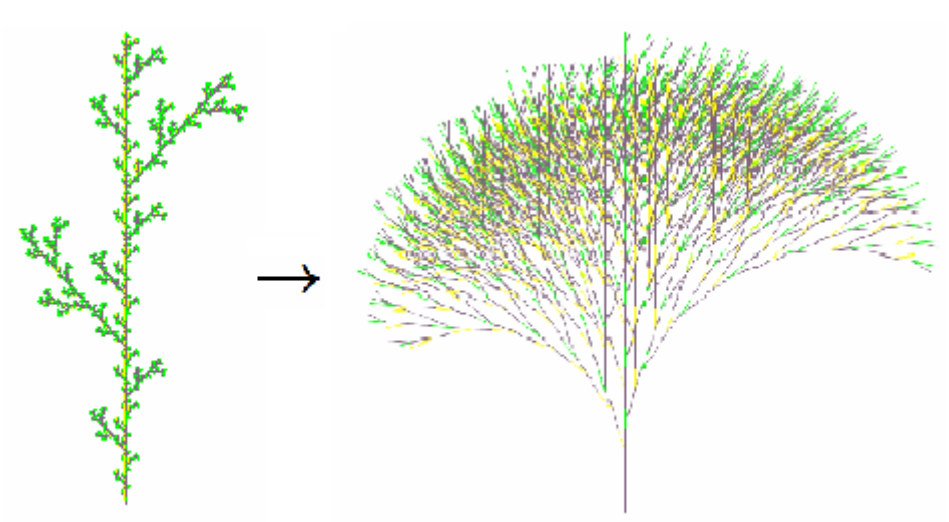
4.2.3 Rozšiřující aplikace selektivního L-systému

Zatímco příklady z aplikací zavedeného selektivního L-systému uvedené výše bychom mohli zařadit do oblasti výtvarné informatiky, respektive fraktálové grafiky, nyní ještě uveďme další možné využití selektivního L-systému z pohledu **biomatematiky**. Inspirace této aplikace vychází z poznatků Prof. D'Arcy Thompsona, jak již bylo naznačeno v kapitole 3.3.

Příklad 4.10: selektivní L-systém v aplikaci biomatematiky, nechť je dán selektivní L-systém:

$V = \{F, A, +, -, c, 2, 7, 0, h, [,]\}$,
$T = \{+, -, c, 2, 7, 0, h, [,]\}$,
$w = c270F\ F\ hA$,
$P = \{1p: A \rightarrow FA[-FhA]FA[+FhA]FhA\}$,
$K = \{ .^{*}h<1p>.^{*}\}$

Pro počet iterací n, kde n=6 dostáváme interpretací nad výslednou větou grafický výstup na obr. 4.9 (vpravo), zatímco obr. 4.9 (vlevo) je tvořen E0L-systémem, jehož předpis je však totožný s předpisem pro zde zmíněný selektivní L-systém s výjimkou absence množiny selektorů K.



Obr. 4.9 – proměna květů v závoj; redukce tvarů rostlin (mezitřídní variability) zavedením selekce dle příkladu 4.10 nad původním vzorem (vlevo) generovým E0L-systém. (výstup z vytvořeného program pro demonstraci selektivního L-systému)

4.3 Zhodnocení, porovnání s ostatními metodami

Rozšíření L-systémů a to zejména v oblastech biologie a fraktálové grafiky vedlo k zavedení několika různých modifikací (rozšíření) z pohledu praktického použití. V předchozí kapitole 4.1 již byly zmíněny 0L, E0L, ET0L systémy, které představují teoretický a empiricky orientovaný základ pro aplikace ve zkoumaných oborech. V této kapitole proto dále uvedeme některé další často využívané varianty L-systémů z pohledu pragmatického návrhu pro dané oblasti. Následně budeme demonstrovat jejich srovnání s prostředky navržených selektivních L-systémů. Přehledově jsou zmíněny následující současné přístupy v L-systémech:

1. stochastické
2. parametrické
3. zásobníkové
4. kontextové (levě i pravě)
5. kombinované

Stochastické L-systémy, pokud se v systému vyskytuje více pravidel s totožnou levou stranou, pak je každé takové pravidlo rozšířeno o výraz, který určuje pravděpodobnost užití tohoto pravidla. Např. pro *n*-násobný výskyt symbolu *F* na levé straně pravidel platí: $F \rightarrow \alpha_1 : p_1$, $F \rightarrow \alpha_2 : p_2$, $F \rightarrow \alpha_n : p_n$; kde $F \in V$, $\alpha_i \in V^*$, $p_i \in \langle 0, 1 \rangle$, kde p_i je pravděpodobnost užití pravidla $F \rightarrow \alpha_i : p_i$ při přepisu symbolu *F* a platí: $\sum_1^n p_i = 1$.

Parametrické L-systémy, pracují s tzv. parametrickými slovy. Parametrické slovo je dáno jako posloupnost modulů. Každý modul obsahuje formální parametry, které mohou nabývat hodnot (skutečných parametrů) z množiny $\Re$. Parametry mohou být vyjádřeny také skrze aritmetické a logické výrazy. Formálně řečeno: parametrický *L*-systém G_p je čtveřice $G_p = (V, \Sigma, \omega, P)$, kde *V* je abeceda symbolů, Σ je množina formálních parametrů, ω - axiom je neprázdné parametrické slovo, $\omega \in (V \times \Re)^+$ a pravidlo *p*, $p \in P$ je tvaru: *pred* : *cond* $\rightarrow$ *succ*, kde *pred* je levé strana pravidla, *cond* je logický výraz nabývající hodnot 0 nebo 1 a *succ* je pravá strana pravidla. Ukázka parametrického L-systému:

$\omega = B(2)A(3, 4)$

$P = \{A(a, b) : b < 2 \rightarrow B(a + b),$

$A(a, b) : b > 3 \rightarrow D,$

$B(a) : a < 1 \rightarrow \mathrm{A}(a - 1),$

$B(a) : a \geq 1 \rightarrow A(a^2)\}$

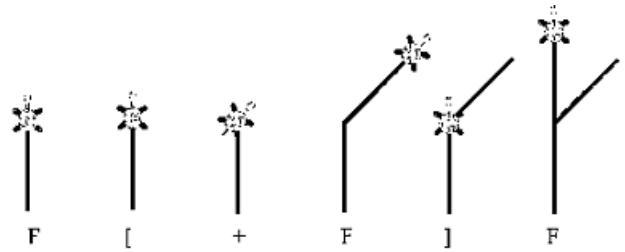
Demonstrace derivačního kroku: $B(2)A(3, 4) \Rightarrow A(9)D$

Zásobníkové L-systémy, abeceda symbolů je rozšířena o tzv. zásobníkové symboly, které mají sémantickou nosnost provádět operace nad zásobníkem pro kreslící želvu. Příklad značení zásobníkových symbolů a jejich význam:

[… operace *push* – ulož stav želvy na vrchol zásobníku

] … operace *pop* – vyjmi stav želvy z vrcholu zásobníku a nastav želvu na tento stav

Stav želvy, který má být uložen, je dán její aktuální souřadnicí v prostoru a úhlem natočení případně dalšími parametry (barva, tloušťka štětce želvy), což představuje levý kontext tedy podřetězec věty, který předchází symbol pro uložení tohoto stavu, viz obr. 4.8.



Obr. 4.8 – grafická interpretace věty: $F[+F]F$ v zásobníkovém L-systému, konstantní úhel natočení je roven 45°

Kontextové L-systémy, při přepisování uvažujeme levý a pravý kontext, který se nachází v okolí přepisovaného symbolu. Levý i pravý kontext je dán jen posloupností symbolů z abecedy. Pravidla kontextového systému pak mají tvar:

$$lexft\_context <A> right\_context \rightarrow \alpha;\ \text{kde}\ \alpha, lexft\_context, right\_context \in V^*, A \in V$$

Význam pravidla: symbol A je přepsán, jestliže je bezprostředně přecházen řetězcem *left_context* a bezprostředně následován řetězcem *right_context*.

Kombinované L-systémy, představují L-systém, který využívá prostředků z výše uvedených modifikací, viz body 1. až 4. V obecném případě tak získáme pravidla ve tvaru:

$lexft\_context <A> right\_context : cond \rightarrow \alpha : p$; kde α, $lexft\_context, right\_context \in V^*, A \in V$, *cond* má stejný význam jako v parametrickém L-systému a p jako ve stochastickém L-systému.

Modifikace *L*-systémů zde uvedené představují v současnosti důležitou roli pro využívané aplikační oblasti. Dále uvedeme tyto modifikace v porovnání se zavedenými selektivními L-systémy. Pokud uvážíme srovnání obecného selektivního L-systému, lze hovořit o abstraktním návrhu formálního prostředku, který je schopen postihnout i přístupy výše uvedené. Navzdory možnostem simulovat v selektivním systému například pravděpodobnost aplikace pravidla nebo splnění aritmeticko-logické podmínky (při simulaci stochastického nebo parametrického *L*-systému) na základě podmíněného

kontextu v selektorech a predikcí následných větných forem, budeme z důvodu snazšího – přímého vyjádření uvažovat spíše rozšíření selektivního *L*-systému o pravděpodobnostní výraz podobně jako v případě stochastického L-systému popřípadě o podmínkový výraz jako u parametrického *L*-systému. V ostatních případech budeme schopni vyjádřit simulace formou přehledného zápisu bez nutnosti zavedení dalších modifikací a to na základě regulárních výrazů v selektorech a pomocných symbolů pro zachycení požadovaného kontextu ve větné formě. Uvažme nyní následující demonstrační příklady 4.8 a 4.9.

Příklad 4.8: srovnání možností v selektivních *L*-systémech. Nechť je dán následný kombinovaný L-systém:

$\omega = aaB(3)bbB(2)ccA(3, 4)dd$

$P = \{ 1p: cc<A(a, b)>dd : b < 2 \rightarrow B(a + b) : 0.6,$

$2p: cc<A(a, b)>dd : b > 3 \rightarrow D : 0.4,$

$3p: aa<B(a)>bb : a < 1 \rightarrow A(a - 1) : 0.7,$

$4p: bb<B(a)>cc : a \geq 1 \rightarrow A(a^2) : 0.3\}$

Derivačního krok má průběh: $aaB(3)bbB(2)ccA(3, 4)dd \Rightarrow aaB(3)bbA(9)ccDdd$
(na základě podmínky kontextu, parametrů a pravděpodobnostního limitu byl přepsán jen druhý výskyt symbolu *B* pomocí 4. pravidla a jediný výskyt symbolu *A* pomocí 2. pravidla).

Příklad 4.9: srovnání možností v selektivních L-systémech. Nechť je dán následný simulačně upravený selektivní L-systém:

$\omega = aaB(3)bbB(2)ccA(3, 4)dd$

$P = \{ 1p: A(a, b) : b < 2 \rightarrow B(a + b) : 0.6,$

$2p: A(a, b) : b > 3 \rightarrow D : 0.4,$

$3p: B(a) : a < 1 \rightarrow A(a - 1) : 0.7,$

$4p: B(a) : a \geq 1 \rightarrow A(a^2) : 0.3\}$

$K = \{aa<3p>bb, bb<4p>cc, cc<2p>dd\}$

Derivačního krok má průběh: $aaB(3)bbB(2)ccA(3, 4)dd \Rightarrow aaB(3)bbA(9)ccDdd$
(na základě podmínky v selektoru, parametrického výrazu a pravděpodobnostního limitu byl přepsán jen druhý výskyt symbolu *B* pomocí 4. pravidla a jediný výskyt symbolu *A* pomocí 2. pravidla).

Pozn.: srovnáním zmíněného kontextového L-systému (vyjadřuje levý a pravý kontext jen formou řetězcových konstant) jsou možnosti selektivního L-systému širší (zejména možností vyjádřit levý i pravý kontext prostředky regulárních výrazů).

Uvedli jsme přehled vlastností současných případů L-systémů a provedli porovnání se zavedenými selektivními L-systémy formou demonstračních příkladů. Obecně můžeme z tohoto pohledu prohlásit selektivní L-systém za abstraktní prostředek pro popis modifikací nad dnešními L-systémy podpořený pragmatickými aplikacemi s vlastností generovat dosud nezkoumané struktury, schopnost vnášet přirozeně asymetrii do fraktálových obrazců. Další praktické výstupy jsou demonstrovány v implementační kapitole 6.

5 Tematická oblast, zpracování přirozených jazyků

V této části se zaměříme zejména na zpracování jazyka prostředky analyzátoru, který je založen na formálním modelu analyzovaného jazyka. Mimo tento přístup jsou dnes využívané také metody statisticky orientované (s aplikacemi neuronových sítí, rozhodovacích stromů), které však zpravidla vyžadují spolupráci se znalostní databází tzv. korpusem.

V oblasti zpracování přirozeného jazyka hovoříme o tzv. rovinách popisu (zpracování) jazyka. Tyto roviny jsou uspořádány zdola nahoru od roviny jednodušší (zabývající se lexikografickou částí textu) po rovinu složitější, rovinu významu. Každá rovina má své jednotky popisu, definice vztahů na této rovině, a navazuje bezprostředně na rovinu vyšší. Zpravidla se hovoří o pěti rovinách (lexikální, morfologické, syntaktické, sémantické a pragmatické), ale často se (např. z praktického hlediska) některé roviny slučují nebo prolínají.

V následujících několika kapitolách se budeme zabývat zpracováním s využitím třech základních rovin popisu (lexikální, morfologické a syntaktické). V každé úrovni popisu se nejprve zaměříme na obecnou formu a prostředky pro její zpracování. Následně bude diskutována podmnožina pro hypotetický jazyk s využitím syntaktického analyzátoru při aplikaci multigramatik.

Zde zmíněná terminologie zahrnuje výrazy zejména lingvistické. Kromě české odborné terminologie budou také použity jejich ekvivalenty známé z prostředí mezinárodních publikací.

5.1 Úvod do současných prostředků

5.1.1 Lexikografická rovina

Představuje prvotní činnost analyzátoru. Hlavním cílem je separovat jednotlivé lexikální jednotky – lexémy a provedení případné filtrace textu a indexace nad rozpoznanými lexémy. V této fázi zpracování se také mimo vlastí jádro (lexikální analyzátor) setkáme s doprovodnými metodami, které nám pomáhají výsledky samotného analyzátoru zpřesnit nebo doplnit.

Postupy při lexikální analýze:

1. **lexikální analyzátor** (*scanner*) – provádí se určení jednotlivých slov a sousloví ve vstupním textu, někdy se identifikace sousloví zavádí, vzhledem ke své složitosti, jako samostatný proces.

2. **nevýznamová nebo nespecifikovaná slova** (*filter*) – jedná se proces, který za pomoci předem připraveného negativního slovníku nebo regulárních výrazů tato slova odstraňuje; někdy se tato procedura považuje za nedílnou součást lexikálního analyzátoru.
3. **lemmatizace** (*stemming*) – cílem je snaha získat základní tvary detekovaných slov, tj. kmen nebo kořen (opakem tohoto procesu je derivace, kdy se k základnímu tvaru slova generují jeho inflexní tvary).
4. **srovnání slov** (*comparison*) – slouží k porovnávání kmenů nebo kořenů s výrazy řízeného slovníku (např. korpus).
5. **vážení** (*weighting*) – stanovení jednotlivých vah výrazů - nejčastěji se provádí na základě frekvenčních metod.

5.1.1.1 Lexikální analyzátor

Jednotlivé lexémy jsou v textu rozpoznávány pomocí slovních separátních znaků. Za tyto separátory nejčastěji považujeme množinu bílých znaků, ačkoli v případě některých typů slov není tato identifikace zcela jednoznačná. Častějším problémem je např. určování zkratek, kde je třeba tečku odlišit od větného separátoru. Dále pak jazyky, které umožňují zápis výrazu se spojovníkem, se potýkají s problémem, kde je nutné rozpoznat, zda se jedná o celistvý výraz nebo o dvě oddělená slova.

Samostatný problém během rozpoznávání slov představují také číslice, kde je zapotřebí určit, zda budou vyhodnoceny jako samostatné výrazy nebo části na sobě závislé (např. spojení: *2. kolo*) nebo zda budou pro a další postupy zpracování vypuštěny. O mnoho složitější je problematika rozpoznání sousloví, která jsou z hlediska sémantické nosnosti a selektivní síly významnější než samostatná slova. Pro rozpoznání sousloví přirozených jazyků bylo vyvinuto několik postupů, zde uvádíme dva nejčastěji používané:

- **statistická identifikace** – v případě kdy se ve vstupním textu vyskytují často některá slova společně, můžeme předpokládat, že se jedná o sousloví. Vychází se přitom z četnosti výskytu daného sousloví (závisí na pořadí slov), současného výskytu slov (nezávisí na pořadí slov) nebo vzdálenosti slov v textu, kde vzdálenost je určena buď počtem slov mezi slovy myšleného sousloví, nebo může být sledován jejich současný výskyt ve větě, odstavci, bloku nebo jiné části textu. Metoda nemusí být vždy zcela úspěšná, neboť vysoká četnost současného výskyt dvou slov ještě není schopna jednoznačně určit, zda jde o sousloví či nikoli.
- **syntaktická identifikace** – vychází se z principu předešlé metody. Navíc je zde u předpokládaného sousloví pomocí dalších metod analyzována jejich syntaktická složka. K tomu nám může napomoci např. přítomnost korpusového slovníku.

Součástí rozpoznávání sousloví může být také normalizace jejich formy. Je zcela běžné, že jednotlivá sousloví se v textu mohou vyskytovat v různých syntaktických, lexikálních nebo morfologických variantách, které je vhodné pro potřeby analyzátoru sjednotit. Používá se následujících metod:

- **normalizace formy** za pomoci „*slovníku variant sousloví*“ - jedná se o jednoduchou metodu, která je však většinou z důvodu omezeného rozsahu slovníku vhodná pouze v rámci určitého odborného zaměření, kterého se text týká.
- **odstraněním slovních druhů** (předložek, spojek, částic apod.) ze sousloví a zanedbáním uspořádání ostatních částí sousloví.
- **syntaktická analýza sousloví** v kombinaci s použitím kmene nebo kořene jednotlivých slov sousloví.

5.1.1.2 Filtrace nevýznamových nebo nespecifikovaných slov

Nevýznamovými slovy rozumíme funkční části textu, které nejsou nositeli význam (patří sem např. spojky, předložky, částice, určitý/neurčitý člen apod.). Nespecifikovaná slova chápeme jako samostatná slova či sousloví, jež se vyskytují ve větším případě textů. Z tohoto důvodu mají z pohledu vyhledávání minimální selektivní vliv. Oba druhy těchto útvarů je třeba s pomocí negativního slovníku odstranit, dochází tak k minimalizaci šumu pro následné fáze analýzy. K vytvoření negativního slovníku je možno použít některé z následujících metod:

- **kritériem jsou slovní druhy**, které nenesou význam a mají jen funkci syntaktického doplnění věty (např. spojky, předložky, částice apod.)
- **kritériem jsou slova s vysokou hodnotou** absolutní nebo relativní četnosti výskytu v textu; vycházíme z empirických předpokladů, kde nevýznamová nebo nespecifická slova mají mnohem vyšší hodnotu četnosti v textu než významová slova. Nedostatkem této metody je, že se v množině slov s vysokou četností se může vyskytnout i důležitý významový termín
- **kritériem jsou krátká slova**; vycházíme z předpokladu, že nevýznamová slova jsou tvořena kratšími řetězci než slova významová. Zde je ovšem nezbytné použití anti-negativního slovníku, ve kterém budou tyto krátké výrazy uloženy.

5.1.1.3 Lemmatizace

Slova a sousloví se v případě většiny přirozených jazyků vyskytují v určitém tvaru mutace od svého základu (tzv. *lemmatu*). Mutací zde rozumíme odlišná tvarosloví vzniklá úpravou čísla, flexí nebo jiných gramatických kategorií. Pro potřeby analýzy je vhodné extrahovat z tvaru mutace jeho předka tedy „*lemma*“. Tento proces nazýváme lemmatizací. Následuje přehled možných metod:

- **slovník kmenů nebo kořenů**; kladným prvkem je vysoká přesnost ovšem na úkor rozsahu takového slovníku, často se jedná jen o určitou odborně zaměřenou část z jazyka
- **odstranění afixů**; vynecháním předpon (*prefix*) a přípon (*sufix*) lze dosáhnout tvaru, který lze považovat za základní. Je ale nezbytné použití takových metod, které nám umožní sledovat i nepravidelnou flexi jako jsou např. hláskové změny (stolu x stůl). Odstraňování probíhá na základě známého seznamu předpon a přípon nebo s ohledem na znalost pravidel, dle kterých jsou tyto tvary vygenerovány
- **statisticky**; na základě skupin ve slově jdoucích znaků a s pomocí četnosti jejich jednotlivých uspořádání jsme schopni detekovat konstrukci slovní stavby (předpona, kořen, přípona). Kladným prvek je nezávislost na cílovém jazyce, metoda však nelze použít pro zpětnou derivaci.

5.1.1.4 Vážení

Pro množinu přirozených jazyků je typické, že jednotlivá slova se mohou ve větách v závislosti také na jejich obsahu vyskytovat s určitou důležitostí – tzv. *váhou*. Stanovujeme tedy váhu jednotlivých termínů a vytváříme tak výsledný indexační seznam. Tato metoda se využívá zejména v případě slovní indexace (vyhledání termínů na základě specifikovaných slovních kritérií).

5.1.2 Morfologická rovina

Morfologická analýza je proces sestavení množiny morfologických kategorií (tzv. gramatických rysů) pro rozpoznané lexikální jednotky (lexémy) a stanovení jejich hodnot.

Vzhledem ke skutečnosti, že morfologická analýza pracuje zpravidla s jednotlivými slovy v textu izolovaně, bez ohledu na kontext, nelze tedy jednoznačně identifikovat hodnoty všech kategorií. Kontextové zpracování často náleží až rovinám vyššího řádu.

5.1.2.1 Proces morfologické analýzy

Při zpracování se zavádí tzv. množina morfologických značek (*morphological tagset*). Každá tato značka pak obsahuje hodnoty morfologických kategorií pro jeden slovní tvar.

Dle způsobu návrhu těchto značek se vychází z několika druhů odlišných notací, mezi nejrozšířenější patří notace tzv. *poziční*. Zde se ke každé kategorii přiřadí pozice ve značce a každé hodnotě jeden znak, který je zapsán na příslušnou pozici. Například slovní druh se může nacházet na první pozici a jeho hodnoty jsou reprezentovány zkratkami lingvistických pojmenování, tedy např.: *N* (*noun*), *A* (*adjektivum*), atd.

Během procesu stanovení hodnot je nezbytné pro každý slovní tvar specifikovat množinu všech kombinací hodnot k dané morfologické kategorii, které k danému tvaru mohou náležet. Ačkoli se může tato fáze zpracování zdát poněkud nepřesnou, jedná se o velmi důležitou část, která usnadňuje činnost

v dalších postupech analýzy. Pro příklad lze uvést: v českém jazyce je počet všech možných značek přes 4400 a průměrný počet těchto značek po vhodné morfologické analýze se redukuje na hodnotu menší než 5 (pro jedno slovo běžného textu).

Během analýzy je třeba řešit ještě tzv. *lemmatizaci* (pokud nebyla již provedena v rámci lexikální analýzy). Lemma pro každý slovní tvar vyjadřuje jeho základní podobu. Tento proces nelze také považovat za obecně jednoznačný (opět z důvodu bezkontextového zpracování). Dále je nezbytné u některých jazyků rozlišovat mezi slovy, která se vyskytují v základním tvaru jako homonymní např. *state* (stav - *N*) a *state* (stát - *V*). Lemmatizace musí tato slova také rozlišovat a identifikovat (např. s využitím indexu k základnímu tvaru tedy: *state-1*, *state-2* apod.).

Z formálního pohledu je tedy možné definovat morfologickou analýzu jako matematickou funkci, která posloupnosti znaků daného jazyka přiřazuje množinu možných výsledků, obsahující uspořádanou dvojici <*lemma,tags*>. Morfologická analýza ovšem při zpracování prostředky výpočetní techniky není realizována jen jako funkce ale jako výpočetní proces. Základní datovou strukturou tohoto procesu je morfologický slovník (*korpus*) specifický pro konkrétní potřeby daného jazyka. Ten je pak za pomoci vlastních metod morfologické analýzy využíván (zpravidla je zadán jako jeden z parametrů vstupu morfologické roviny zpracování). Metody samotné by však měli být navrženy jako jazykově nezávislé.

5.1.2.2 Úvod do korpusové lingvistiky

Výpočetní prostředky nám umožňují třídit a klasifikovat, analyzovat a vyhodnocovat jazyková data v rozsahu, který by byl ruční anotací nedosažitelný. Bez prostředků informačních technologií by tedy bylo jen sotva možné dospět k takovému typu poznání jazyka. Lze podrobně zkoumat v podstatě libovolné jazykové jevy a pokoušet se o jejich opravdu přesné a adekvátní generalizace, proti nimž byly dřívější anotační popisy jazyka jen intuitivními aproximacemi. Hromadnost a rozsah zpracovávaných dat vede ke kvalitativním změnám v metodologii empirické vědy, jímž dnešní lingvistika bezpochyby je.

V současnosti se *korpusem* rozumí rozsáhlý vnitřně strukturovaný a ucelený soubor textů daného jazyka elektronicky uložený a zpracovávaný. Texty jsou v korpusu strukturovány a organizovány se zřetelem k využití pro určitý cíl, vůči němuž pak je korpus považován za reprezentativní. Podle účelu existují různé typy korpusů. Dle zdroje textů mohou být korpusy psaného nebo mluveného jazyka, všeobecné nebo specializované na určitý styl, publicistický nebo odborný. Většina korpusů s ohledem na svou reprezentativnost obsahuje v různém poměru zástupce všech možných kategorií textů. Podle uložených dat mohou korpusy obsahovat pouze holé texty nebo texty různě označkované (anotované). Značkované korpusy samozřejmě poskytují více informací o jazyku, a proto je snaha korpusy značkovat. To lze provádět buď ručně, což je ale velice nákladné, nebo automaticky (strojově), což může někdy znamenat zanesení jisté míry nepřesností do značkování. Proto se také mnoho

výzkumů v korpusové lingvistice zabývá právě automatickým značkováním textů. Významným přínosem v této problematice je český národního korpusu (ČNK), jedná se o akademický projekt zaměřený na budování rozsáhlého počítačové databáze anotující především textové dokumenty v češtině (podrobnosti viz [12]).

V rámci dřívější studie zpracování plánovaného jazyka Esperanto jsme se již zabývali otázkami potřebnými pro automatické (strojové) generování korpusu a jeho reprezentací v databázovém systému na základě zavedených algoritmických pravidel specifických pro tento jazyk (viz [13]).

5.1.3 Syntaktická rovina

V souvislosti s prostředky, které jsou v současnosti při zpracování jazyků přirozených na rovině syntaktické použity, se nejprve seznámíme se podstatnými syntaktickými strukturami. Následně budou diskutovány otázky analyzátoru této roviny – parseru nad popsanými strukturami.

5.1.3.1 Větné syntaktické struktury

Z hlediska větných syntaktických struktur rozeznáváme dvě základní, jsou jimi:

- *závislostní struktury*,
- *frázové struktury*.

V případě frázových struktur nacházíme přímou souvislost se zápisem ve formě formálních gramatik (frázová struktura odpovídá derivačnímu stromu v gramatice, která danou větu generuje).

V teorii frázových struktur pracujeme s abstraktními (zástupnými) symboly, které nám vyznačují části vět s ohledem (vazbou) k určitému slovnímu druhu a každá fráze je v konečném důsledku opět rozložena v jednotlivé slovní druhy. Přičemž určité slovní kategorii náleží i její odpovídající fráze. Například fráze pro substantiva označujeme jako substantivní fráze (dle anglické terminologie pak jako *noun phrase* – NP).

Zatímco frázové struktury jsou pevně spjaty se slovosledem (popisují vztahy mezi slovními druhy), závislostní struktury zachycují druhy vztahu mezi hlavou a ostatními členy (závislosti mezi větnými členy). Na rozdíl od frázového modelu je spojení závislostních struktur se závislostní gramatikou umělé (závislostní strom neodráží způsob svého vzniku).

5.1.3.2 Analýza větných struktur – parsing

Pokud chceme posoudit struktury z hlediska jejich syntaktické správnosti (analyzovat) jedná se obecně o problematiku prohledávání stavového prostoru z oblasti *umělé inteligence*.
V takovém případě je nutné nejprve specifikovat požadavky, kterými jsou:

- reprezentace stavu problému,
- počáteční (výchozí) stav a hledaný stav (*goal*),
- pravidla pro hledání následného stavu.

V aplikaci pro syntaktickou analýzu jazyků (*parsing*), jsou požadavky zastoupeny formou:

- stavem problému je progrese dosažená během parsování,
- pravidly pro určení následných stavů jsou tzv. přepisovací pravidla.

Aby však bylo možné provést nalezení cílového stavu s efektivní časovou a prostorovou složitostí nepracujeme v syntaktické analýze s běžnými metodami pro prohledávání stavového prostoru, nýbrž s postupy, které byly navrženy pro tuto cílovou oblast na základě formálního popisu jazyka – parsing.

Jako formálním prostředek pro zápis přepisovacích pravidel je použita zpravidla gramatika s požadovanou výpočetní silou. Ta jakožto generativní systém umožňuje generovat řetězce, které k danému jazyku přísluší. Opakem generátoru je akceptor, ten je pro jazyky bezkontextové popsán modelem zásobníkového automatu (viz [14]). Generátor a akceptor jsou pak základem pro tvorbu (návrh a implementaci) syntaktického analyzátoru – parseru. Parser během své činnosti simuluje pomocí zásobníkového automatu stavbu derivačního stromu, který nám vyjadřuje reprezentaci postupně aplikovaných pravidel (prováděných derivačních kroků) ve stromové hierarchii, viz obr 5.1.

Existují dva zásadní přístupy, kterými může být činnost parseru provedena: analýza zdola-nahoru (*buttom-up*) nebo shora-dolu (*top-down*). V případě parsingu zdola-nahoru začínáme přepisem některé části věty, která odpovídá určité pravé straně pravidla, na stranu levou ve snaze přepsat celou větu na počáteční symbol (neterminál) gramatiky (provádíme tedy tzv. *redukci*).

U analýzy shora-dolu jde o přístup zcela opačný, začínáme vždy s přepisem počátečního neterminálu gramatiky na straně levé za pravou stranu pravidla ve snaze získat postupnou aplikací pravidel celou větu (provádíme tedy tzv. *expanzi*). Následuje krátký příklad bezkontextové gramatiky použité pro analýzu testovaného vstupu.

Příklad 5.1: demonstrace syntaktické analýzy, mějme dána tato pravidla bezkontextové gramatiky:

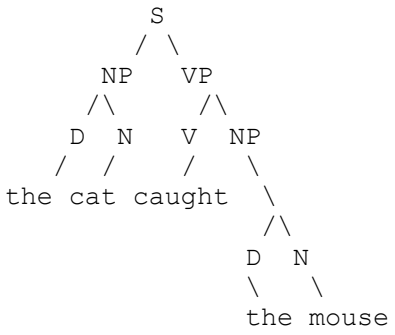
$S \rightarrow NP\ VP$
$NP \rightarrow N$
$NP \rightarrow D\ N$
$VP \rightarrow V$
$VP \rightarrow V\ NP$
$V \rightarrow$ caught |…
$D \rightarrow$ a | the
$N \rightarrow$ cat | mouse |…

Pokračování příkladu 5.1: použitím této gramatiky lze například generovat syntakticky správnou anglickou větu: „*The cat cought the mouse*.“

Posloupnost derivačních kroků (při nejlevější derivací) je pak následující:

$$S \Rightarrow NP\ VP \Rightarrow D\ N\ VP \Rightarrow the\ N\ VP \Rightarrow the\ cat\ VP \Rightarrow the\ cat\ V\ NP \Rightarrow$$
$$the\ cat\ caught\ NP \Rightarrow the\ cat\ caught\ D\ N \Rightarrow the\ cat\ caught\ the\ N \Rightarrow the\ cat\ caught\ the\ mouse$$

Pokud vyjádříme tuto derivaci pomocí derivačního stromu, dostaneme hierarchii na obr. 5.1.



Obr. 5.1 – derivační strom pro testovanou větu

5.1.3.3 Tvorba parseru – přístupy

Důležitým úkolem, který musí každý parser během své činnosti rozhodovat je „*výběr vhodného pravidla*“ pro reprezentaci dalšího derivačního kroku. Tento zcela přirozený požadavek vedl k vývoji různých algoritmů pro návrh například tzv. rozhodovacích tabulek, které nám deterministicky určují, jaké pravidlo gramatiky se má v daném okamžiku k aplikaci vybrat.

Pro přehled zmíníme některé dnes základní metody pro parsování (dobře známé také z oblasti návrhu překladačů): *rekurzivní sestup*, *prediktivní parsing* - oba jako implementace *LL* (*left to right leftmost*) analýzy shora dolu; nebo *operátor-precedenční parser*, *LR* (*left to right rightmost*) *parser* - oba jako implementace analýzy zdola-nahoru; (podrobnosti viz [15]).

I. Chart Parser

Z obecného pohledu nejsou však výše zmíněné metody pro zpracování jazyků přirozených v syntaktické rovině vždy zcela ideální. Prostý parser pracující zdola-nahoru (někdy pro své operace označovaný jako *shift-reduce*) nemusí být schopen nalézt rozbor (*parse*) dokonce i pokud existuje. Parser s rekurzivním sestupem může být velmi neefektivní, pokud vytváří/ruší mnoho stejných podstruktur; dalším podmínkou je pak odstranění levé rekurze, jejíž přítomnost by mohla vést k zacyklení.

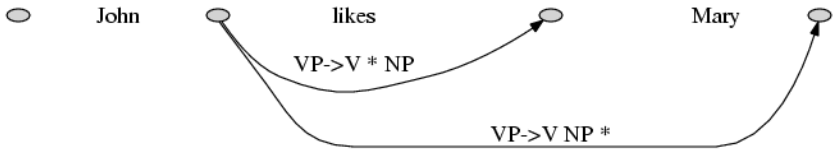
Aby bylo možné při návrhu a následné implementaci dosáhnout požadované úplnosti a efektivity, byla pro oblast zpracování jazyků přirozených navržena metoda parsování využívající techniku tzv. *dynamického programování*, která umožňuje uchovávání mezivýsledků a jejich znovupoužití ve vhodném okamžiku. Na tomto principu je založen tzv. *chart parser*. Vzhledem k jeho častému využití

jako hlavního parseru v oblasti zpracování přirozených jazyků si rozebereme princip tohoto analyzátoru blíže.

Chart parser pracuje se strukturou označovanou jako „*chart*“ potřebnou k uchování informací o předpokládaných větných částech. Jednou z grafických reprezentací této struktury je její možný zápis grafem (viz obr. 5.2). Záznamy jsou ve struktuře vyjádřeny tzv. *tečkovanými hranami* (*dotted edges*).

Tečkovaná hrana tvaru: $[A \rightarrow c_1 \ldots c_d; \bullet c_{d+1} \ldots c_n]@[i:j]$ reprezentuje předpoklady, kde část typu A (vyjádřena uzlem) zahrnuje synovské uzly c_1 až c_d s nimiž pokrývá slova w_i až w_j, zatímco následné synovské uzly v rozsahu c_{d+1} až c_n nebyly slovy dosud pokryty. Hrana, která ve svém typu obsahuje jen pokryté syny, se nazývá úplná jinak neúplná.

V následujícím příkladu grafu (obr. 5.2) hrana $[VP \rightarrow V\ NP \bullet]@[1:3]$ tvoří tzv. úplnou hranu, zatímco $[VP \rightarrow V \bullet NP]@[1:2]$ vytváří hranu neúplnou.



Obr. 5.2 – chart vyjádřený grafem včetně příkladu tečkovaných hran

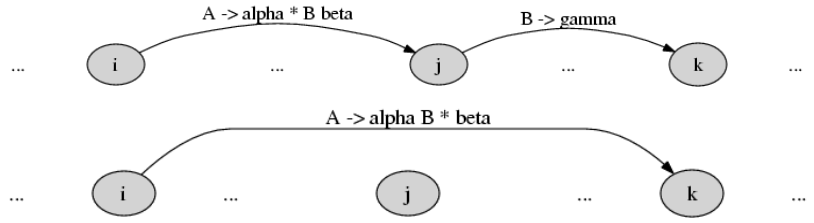
K tomu, abychom však byly schopni vytvářet záznamy (hrany) pro hlavní strukturu (*chart*), je zapotřebí seznámit se s povinnými pravidly, která tvoří základ techniky dynamického programování pro chart parser.

Nad vstupní větou je nejprve vytvořena prázdná struktura. Následně mohou být na základě vstupní znalosti (tj. analyzované věty, gramatických charakteristik - korpusu a bezkontextové gramatiky) vytvářeny ve struktuře jednotlivé hrany a to třemi možnými způsoby:

1. *Vstupní věta vytváří hrany*. Přesněji, každé slovo w_i z věty může přidat do struktury hranu ve tvaru: $[w_i \rightarrow \bullet\ ]@[i:i+1]$.
2. *Bezkontextová gramatika vytváří hrany*. Přesněji, každé pravidlo gramatiky tvaru $A \rightarrow \alpha$ může přidat do struktury sebesměrnou hranu ve tvaru: $[A \rightarrow \bullet\ \alpha\ ]@[i:i]$ a pro všechna i platí: $0 \leq i < n$.
3. *Aktuální obsah struktury* může vést k vytvoření nové hrany.

Zpravidla však nepotřebujeme během činnosti parseru pracovat se všemi hranami, které by šlo do struktury přidat. Chart parser z tohoto důvodu obsahuje také sadu pravidel, označovaných jako *strategie*, která heuristicky rozhodují, kdy má být daná hrana do struktury doplněna.

V neposlední řadě je součást každého chart parseru tzv. *fundamentální pravidlo*. Toto pravidlo umožňuje vytvářet nové hrany spojením hran současných a to v případě kdy neúplná hrana vyžaduje pro první nepokrytý synovský uzel jistý neterminál, který se nachází na levé straně hrany, jež po ní bezprostředně následuje (obr. 5.3). Ilustrujme použití fundamentálního pravidla například následující situací.

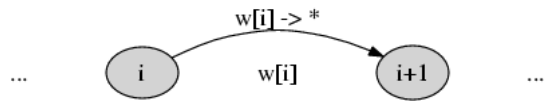


Obr. 5.3 – aplikace fundamentálního pravidla

Pro hrany $[A \rightarrow \alpha \bullet B\ \beta\ ]@[i{:}j]$ a $[B \rightarrow \gamma \bullet\ ]@[j{:}k]$ je na základě fundamentálního pravidla do struktury přidána nová hrana $[A \rightarrow \alpha\ B \bullet \beta\ ]@[i{:}k]$.

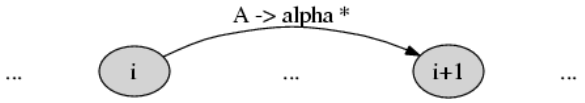
II. Chart parser jako analyzátor zdola-nahoru

K tomu abychom mohli nyní vytvořit úplný chart parser, zvolíme nejprve jednu z metod pro směr analyzátoru, v tomto případě zdola-nahoru (*bottom-up*). Ke stávajícímu fundamentálnímu pravidlu navíc doplníme tzv. *buttom-up inicializační* pravidlo a *buttom-up prediktivní* pravidlo. Pravidlo *inicializační* nám umožní přidat do struktury všechny hrany, které mohou být tvořeny vstupní větou, přesněji jejími slovy, viz obr. 5.4.



Obr. 5.4 – tvar inicializačního pravidla nad slovem w_i: $[w_i; \rightarrow \bullet\ ]@[i{:}i{+}1]$

Pravidlo *prediktivní* nám umožní přidat hranu, pokud je některá stávající hrana úplná pokrývající větu v rozsahu i až $i + 1$ a existuje pravidlo v bezkontextové gramatice, jehož pravá strana začíná s totožným neterminálem, jako se vyskytuje na levé straně této hrany, pak přidej sebesměrnou hranu (tedy v rozsahu od i po i), podrobněji viz obr. 5.5.



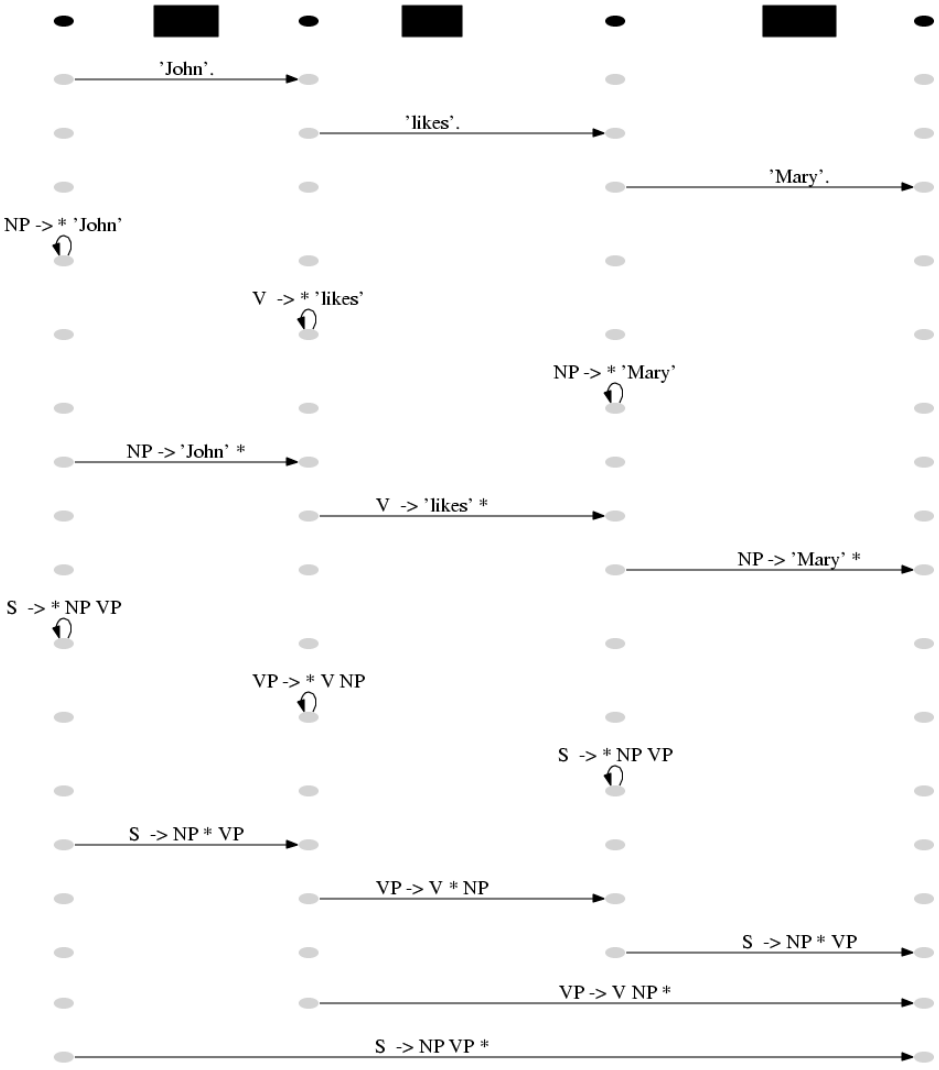
Struktura obsahuje úplnou hranu: [$A \rightarrow \alpha \bullet$]@[i:j] a bezkontextová gramatika obsahuje pravidlo: $B \rightarrow A\ \beta$, pak přidej sebesměrnou hranu [$B \rightarrow \bullet\ \beta$]@[i:i], tedy:

Obr. 5.5 – aplikace prediktivního pravidla

S využitím doplněného inicializačního a prediktivního pravidla, jsme nyní schopni popsat algoritmus chart parseru pro analýzu zdola-nahoru následovně:

1. Vytvoř prázdnou strukturu pokrývající vstupní větu.
2. Aplikuj inicializační pravidlo na každé slovo věty.
3. Dokud lze přidat hranu dělej:
    1. Proveď aplikaci prediktivního pravidla všude, kde je to možné.
    2. Proveď aplikaci fundamentálního pravidla všude, kde je to možné.
4. Vrať všechny nalezené stromy parseru (*parse trees*), které odpovídají hranám ve struktuře.

Ukázku úplné analýzy věty s využitím chart parseru jako analyzátoru zdola-nahoru pak demonstruje příklad na obr. 5.6.



Obr. 5.6 – syntaktická analýza věty *chart parserem*.

Výše zmíněnou metodu pro analyzátor zdola-nahoru bychom mohli také analogicky upravit na metodu shora-dolu. Inovační přístup chart parseru vedl také k vývoji nových metod z něj odvozených, jen výčtem zmíníme například nejznámější: *Viterbiho algoritmus*, *Earlyho algoritmus* nebo *Cocke-Kasami-Younger* (*CKY*) algoritmus (podrobnosti viz [16]). Zastoupení chart parseru a jeho metod představuje ve zpracování přirozených jazyků základ pro tvorbu analyzátorů nejen v oblasti vědeckého výzkumu ale zejména i pro koncový (komerčně vyvíjený) software.

5.2 Návrh metod pro zpracování přirozených jazyků v aplikaci multigramatik

Jazyk je základním dorozumívacím prostředkem mezi lidmi a případně dalšími subjekty. Pokud mluvíme o komunikaci mezi lidmi navzájem, jde zpravidla o určitou formu *přirozeného jazyka*. Většinu těchto jazyků můžeme považovat za důsledek postupného vývoje komunikačních signálů ve snaze vzájemného porozumění. K ustálení současného stavu takového jazyka bylo zapotřebí několik tisíc let vývoje, přičemž jeho forma se stále mění v důsledku historických zvratů a potřeb lidstva. Jak je výše naznačeno, pravidla přirozených jazyků jsou tedy až přímým výsledkem jejich vývoje.

Avšak jazyk, který nebyl determinován svým vývojem, ale jehož pravidla byla nejprve stanovena a až následně došlo k samotné aplikaci jazyka, označujeme za *jazyk umělý*. Tato skupina jazyků zahrnuje velkou množinu dorozumívacích prostředků, kterou si člověk vytvořil pro usnadnění života a v důsledku technického pokroku. Další významnou podmnožinou umělých jazyků je skupina, která byla vytvořena pro napodobení nebo simulaci jazyků přirozených. Tyto jazyky jsou někdy označovány jako „*pseudo-přirozené*“ (plánované). Jejich cílem je odstranit některé nedostatky nebo složité jazykové konstrukce jazyků přirozených (učí se z chyb svých předků). Snahou je také mnohdy zavedení jednotného dorozumívacího prostředku, nebo jen sloučením jejich společných prvků ve vzory tzv. „*universalia*“.

U současných jazyků přirozených narážíme na různorodost zejména ve stavbě slov, jejich vztahu k morfologii, rozdílná pravidla ve slovosledu, syntaxi vět. Abychom se oprostili od rozdílů mezi současnými různými jazyky, a případně jejich některých dalších nedostatků budeme od této chvíle ve všech rovinách zpracování uvažovat hypotetický ***abstraktní plánovaný jazyk*** (*abstract plan language*, dále jen **APL**). Tento jazyk bude navrhován postupně s průběhem popisu jednotlivých rovin zpracování. Na každé z nich budou nejprve uvedeny vlastnosti (meze), kterých mohou v dané rovině jazyky přirozené obecně nabývat a následně budou pro APL stanoveny vlastnosti s vhodnými parametry vzhledem k dané rovině. V poslední fázi pak budou vlastnosti APL demonstrovány.

Pozn.: návrh nového jazyka (APL), byť abstraktního, značně překračuje rozsah a primární cíle této práce. Navzdory této skutečnosti jsme si vědomi potřeby informovat čtenáře o podstatných souvislostech mezi přístupem lingvistického náhledu (popisu) a zamýšleným zpracováním (analýzou) jazyka výpočetní technikou při aplikaci multigramatik. Aby bylo přesto možné takový návrh uskutečnit a zachovat v něm objektivní přístup, jsou v návrhu APL využity také poznatky z dlouholetých studií jazykovědce Richarda A. Morneau (viz [17]).

5.2.1 Lexikální rovina

S ohledem na návrh hypotetického jazyka APL budeme potřebovat v lexikální reprezentaci textu zavést způsob kódování pro speciální znaky. Množina těchto znaků zahrnuje také grafémy, jako jsou například čínské ideogramy (viz [18]) a jiné glyfy, které se napříč rodinou jazyků přirozených vyskytují. Z tohoto pohledu lze uvažovat o způsobu mezinárodního kódování, kterým je například standard UTF-8 (*Unicode transformation format of ISO 10646*). Ten vychází ze stejného základu jako kódování Unicode (viz [19]).

Pozn.: zatímco Unicode kóduje všechny znaky na 16 bitech, v případě UTF-8 se jedná o kód s proměnnou délkou znaku. Běžné znaky (znaky anglické abecedy) jsou kódovány jen na 8 bitech (dtto ASCII) a akcentované na 16 případně 24 bitech. Dochází tak k úspoře velikosti zapsaného textu a zároveň zde odpadá problém „*Little versus Big Endian*“ na 8 bitových souborových systémech.

Jak již bylo zmíněno v kapitole 5.1.1, cílem lexikálního analyzátoru je získání posloupnosti lexémů (samostatných lexikálních jednotek) v konečné formě ohodnocených příznaky, které jsou nositeli informace o lexikální struktuře (uspořádání) těchto jednotek. Lexémy jsou po svém ohodnocení postoupeny na vstup roviny morfologické zde již jako tzv. „*tokeny“*.

Pozn.: toto odlišné znační jednotek zpracování v odpovídajících rovinách nám pomáhá lépe vyjadřovat jednoznačnou příslušnost jednotky k dané vrstvě - podobně jako například v oblasti síťových architektur je datová jednotka na jednotlivých logických vrstvách sítě označována nejprve jako rámec, datagram, paket a následně zpráva.

Z pohledu koncepčního návrhu lze systém lexikální analýzy zvolený pro APL rozdělit do několika logických úrovní:

1. načtení vstupního textu (dle zvoleného kódování)
2. preprocesor (*filter* - příprava textu pro lexikální analyzátor)
3. vlastní lexikální analýza (*scanner*)
4. ohodnocení lexémů jejich lexikálními příznaky (*evaluator* + *indexing tagset*)

5.2.1.1 Načtení vstupního textu

Na vstupu předpokládáme elektronickou podobu textu při vhodně zvoleném kódování (UTF-8). Pokud by se text vyskytoval v jiné podobě (zejména papírová forma dokumentu) je nejprve nutné provést rozpoznání jednotlivých znaků a jejich reprezentaci vyjádřit elektronickým textovým dokumentem. Pro strojový převod textu je možné využít HW scanneru s aplikací *OCR* (*optical character recognition*) algoritmů. Tato problematika je také osobně dotčena v rámci projektu týmové studie: „*Senzory scannerů v aplikaci OCR“* (podrobnosti viz [20]).

Pozn.: proces načítání textu je zpravidla prováděn sekvenčně. S rozvojem hardwarové podpory pro paralelní aplikace (rozumějme například rozšíření vícejádrových procesorů i na poli klientských PC), lze uvažovat i o paralelizaci této úlohy. A to kupříkladu návrhem algoritmů pro načítání textu více směrně (proti sobě), případně zároveň v určitých definovaných částech.

5.2.1.2 Preprocesor - filter

Úkolem preprocesoru je v průběhu načítání provádět filtrování znaků. Do množiny přípustných znaků pak patří:

- abeceda APL jazyka.
- vyjádření čísel.
- větné separátní znaky.
- slovní separátory.
- případně další typografické znaky pro strukturaci textu.

Ostatní znaky jsou z textu odstraněny. Dále jsou provedeny nad textem tyto operace:

- bílé znaky se redukují (pokud mají vícenásobný výskyt).
- všechna číselná vyjádření jsou nahrazena symbolickou hodnotou.
- texty komentářů jsou vypuštěny.
- možná reprezentace některých znaků (např. akcentovaných) jejich víceznakovou formou, nebo jiným kódem.

Pokud se případně někde v textu vyskytují odkazy na vložení jiné části textu, je vhodné provést tato rozvinutí také v rámci preprocesoru. Takto upravený text je připraven k postoupení lexikálnímu analyzátoru.

5.2.1.3 Lexikální analyzátor - scanner

Nad textem, který byl v průběhu načítání preprocesorem zachován lze zavést rozpoznání lexikálních částí. Rozeznáváme rozpoznávání jednotlivých vět a v rámci jejich obsahu pak posloupnosti lexémů. Abychom mohli tyto textové části odlišit, je nejprve nezbytné zavést konvence pro jejich dělení. Zde vycházíme z typografických zásad pro tvorbu textových dokumentů. Věta je dána jako posloupnost lexémů a platí:

1) je ukončena některým z větných separátních znaků (musí být splněno vždy).
2) před separátním znakem může být libovolný počet bílých znaků, ovšem za separátním znakem musí být alespoň jeden bílý znak (s výjimkou poslední věty).
3) pokud v jazyce rozlišujeme velké a malé znaky pak první znak věty by měl začínat velkým písmenem (s výjimkou je-li před začátkem této věty doposud poslední věta ukončena znaky: **,**' **;**' **:**').

Podmínka 1) je považován za nezbytné kritérium pro určení věty. Při nesplnění této podmínky lze rozpoznané věty považovat za nesprávně utvořené a vyřadit z dalšího zpracování. Pokud je věta korektně utvořena můžeme uvažovat o rozpoznání jednotlivých lexémů. Není-li stanoveno jinak, považujeme za slovní separátory (mezi lexémy) množinu bílých znaků. Posléze jsou lexémy ohodnoceny, viz následná část.

5.2.1.4 Ohodnocení lexémů - rating

Pro každý rozpoznaný lexém stanovíme nejprve jeho lexikální typ (odlišný od typů na rovině morfologické). Pro většinu lexémů je lexikálním typem „*slovo*“, případně „*speciální textový výraz*“. Typ *slov*o představuje neporušenou posloupnost grafických znaků hlásek, podrobnosti viz syntaxe slov v kapitole 5.2.2. Tuto část ohodnocení (*rating*) označujeme obecně jako tzv. „*evaluační*“.

Následuje tzv. *indexační* část, jejímž úkolem je detekovat (indexovat) jednotlivé lexémy dle jejich lexikálního postavení ve větě. Pro potřeby požadované na následných rovinách zpracování lze rozeznávat tyto příznaky:

- *snum* (*sentence number*) : udává číslo věty v textu, pro daný lexém
- *position* : udává pozici lexému ve větě
- *howmany* : udává počet lexémů ve větě
- *length* : délka lexému
- *word: vlastní* obsah
- *separ* : použitý větný separátor věty, k níž lexém náleží

Ohodnocené lexémy jsou nyní připraveny na vstup morfologické analýzy. Detekce tzv. lemmatu je uvažována až v rámci morfologického zpracování na základě známé slovní syntaxe, tedy až na úrovni dostupné znalosti o jednotlivých morfémech slov.

5.2.2 Morfologická rovina

V předešlé kapitole 5.1.2 byly zmíněny hlavní požadavky, které jsou kladeny během části morfologického zpracování. V úvodu do této problematiky zde byla také snaha, představit nároky na vstupní znalost, s jejíž pomocí lze pak při využití sady pravidel stanovit potřebné gramatické charakteristiky.

V případě našeho návrhu budeme uvažovat o morfologii z jiného pohledu. Potřeba získání gramatických charakteristik je zde stále nezbytná, avšak snahou bude dosáhnout tohoto cíle bez vstupních znalostních databází (korpusů). Z původního modelu zůstanou tedy využity jen sady pravidel, které lze algoritmizovat.

Abychom však mohli takové restrikce zavést, bude nutné provést rozbor samotných slov, nad nimiž chceme charakteristiky získat. První fáze rozboru spočívá v úvaze, zda slova disponují svou vlastní syntaxí. Pokud postoupíme v této úvaze dále, pak po studii stavby slov daného jazyka a užitím formálního vyjadřovacího prostředku získáme přesný popis syntaxe pro slova samotná. V případě, kdy je tato slovní syntaxe již známa, je možné přistoupit k další fázi, která zkoumá pravidla této syntaxe a jejich vztah ke hledaným gramatickým charakteristikám. Vhodnost užití tohoto přístupu je ovšem závislá na zvoleném jazyce a zejména na vztahu syntaxe jeho slov k morfologii.

5.2.2.1 Návrh stavby slov pro APL

Základem pro tvorbu slov je slabika, v rodině jazyků přirozených se pak každé slovo skládá z jedné až více slabik. Z pohledu teoretické informatiky budeme množinu slabik daného jazyka chápat jako podmnožinu regulárních jazyků (množina slabik je popsatelná pomocí formálního prostředku - regulárních výrazů). Avšak ani definice slabik není mezi jazyky jednotná. Uvedeme proto nejprve některé možné varianty a jejich vlastnosti.

Každá slabika je složena ze symbolů, které budeme nazývat hláskami (pro každou hlásku je možné definovat jeden jednoznačný symbol). Dále je třeba rozeznávat jednotlivé kategorie hlásek. Pro popis těchto kategorií zavedeme následující množinová značení:

C – množina souhlásek
V – množina samohlásek
S – množina polosamohlásek
N – množina nosových hlásek

Regulární výraz popisující množinu slabik budeme značit r a jazyk daný tímto výrazem pak jako $L(r)$.

1. **tvar slabik** – komplexní forma:

$r = (c^*ww^*c^*)$, kde $c \in C$ a $w \in (V \cup S)$

Pozn.: existuje jen velmi málo jazyků, které plně využívají schopností lidského řečového ústrojí.

2. **tvar slabik** – zjednodušená forma, ovšem častá:

$r = (c?s?vv?s?n?)$, kde $c \in C$, $v \in V$, $s \in S$ a $n \in N$

Tento tvar slabik nám například umožňuje vytvářet slabiky známé z anglického jazyka jako: "*him*", "*queen*", "*boa*" a "*toy*", ale již nevytvoříme slabiky jako: "*hit*", "*string*", "*plank*" či "*flirt*". Nedostatek souhlásek a přítomnost nosové hlásky jako konečné souhlásky zde na jedné straně omezuje velikost množiny možných slabik, avšak na straně druhé spojení takových slabik nečiní potíže při výslovnosti (nosová hláska je přirozeně vysoce vokální a tedy snadno vyslovitelná s předcházející samohláskou).

3. **tvar slabik** – forma vhodná pro APL, oproti předchozímu tvaru 2) zavedeme rozšíření, které bude představovat kompromis při zachování snadné výslovnosti pro většinu lidí.

$r = (c_1?s?vv?s?c_2?)$, kde $c_1 \in C$, $s \in S$, $v \in V$, $c_2 \in (C \cup N)$

V případě kdy při spojování slabik bude c_2 první slabiky identické s c_1 slabiky druhé, může vyvstat problém při výslovnosti jako např. ve spojení slabik „*dus san*“. Jako řešení se nabízí například možnost zkracování při výslovnosti, nebo lze c_1 považovat za nesouhlásku a zároveň za nenosálovou hlásku.

Jakmile máme stanovenu definici slabik, lze s jejich pomocí tvořit jednotlivé morfémy a z těch pak sestavovat celá slova. V této chvíli jsou obecně pro stavbu APL možné dva přístupy. První z nich je tzv. „*ad hoc*“, kde jednotlivé morfémy převezmeme z již existujících jazyků (tento přístup byl například použit také při návrhu jazyka Esperanto). Druhý více formální přístup počítá s odlišným tvarem morfémů dle jejich postavení (funkce) ve slově. Přepona, přípona i kořen jsou pak v rámci každého slova jednoznačně identifikovatelné bez nutnosti pomocných oddělovacích prvků. Dostáváme tak slova, která se skládají jen ze *sebe-separátních* morfémů, což je zejména při počítačovém zpracování jazyků vhodné (ve zpracování řeči přirozených jazyků, které nejsou sebe-separátní, se problém separace morfémů jeví jako jeden z nejobtížnějších).

Následovat bude postupný návrh pro APL, kde zvolíme druhý zmíněný přístup, tedy jazyk se sebe-separátní morfologií. Získáme tak tři základní typy morfémů, které budou mít jednoznačný tvar a budou se dle své funkce nacházet na odpovídající pozici ve slově. Uvažme nejprve následující prostý příklad (každý morfém je jednoslabičný), varianta A.

A.

$C = \{b, p, d, t, g, k, z, s, v, f, r\}$

$V = \{a, e, i, o, u\}$

$S = \{y, w\}$

$N = \{m, n\}$

předpona $\in L(r_p)$, kde $r_p = (csv)$ a $c \in C, s \in S, v \in V$

kořen $\in L(r_r)$, kde $r_r = (cvn)$ a $c \in C, v \in V, n \in N$

přípona $\in L(r_s)$, kde $r_s = (cv)$ a $c \in C, v \in V$

slovo $\in L(r_w)$, kde $r_w = (předpona^* kořen^* přípona?)$

Následují příklady slov, které můžeme za pomocí tohoto příkladu tvořit: ***ze (přípona), te (přípona), kemto (kořen-přípona), pangu (kořen-přípona), grake (předpona-přípona), krosenfi (předpona-kořen-přípona), vyusintamku (předpona-kořen-kořen-přípona), ryukwesu (předpona-předpona-přípona)***, atd. sebeseparátnost morfémů je zde bezesporná. V tomto jednouchém příkladě je však možno vytvořit jen 100 různých kořenů (10 x 5 x 2 kombinací). Pro odstranění tohoto omezení lze zavést víceslabičný morfém – kořen, viz následující model B.

B.

$C = \{b, p, d, t, g, k, z, s, v, f, r\}$

$V = \{a, e, i, o, u\}$

$S = \{y, w\}$

$N = \{m, n\}$

$EXTRA = \{q$ (čti ‚č‘), x (čti ‚š‘)$\}$

předpona $\in L(r_p)$, kde $r_p = (csv)$ a $c \in C, s \in S, v \in V$

kořen $\in L(r_r)$, kde $r_r = ((cvn)+(cvn?qvn?)+(cvn?xvn?cvn?))$

a $c \in C, v \in V, n \in N; g, x \in EXTRA$

přípona $\in L(r_s)$, kde $r_s = (vc)$ a $c \in C, v \in V$

koncovka $\in L(r_f)$, kde $r_f = (c?v)$ a $c \in C, v \in V$

slovo $\in L(r_w)$, kde $r_w = (předpona^* kořen^* přípona? koncovka)$

Extra hlásky ‚*q*‘ a ‚*x*‘ nám indikují víceslabičnost kořene. Zde zavedená fonetika pro tyto extra hlásky nám usnadňuje výslovnost mezi víceslabičnými návaznostmi. Příklady tak vytvořených dvouslabičných kořenů mohou být následující: ***vinqam (čti ‚vinčam‘), penqu (čti ‚penču‘)***, ***zeqan (čti ‚zečan‘)***. Trojslabičné kořeny pak například tyto: ***guxita (čti ‚gušita‘), rixemsi (čti ‚rišemsi‘), kunxari (čti ‚kunšari‘), femxaren (čti ‚femšaren‘)***.

Po spojení s morfémy jako předpona, přípona a koncovka pak dostáváme například tato slova: ***pweti-mqanku (čti ‚pwetimčanku'), ginqase (čti ‚ginčase'), tyoseqanke (čti ‚tyosečanke'), guxiroka (čti ‚guširoka'), kyotaxensike (čti ‚kyotašensike'), zonxedati (čti ‚zonšedeti')***, nebo ***ramxukonesgi (čti ‚ramšukonesgi')***. V případě dvojslabičných kořenů tak získáme 2250 různých (250 + 2 x 500 + 1000). U trojslabičných pak dokonce 337 500 různých kořenů. Slova takto tvořená nelze rozhodně považovat za eurocentrická.

Takto popsaný návrh představuje demonstraci formálního přístupu s ohledem na potřebné vlastnosti. V případě jiných požadavků, lze vhodně upravit jak samotné slabiky, tak pravidla pro tvorbu morfémů.

5.2.2.2 Vztah slovní syntax k morfologii

Zde nejprve vyjdeme z přirozeného požadavku, kterým je schopnost odvozovat další slova pomocí kombinace zavedených primitiv. Tato primitiva budou pro nás představovat výše definované morfémy. Zatímco jsme slova v předešlé sekci tvořili jen dle povoleného pořadí morfémů ve slově, nyní budeme navíc požadovat, aby nám určitá povolená kombinace vyjadřovala jednoznačně svou stavbou některé informace potřebné pro stanovení gramatických charakteristik.

Pokud je navíc určitý morfém (například kořen) víceslabičný, lze uvažovat i o pravidlech pro odvozování víceslabičné formy od jednoslabičné. Zaměříme se však zejména na systém vazeb v rámci celého slova. Tento systém, kdy slova získávají skutečný význam lze také chápat jako mapovací funkci (vzájemně jednoznačné zobrazení) z nám známých skutečností na generovaná značení (slova). Systémy jsou mezi přirozenými jazyky zpravidla různé a často zahrnují řady výjimek, které mohou postrádat i jakoukoli logickou souvislost. V plánovaném jazyce Esperanto byl proto navržen systém na základě „*pevných*" pravidel, která nám nejen usnadňují samotný proces poznání slovní zásoby takového jazyka, ale jsou zejména důležitá pro stanovení jednoznačných vazeb mezi slovní syntaxí a hledanou gramatickou charakteristikou. Obdobný způsob bude také využit v případě návrhu pro APL. Základ takového systému pak představuje následující vztah:

$$slovo \in L(r_m),\ r_m = (modifikátor^*\ kořenová\_část\ klasifikátor?\ identifikátor).$$

Funkce jednotlivých částí jsou pak následující:

- *modifikátor* – upřesňuje význam kořenové části, vyskytuje se na pozici morfému předpony
- *kořenová_část* – základní význam slova, vyskytuje se na pozici morfému kořene
- *klasifikátor* – vyjadřuje zařazení kořenové_části v určité hierarchii, vyskytuje se na pozici morfému přípony
- *identifikátor* – jednoznačně identifikuje slovní druh a případně další informace, které k němu přísluší, vyskytuje se na pozici morfému koncovky

Demonstraci systému předvedeme na následném nově vytvořeném slově. Nechť je dán jednoslabičný kořen „*don*“, který představuje označení pojmu „*vlk*“. Dále zavedeme modifikátor „*fya*“ pro označení „zdrobněliny“, klasifikátor „*eg*“ určující zařazení tvora do kategorie „*bájné bytosti*“ a koncovku „*o*“ pro označení „*podstatného jména v čísle jednotném*“. Dle zavedeného systému pak dostáváme výsledné slovo „*fyadonego*“, které představuje významově pojem „*zdrobněliny vlkodlaka*“.

Nejdůležitějším gramatickou charakteristikou je bezesporu slovní druh. V návrhu APL lze například uvažovat poslední hlásku v identifikátoru (morfém koncovky) za jednoznačné typové označení (slovní druh). Navíc na rozdíl od jazykového členění, kde rozeznáváme deset slovních druhů, je přístup vyžadovaný při jazykovém zpracování odlišný. Určité slovní druhy mohou zastávat postavení ve větě identicky. Během analýzy není proto důležitá skutečná jazyková podstata o slovním druhu nýbrž jeho chování. Některé slovní druhy lze pak považovat z tohoto pohledu jako jeden typ. Posloupnost takto rozpoznaných slovních druhů, bude následně představovat základní vstup pro zpracování na navazující rovině syntaktické.

5.2.3 Syntaktická rovina

Než přikročíme k vlastnímu návrhu aplikace multigramatik v syntaktické analýze při zpracování přirozených jazyků v aplikaci multigramatik, budou nejprve uvedeny doprovodné metody a poznatky, jejichž důsledky jsou pro aplikace multigramatik na této rovině relevantní (nezbytné také pro dokončení návrhu APL).

5.2.3.1 Závislostní struktury

V rodině jazyků přirozených existuje několik reprezentantů, modelů pro závislostní vztahy mezi základními větnými členy jako je podmět (*S*), přísudek (*V*) a předmět (*O*). Ačkoli existuje nezanedbatelné množství jazyků, které nám dávají možnost přecházet z jednoho modelu na druhý, existují také vždy určitá omezení, respektive tendence inklinovat k určitému modelu ve větší míře. Z tohoto předpokladu pak vycházejí i následující poznatky. Dostáváme tak napříč všemi jazyky šest modelů pro popis vět dle postavení základních větných členů: VSO, SVO, SOV, VOS, OVS a OSV. Následuje přehled řady přirozených jazyků a jejich zařazení k jednotným modelům.

SOV - Turečtina, Tamilština, Japonština, Tibetština, Kečuánština
Tato skupina pokrývá zhruba 40% jazyků.
Příklad věty: *Pavel knihu čte*.

SVO - Angličtina, Čeština, Swahilština, Čínština, Indonézština
Pokrytí nižší než SOV, přesto zahrnuje také téměř 40% jazyků.
Příklad věty: *Pavel čte knihu*.

VSO - Velština, Hawaiština, Berberština, Klasická Arabština
Pokrývá přibližně 15% jazyků
Příklad věty: *Čte Pavel knihu*.

V případě výše třech uvedených se podmět vyskytuje vždy před předmětem a mění se jen pozice přísudku. Zbývající tři modely pokrývají relativně již jen malé množství jazyků.

OVS - Guarijio (Aztécko-Tanoanáská rodina, Mexiko)
Hixkaryana (Karibská oblast, Brazílie)
Klingonština (plánovaný jazyk ze sci-fi série Star Trek)
Příklad věty: *Knihu čte Pavel*.

VOS - Fidžijština (Austro-oceánská rodina, Fidži)
Terenáština (Arahuánská rodina, Brazílie)
Malagáština (Austronesijská rodina, Madagaskar)
Příklad věty: *Čte knihu Pavel*.

OSV - Jamamádština (Arahuánská rodina, Brazílie)
Jazyk Yodův (plánovaný jazyk mistra rytíře Jedi ve sci-fi sérii Hvězdné války)
Příklad věty: *Knihu Pavel čte*.

Pokud chceme zvolit nejrozšířenější model, pak lze hovořit o SOV, v případě eurocentrického přístupu pak SVO. Ovšem u návrhu jazyka vhodného pro počítačové zpracování je výhodnější neutrální model, kterým je VSO, tento model bude také uvažován v případě APL. Hlava věty bude tedy tvořena přísudkem a ostatní členy budou představovat její argumenty. Komplexní návrh bude zmíněn až v rámci části 5.2.3.3, tedy až po obeznámení s návrhem u struktur frázových.

5.2.3.2 Frázové struktury

V okamžiku, kdy je na základě (jako jeden z výstupů) morfologické analýzy známá příslušnost každého slova věty ke kategorii slovních druhů, lze uvést pravidla, jež nám pomohou pochopit vztahy mezi jednotlivými druhy.

Nejprve se zaměříme na tzv. substantivní fráze. Hlavní roli zde plní vztah mezi substantivy (***N***) a adjektivy (***A***), neboť společně tvoří tzv. *modifikátor* pro zmíněné fráze substantivní. V rodinách jazyků přirozených však nejsou jedinými modifikátory substantivních frází jen substantiva či adjektiva. K úplnému zachycení těchto typologických forem, které se mohou v běžných větách vyskytovat, je třeba vzít v úvahu i přítomnost tzv. *těžkých modifikátorů*, které také rozvíjejí podstatné jméno. Patří sem například věty vztažné (*relative clause*), prepoziční fráze nebo prepoziční argument zpodstatnělých (substantivních) verb. Tyto těžké modifikátory budeme souhrnně označovat symbolem ***R***, neboť z pohledu frázových struktur zastávají ve větě stejná postavení. Z typologického hlediska tak získáme následné případy těchto forem:

NAR - Thajština, Francouzština, Hebrejština, Swahilština
Příklad věty: *člověk dobrý, který žije sám*

ANR - Kečuánština, Angličtina, Čeština, Ruština, Perština
Příklad věty: *dobrý člověk, který žije sám*

ARN – žádný jazyk
Příklad věty: *dobrý sám žije který člověk*

NRA - žádný jazyk

Příklad věty: *člověk, který žije sám dobrý*

RNA - Baskičtina, Abcházština, Barmština

Příklad věty: *sám žije který člověk dobrý*

RAN - Turečtina, Tamilština, Korejština

Příklad věty: *sám žije který dobrý člověk*

Zajímavé je povšimnout si, že neexistují jazyky typu ARN nebo NRA jinými slovy tedy jazyky, které by vkládaly například věty vztažné mezi substantiva a adjektiva.

Dalším prvkem, který výrazně ovlivňuje substantivní fráze, jsou tzv. *specifikátory*. Tato skupina zahrnuje ukazovací, neurčitá, přivlastňovací pronomina a dále pak také členy určité (např. Esperanto) i neurčité (např. Angličtina), pokud je jimi daný jazyk vybaven. Označení specifikátor je dáno jejich smyslem pro omezení (výběru) skupiny, kterou v substantivní frázi vyjadřujeme. Mějme například větu: „*tito studenti jsou pilní*“, zde je specifikátorem ukazovací pronomino „*tito*“ (bez tohoto specifikátoru nabývá věta v části substantivní zcela jiný význam).

Z pohledu frázových struktur zastává postavení specifikátorů stejný význam jako adjektiva u modifikátorů. Avšak zatímco v případě adjektiv se u jazyků přirozených nevyskytují možnosti jako ARN a RNA, u specifikátorů formy jako SRN a NRS přípustné být mohou (např. Indonézština). Navíc se specifikátory mohou vyskytovat také před adjektivy (Čeština, Angličtina), či za nimi (Vietnamština), nebo v kombinaci obou zmíněných přístupů.

5.2.3.3 Komplexní návrh syntaxe pro APL

V této části kapitoly se budeme zabývat komplexním návrhem syntaxe APL při užití vlastností syntaxe jazyka demonstrovaných v částech předchozích.

Formálních prostředků pro popis syntaxe frázových struktur existuje několik druhů, lingvistickému pojetí bližší *X-Bar* teorie, jinou alternativou je například *Backus-Naurův* formalismus (BNF), nebo bezkontextové gramatiky z prostředí teoretické informatiky, návrhu kompilátoru.

Vzhledem k požadavku aplikace multigramatik v syntaktické rovině, budeme pracovat s modelem bližším bezkontextovým gramatikám. Z důvodu komplexnosti a efektivity zápisu však zvolíme model nad bezkontextovými gramatikami, který bude představovat kratší verzi zápisu (přehlednost) pro účely tohoto návrhu, bližší BNF. Tuto notaci označíme jako APLF.

Definice 5.1:
APLF je čtveřice (N, T, P, S), kde N je množina neterminálů, T je množina terminálů, S je startující neterminál a $S \in N$. P je konečná množina pravidel tvaru $x \rightarrow r$, kde $x \in N$, r je regulární výraz a platí: $L(r) \subseteq (T \cup N)^*$.

Pozn.: v příkladech, které budou pro názornost v průběhu návrhu prezentovány, si pro snazší pochopení syntaxe vět dočasně propůjčíme k APL již hotová slova z našeho rodného jazyka, jejichž význam je nám dobře znám.

Popis frázových struktur bude tedy zapsán formou APLF, přičemž jednotlivé frázové struktury budou respektovat závislostní model VSO. Neterminály budou zastoupeny výrazy začínajících velkým písmenem a terminály pak výrazy s malými písmeny. Počáteční neterminál bude pro větu reprezentován označením *Sentence*. Komplexní popis věty jazyka APL pak můžeme popsat jako:

Sentence → (*verb Verb_mod* Verb_arg* Sentence_partic**)
Verb_mod → (*adverb* + *tense_tag*)

Verb_mod zde představuje adverbia jako: ihned, pomalu, včera apod. Zároveň umožňuje vyjádřit změnu času predikátu pomocí příznaku *tense_tag* (pokud není tato změna zohledněna již přímo v koncovkách verb). *Sentence_partic* nám dává možnost změnit větu do formy tázací nebo rozkazovací podobně jako například tázací částice v Esperantu nebo Japonštině. *Verb_arg* pak vyjadřuje substantivní frázi, adjektivní frázi nebo větu vloženou.

Verb_arg → (*Case_tag*? *Expr Arg_partic**)
Expr → (*Noun_phrase* + *Adj_phrase* + *Sentence*)

Case_tag odpovídá předložkám pro uvození argumentu hlavy věty nebo k uvození vět vedlejší. Mějme například věty: „*Jde Petr **do** školy*.“ (uvození argumentu hlavy), „*Psal (on) práci, **když** pršelo*.“ (uvození věty vedlejší). *Arg_partic* plní podobnou úlohu jako *Sentence_partic*. Narozdíl od *Sentence_partic* je jeho rozsah vztažen jen k argumentu hlavy, kterou bezprostředně následuje. Podobně jako *Sentence_partic* může vyvolat ve větě důraz nebo ji změnit v tázací. Například ve větě „*Půjčil_si (on) knihu **kterou***?“ je *Arg_partic* vyjádřen tázacím pronominem.

Noun_Phrase → (*noun Simp_noun_mod* Comp_noun_mod**)
Simp_noun_mod → (*Adj_phrase* + *number* + *artikle* + *demon_pronoun* + *posses_pronoun* + *quant_pronoun*)

Comp_noun_mod → ((*Noun_phrase_tag Noun_phrase*) +
(*Noun_clause_tag Sentence*))
Adj_phrase → (*adjective Adjective_mod* Adjective_arg**
*Adjective_partic**)
Adjective_mod → (*adverb Adverb_partic**)
Adjective_arg → (*Adjective_phrase_tag Noun_phrase*)

Noun_phrase_tag odpovídá prepozicím jako například ve větě: „*Zadej* (*Ty*) *data* ***do*** *PC*.“. *Noun_clouse_tag* vyjadřuje vztažná pronomina uvozující věty vztažné. *Adjectice_arg* nám umožní tvořit věty jako: „*bílá s černou*“ nebo „*sténání v tichu*“. *Adverb_partic* podobně jako adjektiva bude plnit zesílení nebo zeslabení významu svého předchůdce.

Pomocí zde uvedených pravidel byla prezentována syntaxe vhodná pro návrh abstraktního jazyka, kterým je APL. Za povšimnutí stojí skutečnost, že takto tvořené věty nám dávají větší volnost ve tvorbě slovosledu vět podobně jako například Esperanto či Čeština ve srovnání s Angličtinou. V následné části této kapitoly uvedeme ještě některá možná rozšíření této syntaxe, která se mohou také v rodině jazyků přirozených vyskytnout.

5.2.3.4 Rozšíření syntaxe

Mimo pravidla zavedená v předešlé části budou nyní demonstrována některá vhodná rozšíření syntaxe pro přirozené jazyky.

1) **spojení částí vět** -umožní tvořit nové syntakticky správné věty spojením dvou libovolných větných prvků (popsaných v části předešlé) pomocí souřadných konjunkcí.

   Part → (*Part* (*coord_conjunction Part*)*)

Kde *Part* zastupuje libovolnou část věty a *coord_conjunction* je souřadnou konjunkcí. Vzniknout tak může například i tato věta: „*Šimon a Matouš*“.

2) **rozšíření významu částic** - přidáním významové částice k libovolnému větnému prvku lze změnit důraz nebo ukončit tento prvek.

   Part → (*Part extra_particle**)

3) **nahrazení morfologie v rovině větné syntax** - pokud existují v jazyce určitá morfologická omezení (pro tvorbu slov požadovaného významu), nebo chceme mít možnost i jiného přístupu, lze nového významu dosáhnout také spojením slova s přechodným příznakem a následovaného některým argumentem predikátu.

New_PartOfSpeech → (*PartOfSpeech_tag Verb_arg*)

Kde *PartOfSpeech* zastupuje libovolný slovní druh a *tag* je přechodovým příznakem. Příklad pravidla pro *PartOfSpeech,* kde *PartOfSpeech* = substantivum:

New_noun → (*noun_tag Verb_arg*)

Například nový význam slova „*nabídky*“ lze tak vytvořit ve větě: „*Čtu (Já) nové* ***nabídky_ke_studiu***“.

V závěru zavádění syntaxe pro APL uvedeme ještě některé příklady syntakticky správně tvořených vět dle APL v porovnání s jinými jazyky, viz následná tabulka 5.1.

Čeština	Angličtina	APL
Pavel šel domů.	*Pavel went home.*	*Šel Pavel domů.* | *Jít suq Pavel domů.* (*suq* = *Verb_mod* pro minulost)
Ukradl zloděj ten dopis?	*Did the burglar steal the letter*?	*Ukrást suq zloděj dopis ten cog*? (cog = *Sentence_partic* pro otázku)
Ten dopis je na stole.	*The letter is on the table.*	*Je dopis na stole ten.* | *Je_na dopis stole ten.* (*na* = *Case_tag*) (*je_na* = přechodná verba)
Utekl pryč ten člověk, který to způsobil ?	*Did the man who did it run away*?	*Utéct_pryč suq člověk ten který způsobit suq to cog*? (*který* = *Noun_clouse_tag*)
Pavel nerad vyplňuje úřední formuláře.	*Pavel hates filling official forms.*	*Nerad Pavel vyplňovat formulář úřední qes.* (*qes* = specifikátor pro plurál)

Tabulka 5.1 – příklady správných vět pro APL dle zavedené syntaxe

5.2.3.5 Aplikace multigramatik v syntaktické analýze

V okamžiku, kdy jsme schopni popsat syntaxi vět jazyka (například způsobem výše zmíněným) a jsou nám známi gramatické charakteristiky z roviny morfologické, lze provést analýzu větné syntaxe s užitím uvažovaných multigramatik. Aplikaci multigramatik budeme v syntaktické rovině pro zpracování jazyků přirozených chápat z několika úhlů pohledů. Zde uvedené aplikace nachází navíc uplatnění také v příbuzné oblasti, kterou je výstavba překladačů. Přehledově:

I. Rozšíření formálního popisu pro syntaxi jazyků přirozených
II. Prostředek pro zachycení kontextových vazeb věty
III. Podpůrný mechanismus k popisu deterministického parseru
IV. Formální (paralelní) systém za hranicí rodiny bezkontextových jazyků

I. Rozšíření formálního popisu syntaxe

Pokud se v případě určitého jazyka vyskytují závislosti mezi větnými částmi, či jejich vazby s jinými členy a nechceme nebo není možné tuto skutečnost postihnout běžným formálním prostředkem (jako například bezkontextovou gramatikou), je potřeba užít k tomuto účelu jiného prostředku (slovního popisu nevyjímaje). Navzdory pochopitelnosti vyjádření popisu neformálním způsobem (kdy na úkor exaktnosti opouštíme jazyk matematický) je pro strojové (počítačové) zpracování taková forma nepřípustná. Pokud uvážíme jiné formální prostředky (zmíněné např. v rámci části 5.2.3.3) je zde zřejmá absence vazeb mezi formálním mechanismem primárního a sekundárního charakteru a to jak mezi částmi popisu v rámci jedné roviny (např. syntaxe) tak i mezi více rovinami (např. syntax-sémantická vazba). Ilustrujme si tuto situaci na následující případové studii.

Mějme dáno bezkontextové pravidlo „$VP \rightarrow V\ NP$“, které nám vyjadřuje segment z frázové struktury pro anglické věty. Dále mějme například požadavek pomocí formálního prostředku popsat určitou sekvenci větných členů. Ten si vyjádříme jiným formalismem – regulárním výrazem jako: „*predicate object?*“. Zkoumáme zde tedy popis syntaxe ze dvou různých pohledů s užitím různých formalismů (pro každý volíme ten, který je k němu specifičtější), přičemž nám však schází prostředek, který by nám vyjádřil skutečnost, že v obou případech popisujeme tutéž část analyzované věty - tedy vazba mezi formálním mechanismem primárního a sekundárního charakteru.

Pozn.: pro snazší představu můžeme uvést přirovnání z oboru geografických informačních systémů. Zde mohou nad určitým geologickým terénem existovat různé mapové vrstvy, a zatímco tyto vrstvy zkoumají tutéž oblast, často také požadujeme vyjádřit například polohu z jedné vrstvy ve vrstvě druhé (toho lze dosáhnout např. zavedením společného souřadného systému nebo definicí transformací mezi nimi).

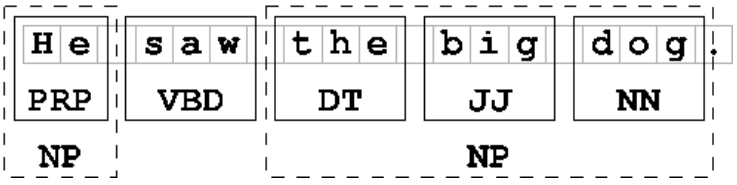
V případě multigramatik je první formalismus reprezentován upravenou bezkontextovou gramatikou a druhý je pak vyjádřen pomocí množiny regulárních výrazů – selektorů (viz definice v kapitole 2.1).

Přičemž je ale pevně stanoven vztah mezi těmito formalismy pomocí tzv. aktivních částí v selektorech, které jsou s pravidly gramatiky spjaty. Tím je dosaženo potřebného rozšíření v popisu syntaxe. Z obecného pohledu bychom pochopitelně nemuseli uvažovat jen o formalismech v multigramatikách a mohli zavést jiné prostředky a jejich vazby, které by lépe vyhovovaly našim požadavkům.

V souvislosti s multigramatikami se nabízí jedna možnost, a sice zobecnění pojmu „*aplikace pravidel povolených selektorem*“ na pojem „*volání obecných akcí povolených takto zavedeným selektorem*“. Podrobnosti k těmto otázkám viz kapitola 4. – „*Aplikace multigramatik v L – systémech*“.

Z pohledu širšího popisu syntaxe prostřednictvím multigramatik bychom zmínili také tzv. „*chunk parsing*“ (viz [21]). Chunk parsing je syntaktická analýza založená na extrakci větných částí, ignorování aktuálně nepodstatných celků k získání větných struktur určitého relevantního kontextu. Přičemž jsou k popisu pravidel pro hledaný výběr využity prostředky regulárních výrazů. Ačkoli se tato původní myšlenka může zdát být poněkud triviální, chunk parsing je schopen zavést robustní základ pro návrhu parseru při zachování efektivity (časová složitost je zde lineární). Vzhledem k prostředkům, které nám v rozšíření popisu syntaxe multigramatika poskytuje, uvedeme prostý příklad simulace chank parsingu pomocí rozšířených multigramatik (tato problematika také úzce souvisí s nadcházející částí II.).

Příklad 5.2: mějme dánu větu (obr. 5.7), tečkovaná část zde představuje frázové struktury vyšší úrovně, a pravidlo gramatiky $NP \rightarrow$ (<*DT*>?<*JJ*>*<*NN*>), dle notace APLF. K získání (identifikaci relevantního kontextu) v našem případě druhé substantivní fráze (*NP*) nám postačí zavedení selektoru ve tvaru: $.*<VBD>\overline{<DT>?<JJ>*<NN>}$, kde část výrazu .*<*VBD*> představuje libovolný pasivní levý kontext následovaný verbem a zbývající část pak aktivní prvek, který bude nahrazen za levou stranu zmíněného pravidla (analýza zdola-nahoru), tedy za frázi *NP*.



Obr. 5. 7. – simulace *chunk parsingu* pomocí multigramatik

II. Zachycení kontextových vazeb

Kontextovou vazbou rozumíme vztahy, které se mezi větnými částmi běžně vyskytují. V předchozí části již byl naznačen požadavek provedení určité akce (zde rozumějme provedení derivačního kroku) jen v případě, kdy jsou splněny požadavky, které na kontext větných částí klademe. V případě multigramatik lze tento kontext vyjádřit pomocí selektoru, kde *aktivními částmi* vyznačíme větné části, nad nimiž budou aplikována pravidla gramatiky a okolní (*pasivní*) *části* výrazu pak představují tzv. po-

vinný kontext, který se musí ve větě kolem přepisovaných částí vyskytovat. V případě, kdy kontext věty neodpovídá požadovanému předpisu v selektoru, nelze aplikaci pravidel provést, derivace je tak předčasně zablokována (končí neúspěchem) a lze vyvodit důsledky pro syntaktickou chybu ve větě. Pro ilustraci této problematiky demonstrujme následnou situaci.

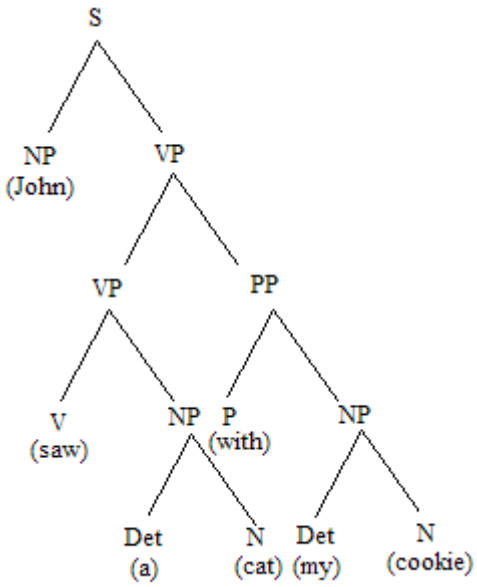
Příklad 5.3: nechť je dána věta: „*John saw a cat with my cookie*“ a fragment pravidel bezkontextové gramatiky nezbytný pro syntakticky správný popis této testované věty:

1. $S \rightarrow NP\ VP$
2. $VP \rightarrow V\ NP \mid VP\ PP$
3. $V \rightarrow$ "*saw*"
4. $NP \rightarrow$ "*John*" $\mid Det\ N \mid NP\ PP$
5. $Det \rightarrow$ "*a*" | "*my*"
6. $N \rightarrow$ "*cat*" | "*cookie*"
7. $PP \rightarrow P\ NP$
8. $P \rightarrow$ "*with*"

Uvažme analýzu zdola-nahoru nad touto větou. Derivační strom při požadované aplikaci pravidel je znázorněn na obr. 5.8. Ze stromu je navíc patrné, že dvojí aplikace pravidla *4c* lze provést souběžně, což ale běžné pojetí derivace nedovoluje, zatímco v případě multigramatiky je možno provádět aplikace pravidel paralelně v rámci jednoho derivačního kroku. Zároveň je ale vhodné vyjádřit povinný kontext, který nám povolí paralelně takový přepis provést. Zavedeme proto selektor tvaru $.^*\!<V>.^*\overline{<Det \asymp N>}P\overline{<Det \asymp N>}$. Výraz selektoru lze interpretovat následovně, proveď paralelní přepis užitím pravidla $NP \rightarrow Det\ N$ pokud se mezi pravými částmi pravidel vyskytuje prepozice (P) a zároveň jsou obě části předcházeny verbem (V) v levém pasivním kontextu. Formálně lze pak derivační krok povolený zavedeným selektorem vyjádřit jako:

$$NP\ V\ Det\ N\ P\ Det\ N \Rightarrow NP\ V\ NP\ P\ NP.$$

Podobně bychom mohli zavést i selektory pro úspěšný zbytek celé derivace. Jen podotkneme, že návrh každého selektoru není věcí nikterak náhodnou, nýbrž vychází z předpokladu znalosti správné syntaktické stavby věty. V situaci, kdy by nebyla splněna kontextová podmínka zavedená selektorem, derivace by byla předčasně zablokována - například pokud bychom z věty vypustili verbum, pak není možné v derivaci větné formy *NP_Det_N_P_Det_N* pokračovat a není tak proveden ani jeden derivační krok.



Obr. 5.8 – derivační strom nad testovanou větu

V porovnání s případem provedení derivace jen na základě vstupní bezkontextové gramatiky bychom tutéž chybu (absenci verba) detekovali až při 3. nebo 4. derivačním kroku. Zachycení pevných kontextových vazeb věty multigramatikou nám tedy poskytuje i prostředek pro včasné rozpoznání syntaktické chyby, a to i nad větnou částí, která ještě neprošla derivací (nebyla dosud přepisována).

III. Podpora pro deterministický parser

V podkapitole 5.1.3.3 jsme již zmínili důležitou úlohu každého parseru a tedy schopnost rozhodovat o výběru vhodného pravidla k simulaci provedení následného derivačního kroku. Implementace parseru musí z tohoto důvodu zahrnovat algoritmy, případně jinou vstupní znalost (podobně jako v případě tabulkou-řízeného parseru) ve snaze zaručit správnou progresi v rozboru.

Uvažme nyní úlohu postavení bezkontextové gramatiky v parseru. Bezkontextová gramatika, jakožto formální prostředek, neposkytuje samostatně mechanismus pro deterministické rozhodování o výběru „*vhodných*“ pravidel. Podobně bychom mohli uvažovat i o multigramatikách. Na straně druhé budeme ale jistě schopni pomocí multigramatik omezit velikost množiny potenciálně možných pravidel v určitém derivačním kroku při porovnání s prostředky bezkontextové gramatiky. Dále abychom u multigramatik vyjádřili jednoznačný vztah mezi pravidly a jim odpovídajícími aktivními částmi v selektorech, lze nadále uvažovat v aktivních částech nahrazení symbolů strany pravidel přímo jejich číselným značením (odkazem, referencí).

Uvažme příklad věty z části II. viz obr 5.8. Ke gramatice doplníme tzv. „*matoucí pravidlo*“:
9. $X \rightarrow Det\ N$ (toto pravidlo bude představovat nezbytnou součást gramatiky, je podstatné pro jiné derivace, avšak v derivaci tohoto příkladu je nežádoucí). Následně uvažme tuto (nejlevější) derivaci:

$$John\ saw\ a\ cat\ with\ my\ cookie - NP\ V\ Det\ N\ P\ Det\ N \Rightarrow NP\ V\ X\ P\ Det\ N \Rightarrow NP\ V\ X\ P\ X.$$

Z průběhu derivace vidíme, že užitím nesprávného (matoucího) pravidla namísto pravidla $NP \rightarrow Det\ N$ je derivace zablokována. Tomu lze ale v multigramatice předejít například zavedením selektoru $.*XX.*\overline{<9>}.*XX.*$, kde aktivní část výrazu $\overline{<9>}$ vyjadřuje možnost aplikaci 9. pravidla, ale jen při výskytu symbolicky zavedeného kontextu $.*XX.*$ před i za částí <*Det N*> ve větě. Protože se však kontext $.*XX.*$ ve větě nikde nenachází, je tím aplikace matoucího pravidla v této derivaci zcela vyloučena.

Další nežádoucí situace, které jsme schopni se při aplikaci multigramatiky vyvarovat, je případ, kdy nevhodným pořadím aplikace pravidel derivace předčasně končí, aniž bychom pokryly všechny fráze. Takové derivace je například:

$$John\ saw\ a\ cat\ with\ my\ cookie - NP\ V\ Det\ N\ P\ Det\ N \Rightarrow NP\ V\ NP\ P\ Det\ N \Rightarrow$$
$$NP\ VP\ P\ Det\ N \Rightarrow S\ P\ Det\ N \Rightarrow S\ P\ NP \Rightarrow S\ PP,$$

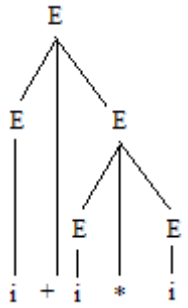
kde derivace je předčasně ukončena. Řešení této situace spočívá v zavedení selektoru ve tvaru $\overline{<1>}$, čímž vyjádříme požadavek na nepřítomnost jakýchkoli symbolů před i za částí <*NP VP*>, jen tehdy pokud bude tato podmínka splněna, budeme moci provést aplikaci pravidla číslo 1. Tedy až v okamžiku kdy bude věta zcela zredukována jen na fráze *NP VP*.

Poslední otázkou, kterou se zde na podporu deterministického parseru budeme zabývat, je schopnost s užitím multigramatik rozhodovat nejednoznačnost u gramatik bezkontextových. Uvažme následující příklad 5.4, známý zejména z prostředí výstavby překladačů, s pravidly nejednoznačné bezkontextové gramatiky.

Příklad 5.4: rozhodnutí nejednoznačnosti, mějme dána pravidla bezkontextové gramatiky:

1. $E \rightarrow E + E$
2. $E \rightarrow E * E$
3. $E \rightarrow (E)$
4. $E \rightarrow i$

Standardně budeme požadovat, aby během derivace nad větnou formou:"*i + i * i*" aplikace pravidla *2* (násobení) předcházela aplikaci pravidla *1* (sčítání), viz obr 5.9.



Obr. 5.9 – hledaný derivační strom nejednoznačné gramatiky

Zavedeme tři selektory: $.*\overline{<4>}.*$, $.*\overline{<2>}.*$ a $[\hat{}<"*">]\overline{<1>}[\hat{}<"*">]$, kde první nám umožňuje pomocí 4. pravidla neomezeně přepisovat *i* na *E*, druhý selektor vyjadřuje možnost násobit kdykoli pomocí 2. pravidla a třetí pak povoluje sčítání pomocí 1. pravidla jen pokud se před i za sčítáním již nevyskytuje žádná operace s násobením. Zavedením těchto selektorů je nad gramatikou pro daný vstup možné dosáhnout jen jednoho – požadovaného derivačního stromu a tedy rozhodnutí nejednoznačnosti gramatiky původní.

IV. Za hranicí rodiny bezkontextových jazyků

Pohledem napříč různorodostí a bohatstvím syntaxe, které nacházíme v rodině jazyků přirozených, si lze povšimnout rozdílů mezi nároky na sílu formálních prostředků pro popis určitého zvoleného jazyka. V abstraktní rovině pohledu (společné pro celou rodinu těchto jazyků) však zpravidla požadujeme takové formální prostředky, které by nám v obecné míře dokázali postihnout jakýkoliv jazyk z této rodiny. A právě snaha o nalezení přesného formálního prostředku s touto silou vedla k zavedení nové rodiny jazyků označovaných jako ***jazyky s jemným kontextem*** (*mildly context-sensitive languages* - *MCSL*). Požadavky na gramatiku pro MSCL jsou v porovnání s definicemi jiných gramatik rozdílné, obecně lze rodinu MSCL označit následnou definicí (podrobnosti viz [22]).

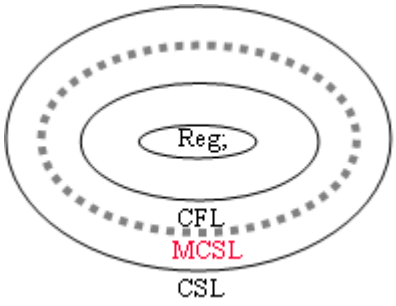
Definice 5.2:

MSCL jsou dány jako rodina jazyků *L*, která splňuje následující podmínky:

- každý jazyk v *L* je semilineární (viz [23])
- pro každý jazyk v *L* je problém členství řešitelný s časovou složitostí $m(n) = O(n^k)$.
- *L* zahrnuje následující ne-bezkontextové jazyky:

  - *vícenásobná shoda*: $L_1 = \{a^n b^n c^n \mid n \geq 0\}$
  - *křížená shoda*: $L_2 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$
  - *duplicita - opakování*: $L_3 = \{w\,w \mid w \in \{a, b\}^*\}$

Zařazení rodiny MCSL v porovnání s chomského klasifikací jazyků lze pak vidět na obr. 5.10.



Obr. 5.10 – zařazení MSCL v chomského hierarchii

Z obecného pohledu lze tedy prostředky bezkontextových gramatik (CFG) pro rodinu jazyků přirozených považovat za nepostačující. Pokud bychom uvážili nejbližší možný alternativní generativní systém z chomského hierarchie, dostáváme tak tzv. *kontextovou gramatiku* (CSG). Tato gramatika nám generuje tzv. *jazyky kontextové* (CSL), jazyky označované jako typu 1.

Definice 5.3:
Kontextová gramatika je čtveřice (N, T, P, S), kde N, T, S mají stejný význam jako u bezkontextové gramatiky G. P je konečná množina pravidel tvaru $x \rightarrow y$, kde $|x| \leq |y|$, $x, y \in \{T \cup N\}^*$.

Pozn.: jak můžeme z definice vidět, na rozdíl od CFG kde se na levých stranách pravidel mohou vyskytovat pouze osamocené neterminály, CSG obsahují pravidla, kde na jejich pravých i levých stranách mohou být celé řetězce. Jejich omezení však spočívá v délce řetězce levé strany. Zde je vyžadováno, aby délka řetězce na levé straně pravidla byla větší nebo rovna 1 a zároveň menší nebo rovna délce řetězce na straně pravé. Bez těchto omezení bychom dostali výpočetní sílu *Turingova stroje*, respektive pravidla neomezené (obecné) gramatiky (generující jazyky typu 0).

Uvažme nyní ukázku k ilustraci problému *vícenásobné shody* ve větě, viz příklad 5.5.

Příklad 5.5: generování vícenásobné shody, mějme dány následující pravidla kontextové gramatiky:

1. $S \rightarrow aSBC$
2. $S \rightarrow aBC$
3. $CB \rightarrow BC$
4. $aB \rightarrow ab$
5. $bB \rightarrow bb$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

Jak můžeme na příkladu níže vidět, pravidla této gramatiky nám generují jazyk všech neprázdných řetězců – posloupnosti symbolů ***a*** následovaných stejným počtem symbolů ***b*** a taktéž následovaných stejným počtem symbolů ***c***; formálně $L(G) = \{a^n b^n c^n \mid n \geq 1\}$. Příklad generovaného řetězce (v závorce uvedeno číslo užitého pravidla):

$S \Rightarrow aSBC$ [1] $\Rightarrow aaSBCBC$ [1] $\Rightarrow aaaBCBCBC$ [2] $\Rightarrow aaaBBCCBC$ [3] $\Rightarrow$ $aaaBBCBCC$ [3] $\Rightarrow aaaBBBCCC$ [3] $\Rightarrow aaabBBCCC$ [4] $\Rightarrow aaabbBCCC$ [5] $\Rightarrow aaabbbCCC$ [5] $\Rightarrow$ $aaabbbcCC$ [6] $\Rightarrow aaabbbccC$ [7] $\Rightarrow aaabbbccc$ [7]

Příklad 5.6: uvažujme dále pragmaticky tuto anglickou větu:

"*Peter, Paul and Mary could achieve marks ten, seven and eight in Mathematics, Linguistics and English, respectively*".

Kontextovou strukturu této věty lze vyjádřit jako: $a^3b^3c^3$ (vícenásobná shoda). Obecně můžeme tyto struktury vícenásobné shody vyjádřit jako jazyk L, kde $L = \{a^n b^n c^n \mid n \geq 2\} \in$ CSL ($L \notin$ CFL).

Podobně bychom mohli pomocí kontextové gramatiky demonstrovat i ostatní požadavky pro rodinu MSCL tedy generování jazyka s *kříženou shodou* nebo *duplicitou*. Zaměřme se však nyní na prostředky, které nám může poskytnout generativní systém multigramatik.

V kapitole 2. jsme zmínili generativní sílu multigramatik (*pro každý jazyk typu 0 přijímaný Turingovo strojem lze sestrojit multigramatika, která tento jazyk generuje*). Jedná se tedy nepochybně o prostředek s větší vyjadřovací silou než v případě zmíněných *kontextových gramatik*. Svou výpočetní silou nám tedy multigramatika zcela splňuje požadavky na generování zmiňovaných ne-bezkontextových jazyků – klíčových právě pro MSCL.

Ilustrujme si nyní generování jazyků s *vícenásobnou shodou*, *kříženou shodou* a *duplicitou* pomocí multigramatiky na několika následujících příkladech.

Příklad 5.7: $\boldsymbol{L_1}$ – generování vícenásobné shody multigramatikou, mějme dána tato pravidla:

1: $S \rightarrow abc$,
2: $S \rightarrow \varepsilon$,
3: $a \rightarrow aa$,
4: $b \rightarrow bb$,
5: $c \rightarrow cc$

a posloupnost dvou selektorů: ($\overline{S}$), $a^*(\overline{a})b^*(\overline{b})c^*(\overline{c})$, kde první selektor nám zaručí možnost zahájit generování z neterminálu S (pravidla 1 a 2), zatímco druhý selektor představuje nezbytnou podmínku generovat neprázdné řetězce jedině při současném užití pravidel 3, 4 a 5. Pro zajímavost si můžeme porovnat tato pravidla s pravidly uvedenými v příkladě CSG, jež generovala tentýž jazyk s výjimkou neprázdného řetězce. Příklad generování řetězce $a^3b^3c^3$:

$S \Rightarrow abc$ [1] $\Rightarrow aabbcc$ [3, 4, 5] $\Rightarrow aaabbbccc$ [3, 4, 5].

Příklad 5.8: $\boldsymbol{L_2}$ – generování křížené shody multigramatikou, mějme dána tato pravidla:

1: $S \rightarrow abcd$, – pro $n \geq 1, m \geq 1$
2: $S \rightarrow \varepsilon$, – pro $n = 0, m = 0$
3: $S \rightarrow ac$, – pro $n \geq 1, m = 0$
4: $S \rightarrow bd$, – pro $n = 0, m \geq 1$
5: $a \rightarrow aa$,
6: $b \rightarrow bb$,
7: $c \rightarrow cc$,
8: $d \rightarrow dd$

a posloupnost třech selektorů: ($\overline{S}$), $a^*(\overline{a})b^*c^*(\overline{c})d^*$, $a^*b^*(\overline{b})c^*d^*(\overline{d})$, kde podobně jako v případě jazyka L_1 nám první selektor umožní zahájit generování z počátečního neterminálu S (pravidla 1, 2, 3 a 4). Zatímco selektor druhý vyjadřuje nutnost současného užití pravidel 5 a 7 pro množení symbolů ***a*** a ***c***. A selektor třetí pak vyjadřuje nutnost současného užití pravidel 6 a 8 pro množení symbolů ***b*** a ***d*** (nezávisle na ***a*** a ***c*** a naopak). Příklad generování řetězce $a^2b^3c^2d^3$:

$S \Rightarrow abcd$ [1] $\Rightarrow aabccd$ [5, 7] $\Rightarrow aabbccdd$ [6, 8] $\Rightarrow aabbbccddd$ [6, 8].

Příklad 5.9: $\boldsymbol{L_3}$ – generování duplicity multigramatikou, mějme dána tato pravidla:

1: $S \rightarrow AA$,
2: $S \rightarrow BB$,
3: $A \rightarrow Aa$,
4: $A \rightarrow Ba$,
5: $B \rightarrow Ab$,
6: $B \rightarrow Bb$,
7: $A \rightarrow \varepsilon$,
8: $B \rightarrow \varepsilon$,

a posloupnost dvou selektorů: ($\overline{S}$), ($\overline{A}$)$(a+b)^*$($\overline{A}$)$(a+b)^*$, ($\overline{B}$)$(a+b)^*$($\overline{B}$)$(a+b)^*$. Příklad generování věty „*bbabba*“: $S \Rightarrow AA$ [1] $\Rightarrow BaBa$ [4] $\Rightarrow BbaBba$ [6] $\Rightarrow BbbaBbba$ [6] $\Rightarrow bbabba$ [8].

Demonstrovali jsme schopnosti multigramatik generovat jazyky z rodiny jazyků přirozených, které jsou však za hranicí jazyků bezkontextových (překračují prostředky bezkontextových gramatik).

5.3 Zhodnocení, porovnání s ostatními metodami

V kapitole 5.1 jsme diskutovali základní metody užité při současném zpracování přirozených jazyků na třech bázových rovinách (lexikální, morfologické a syntaktické).

V části následné 5.2 jsme navázali na poznatky předchozí a postupným výkladem a diskusí jsme rozebírali možné návrhy pro jednotlivé roviny zpracování. Od počátku návrhu je uvažován nově zavedený hypotetický jazyk APL, jehož hlavním úkolem je reprezentovat vlastnosti z rodiny přirozených jazyků na vyšším stupni abstrakce, tak abychom dosáhli obecnější platnosti poznatků zavedených během návrhu na jednotlivých rovinách zpracování. Zároveň je snahou vyzvednutí a začlenění do APL těch vlastností, které jsou vhodné pro počítačové (strojové) zpracování jazyků v porovnání s požadavky lingvistickými. Snahou je také vyvarovat se tzv. *eurocentrickému přístupu* a využití i vlastností jazyků z různých zeměpisných poloh.

K získání objektivního přehledu a zkušeností s jinými jazyky bylo také zejména využito dlouhodobých studií jazykovědce Richarda A. Morneau, vlastních zkušeností s rodným jazykem, jazykem mezinárodní komunikace – angličtinou, poznatků získaných během práce zabývající se zpracováním plánovaného jazyka Esperanto [13] a dále pak poznatků nabytých během intenzivního vystavení se poslechu přirozené komunikace a záznamu zpráv v několika mimoevropských jazycích.

Na rovině lexikální jsme se zaměřili na popis lexikálních jednotek (lexémů) pro APL a dále pak na filtrační a indexační přístupy, které nám pomáhají připravit textový vstup pro morfologickou rovinu zpracování. Princip je založen v nasazení preprocesoru jako prostředku pro přípravnou fázi analýzy a současný sběr lexikálních příznaků pro každý rozpoznaný lexém.

V rovině morfologické jsme nejprve definovali jednotlivé množiny hlásek, následně stanovili formální popis pro slabiky a posléze také pro celé morfémy. Dále pak jsme na stavbě slov demonstrovali možnost sebeseparátních morfémů. Důležitým prvkem pro morfologickou rovinu u APL bylo zavedení prostředků pro stanovení *gramatických charakteristik* (převážně jednoznačnou příslušnost ke slovním druhům) zejména na základě vhodně navržené slovní syntaxe namísto tradičního korpusového přístupu, který vyžaduje rozsáhlou počáteční vstupní znalost.

V okamžiku, kdy jsme schopni získat požadované gramatické charakteristiky v podobě detekovaných morfologických příznaků, můžeme se zabývat zhodnocením návrhu na rovině syntaktické. Zde jsme nejprve uvedli popis pro frázové a závislostní syntaktické struktury. Následně jsme provedli návrh formálního prostředku pro vyjádření syntaxe APL (viz APLF).

Z pohledu závislostní struktury jsme zvolili model VSO. Posléze byl demonstrován komplexní návrh syntaxe jazyka APL a zmíněna některá vhodná rozšíření.

V nadcházející části jsme se zabývali obecnou úlohou *aplikace multigramatik* v syntaktické rovině. Zmíněny byly podstatné úkoly pro návrh vhodného parseru. Dále jsme diskutovali postavení generativních gramatik v návrhu parseru při zaměření na multigramatiky. Aplikaci multigramatik jsme uvedli z několika rovin pohledu, demonstrovány byly některé nedostatky a omezení, které plynou z použití běžných bezkontextových gramatik a předvedena jejich řešení při zavedení prostředků multigramatik. Navržená řešení a příklady byly koncipovány zejména s ohledem na potřeby z oblasti zpracování přirozených jazyků a lingvistiky, ačkoli lze obecně některé poznatky využít i pro oblasti příbuzné, mezi které také bezpochyby patří i návrh překladačů.

6 Implementační aspekty pro aplikace multigramatik

V nadcházející kapitole popíšeme problematiku analýzy a návrhu softwarových produktů pro vybrané tematické oblasti k aplikacím multigramatik. Nejprve se zaměříme na společné otázky obou tematických oblastí (model multigramatik, životní cyklus, softwarová architektura, volba implementačního jazyka) a následně se budeme podrobněji zabývat specifickými otázkami každé tematické oblasti zvlášť (odlišnosti v návrhu, demonstrace činnosti, popis výstupů).

6.1 Volba implementačního jazyka

S ohledem na požadavky projektu a tendence v současném vývoji software byl od počátku zamýšlen objektově orientovaný jazyk, který by umožnil vytvoření vazby mezi abstraktním chápáním zpracovávané problematiky (zde v podobě objektů) a jeho realizací prostředky programovacího jazyka.
Dále s ohledem na požadavky platformové nezávislosti (přenositelnosti) a dostupnosti zdrojových kódů jazyka byl zvolen jazyk Java.

Jazyk Java mimo jiné usnadňuje vývoj podporovanou sadou knihoven, které jsou jeho implicitní součástí. Vývojář softwaru tak není při řešení zpravidla nucen, používat knihovny třetích stran (ne vždy dostupných jako freeware, případně může být omezení způsobeno restriktivnější licencí) nebo věnovat další čas jejich vlastní implementací. S uvolněním jazyka Java jako *opensource* není navíc tvořena žádná zábrana mezi znalostí kódu knihovny a jejím použitím (dává také možnost stávající kód jen upravit a implicitní část tak nechat překrýt vlastní modifikací). Ve výsledku je tak vývojář v jazyce Java postaven před volbu ze třech situací:

1. implementace zcela vlastní knihovny (časově náročné),
2. implementace knihovny na základě modifikace stávající (vhodnost dle rozsahu úprav),
3. využití stávajících knihoven bez nutnosti zásahu (časově úsporné, vyžaduje jen znalost a orientaci v dokumentaci „*JavaDoc*“, volitelně pak zhlédnutí zdrojových souborů knihovny).

Pokud je možné využití knihovny stávající (varianta 3), jedná se o časově úsporné řešení a navíc je možné věnovat možný čas na vývoj vlastní problematiky (řešíme efektivně skutečný problém) namísto vývoje řady problémů podružných, jejichž řešení je často známé.

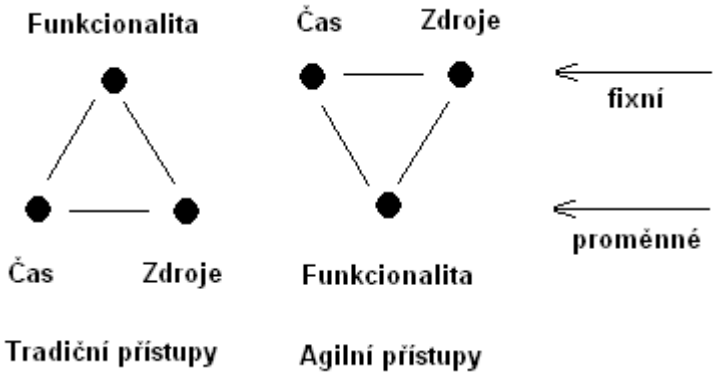
V případě řešení rozsáhlejších podproblémů pak přecházíme zpravidla do styku s prvky, jako jsou návrhové vzory („*design patterns*“) a pracovní šablony („*frameworks*“). Zatímco návrhový vzor lze chápat jako předpis pro řešení známé problematiky, framework pak představuje sadu knihoven či pomocných skriptů, která již danou problematiku přímo řeší. I zde nám může být jazyk Java nápomocen, neboť podporuje vývoj a distribuci těchto prostředků.

Pozn.: v souvislosti s pojmy (zmíněné zejména v předešlém odstavci), které jsou často převzaty z anglického jazyka, si dovolíme poznamenat následující. Tyto termíny (jako např. „*design pattern*“) jsou užívány jako zcela běžná součást IT terminologie. Je však s podivem, pokud se setkáme s mluvčím, který takovýchto termínů užívá, aniž je schopen je řádně vysvětlit. Podotkněme také, že většina těchto termínů již existovala mnohem dříve před vznikem a aplikací v softwarovém vývoji. Například návrhové vzory jsou zcela běžné v každodenním životě. K asi nejmarkantnějším a nejstarším příkladům patří architektura. Gotickou katedrálu je možné rozpoznat jen na základě návrhových vzorů, které jí charakterizují (např. rozeta, gotický opěrný systém, klenba, lomený oblouk atd.).

6.2 Životní cyklus software produktů

I v rámci vývoje software produktu pro aplikace multigramatik je nezbytné zastávat všechny pracovní pozice/funkce, které životní cyklus softwarového produktu vyžaduje. Jedná se zejména o kompletaci specifikace, tvorbu návrhu, implementaci, sestavení průběžné dokumentace, prezentaci produktu, ale také vnitřní organizaci jako je plánování projektu a jeho řízení včetně rovnoměrného rozvržení pracovní zátěže v časových intervalech.

S ohledem na tyto požadavky a samostatný vývoj, byl zvolen přístup tzv. **agilního vývoje**. Uveďme si nyní pro přehled základní charakteristiky tohoto vývoje. Agilní přístup považuje čas a zdroje za fixní (stanovené na začátku projektu) a mění se jen funkcionalita. Průběžně se tak přizpůsobuje a mění se ve své šíři i hloubce (viz obr. 6.1).



Obr. 6.1 – charakteristika agilního vývoje v porovnání s tradičními přístupy

Hlavní zásady agilního vývoje jsou shrnuty do manifestu, ten zavádí následujících pět priorit, které jsou i v rámci vývoje dodržovány:

1. Dobrá práce pochází jen z dobrého člověka.
2. Jednotlivci a interakce před procesy a nástroji.
3. Fungující software před úplnou dokumentací.
4. Spolupráce se zákazníkem před vyjednáváním kontraktu.
5. Reagování na změny před dodržováním plánu projektu.

Pro agilní vývoj jsou dále typické malé iterační cykly, které představují drobné životní cykly v porovnání například s často zmiňovaným vodopádovým modelem. Těchto cyklů bylo v našem případě využito zejména pro kompletaci dokumentace, stanovení dílčích milníků, případně prezentaci software v podobě aktuálního ho sestavení („*build*"). Následně zmíníme podrobnější techniky vývoje, které byly během implementace využity.

6.2.1 Agilní vývoj, programování řízené testy

Programování řízené testy („*test driven developement*" - TDD) představuje druh vývoje, kdy nejprve napíšeme testy předtím, než máme dostatek kódu, který by testy prošel (*test-first* přístup) a následně se dle výsledku snažíme pokrývat požadavky na výstupy programu, čímž řídíme vývoj celého software. Testy zde tedy nejsou jen nástrojem pro ověření funkčnosti, ale jsou součástí nástrojů pro návrh systému.

Pozn.: ačkoli se tato technika extrémního programování může zdát být poměrně novou a v poslední době často skloňovanou, její využití při vývoji software existovalo už v historii dříve. Prvky programování řízeného testy již byly použity NASA v projektu Mercury v 50. letech 20. století.

Vlastní vývoj lze pak shrnout do základních třech bodů (podrobněji viz [30]):

1. Tvorba testu. První spuštění, pravděpodobně bude obsahovat určité chyby.
2. Implementace testované logiky k úspěšnému dokončení testu.
3. Využití refaktorování (neměníme funkčnost, ale jen strukturu, organizaci kódu) a opakovaného spouštění testu k modifikaci kódu do přijatelné podoby.

Neformálně je tato trojice nazývána jako: „*červená, zelená, refaktorizace*". Tento výrok podtrhuje také využití těchto technik v jazyce Java tvorbou tzv. „*JUnit*" testů. V kombinaci s vhodným vývojovým prostředím (např. *NetBeans*) jsou tak poskytnuty tvůrci všechny základní prostředky pro naplnění základních myšlenek tohoto vývoje. Mimo jiné programování řízené testy zvyšuje důvěru ve fungující systém, který odpovídá požadavkům, čímž motivuje vývojáře k dalším cílům.

Kromě zde popsané techniky využité při implementaci produktů pro aplikace multigramatik, byla věnována pozornost také volbě a využití vhodné softwarové architektury. Té se budeme věnovat v nadcházející části.

6.2.2 Volba softwarové architektury

Před zahájením vlastního implementace byly vybrány a přijaty principy vhodné softwarové architektury. Na základě požadavků, které klade problematika obou tematických oblastí, byla zvolena softwarová architektura MVC („*model view controller*"). Uveďme si nyní pro přehled její základní charakteristiku.

MVC (lze přeložit jako *architektura řízená modelem*) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku („*business logic*“) do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní. Základním požadavkem je zde oddělení aplikační a prezentační logiky. Na architekturu MVC lze také nahlížet jako na tzv. návrhový vzor, tedy sadu doporučení, které je v případě odpovídající úlohy vhodné dodržovat.

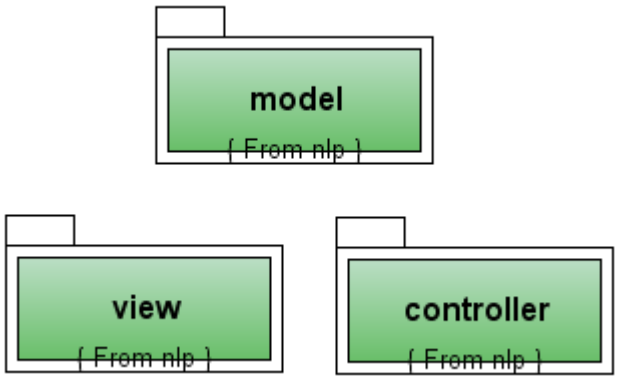
Obecně řečeno, vytváření aplikací s využitím architektury MVC vyžaduje rozdělení vývoje do tří základních logických komponent, mezi ty patří:

- **Model (model)**, což je doménově specifická reprezentace informací, s nimiž aplikace pracuje.
- **View (pohled)**, který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli.
- **Controller (řadič)**, který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu.

V kontextu s implementací tematických oblastí pro multigramatiky tak získáme koncept, který lze popsat následovně:

- **Model multigramatik** (*package model*), zahrnuje zejména model multigramatik, jeho pravidel a selektorů.
- **View** (*package view*), zajišťuje interakci s uživatelem, v případě tematické oblasti zpracování přirozených jazyků v podobě konzolové aplikace, v případě tematické oblasti návrhu selektivního L-systému pak ve formě GUI.
- **Kontrolér** (*package controller*), zprostředkovává komunikaci mezi modelem a uživatelským prostředím. V případě tematické oblasti zpracování přirozených jazyků zahrnuje lexikální analýzu a parser nad modelem multigramatik, v případě druhé tematické oblasti návrhu selektivního L-systému je pak navíc rozšířen o sémantické akce a jejich zpracování.

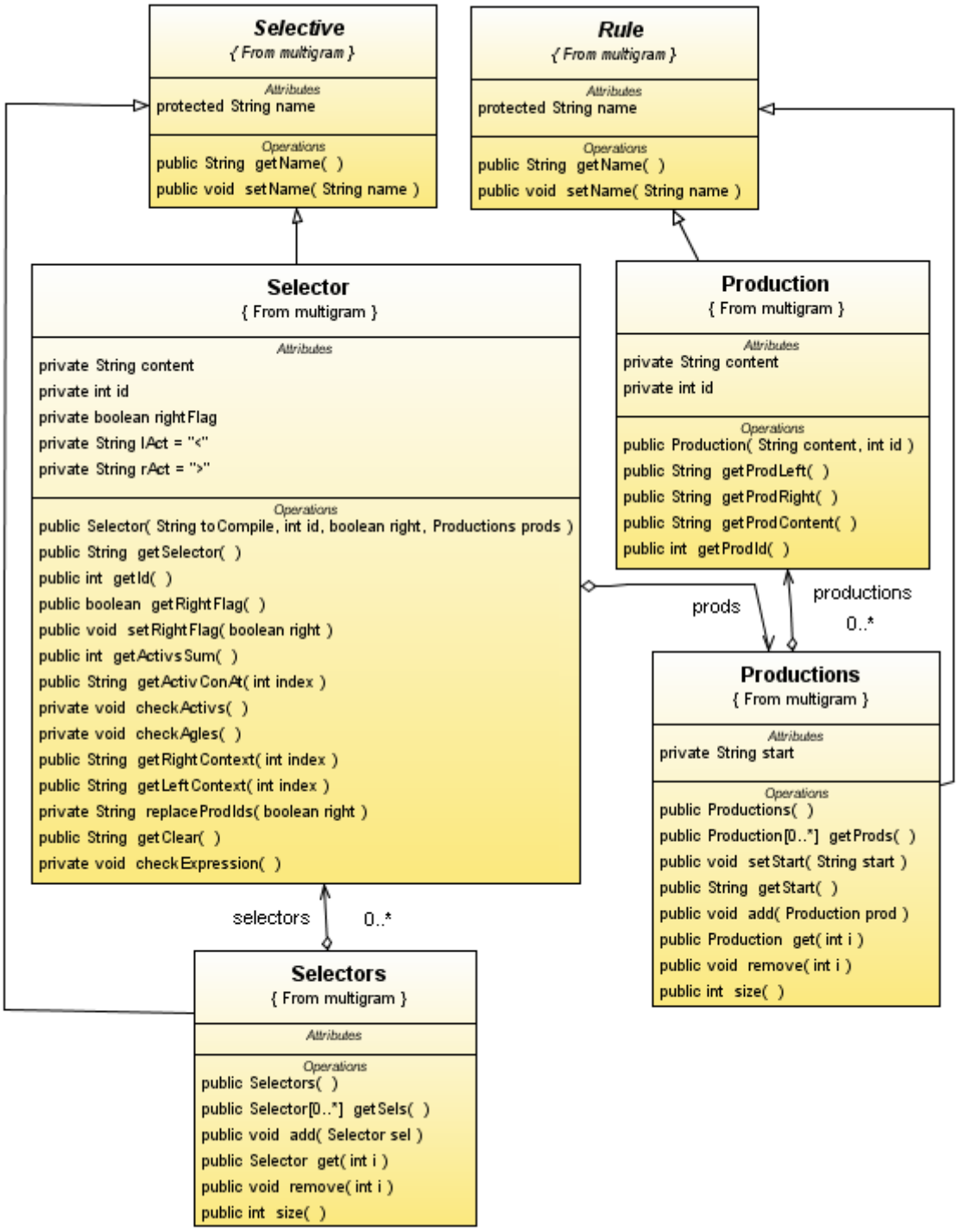
Například pokud uživatel provede v aktuálním grafickém zobrazení určitou akci, je v rámci GUI zachycena a předána ke zpracování v rámci kontroléru, což zahrnuje volání operací nad modelem, provedené změny jsou v modelu uloženy a zpětně skrze kontrolér předány k vykreslení v GUI. Diagram tříd pro vybranou architekturu MVC je zobrazen na obr 6.2.



Obr. 6.2 – struktura balíčkového systému při implementaci tematických oblastí, s ohledem na požadavky architektury MVC

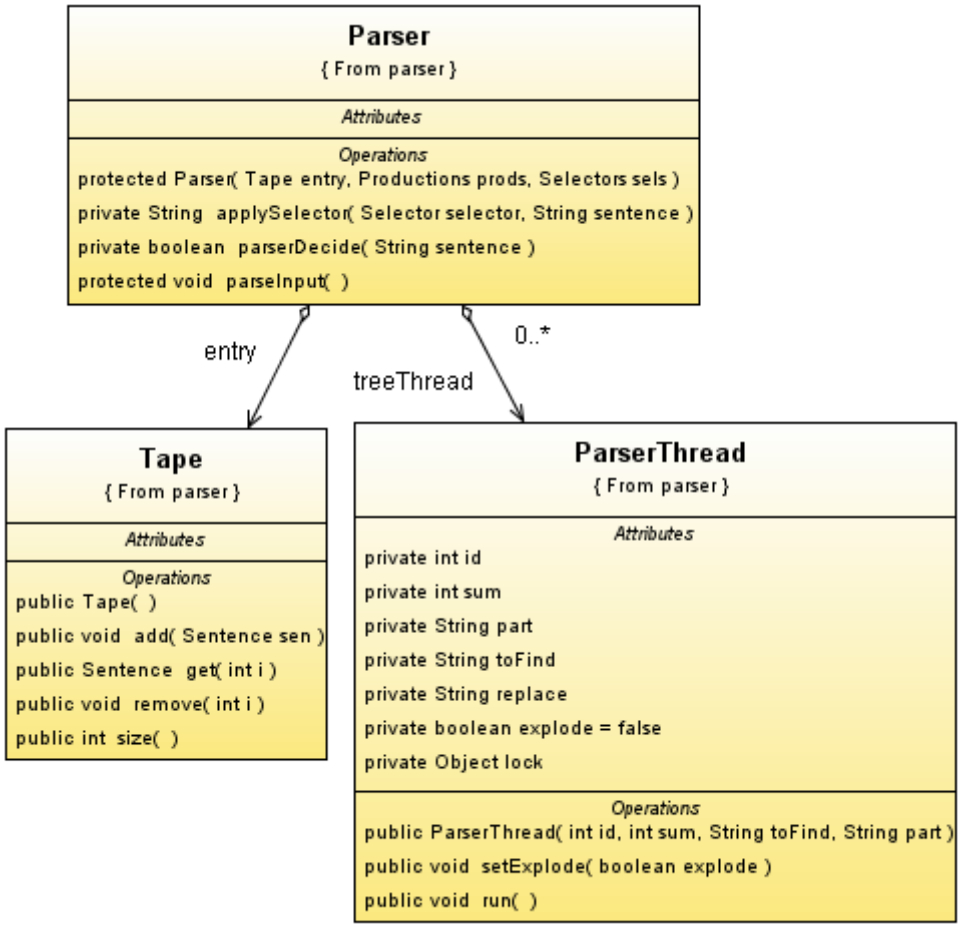
6.2.3 Struktura architektury MVC podrobněji

Kromě základní struktury, která vyplívá z požadavků architektury MVC (jak již bylo také naznačeno v předchozí části), budou nyní podrobněji popsány další třídy, jejich závislosti dle návrhu plynoucího ze specifikace vybraných tematických oblastí. Podotkněme jen, že tato část návrhu je společnou pro obě tematické oblasti. Popis bude proveden formou diagramů tříd dle notace UML. Struktura navrženého modelu multigramatik je znázorněna obr 6.3.



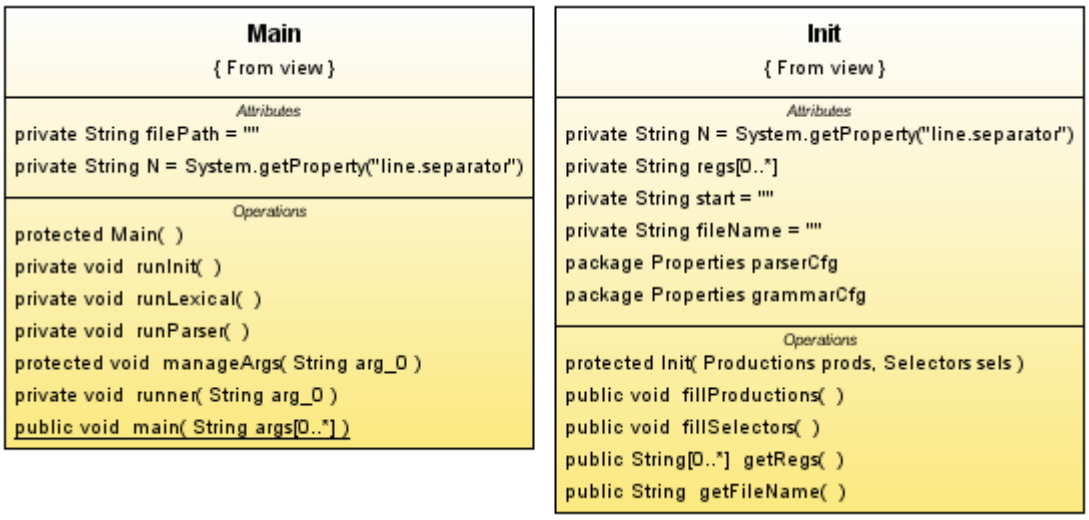
Obr. 6.3 – struktura balíčkového systému modelu multigramatik (*package model.muligram*)

Struktura kontroléru je v obou tematických oblastech zastoupena řídícím parserem nad modelem multigramatik, viz obr 6.4.



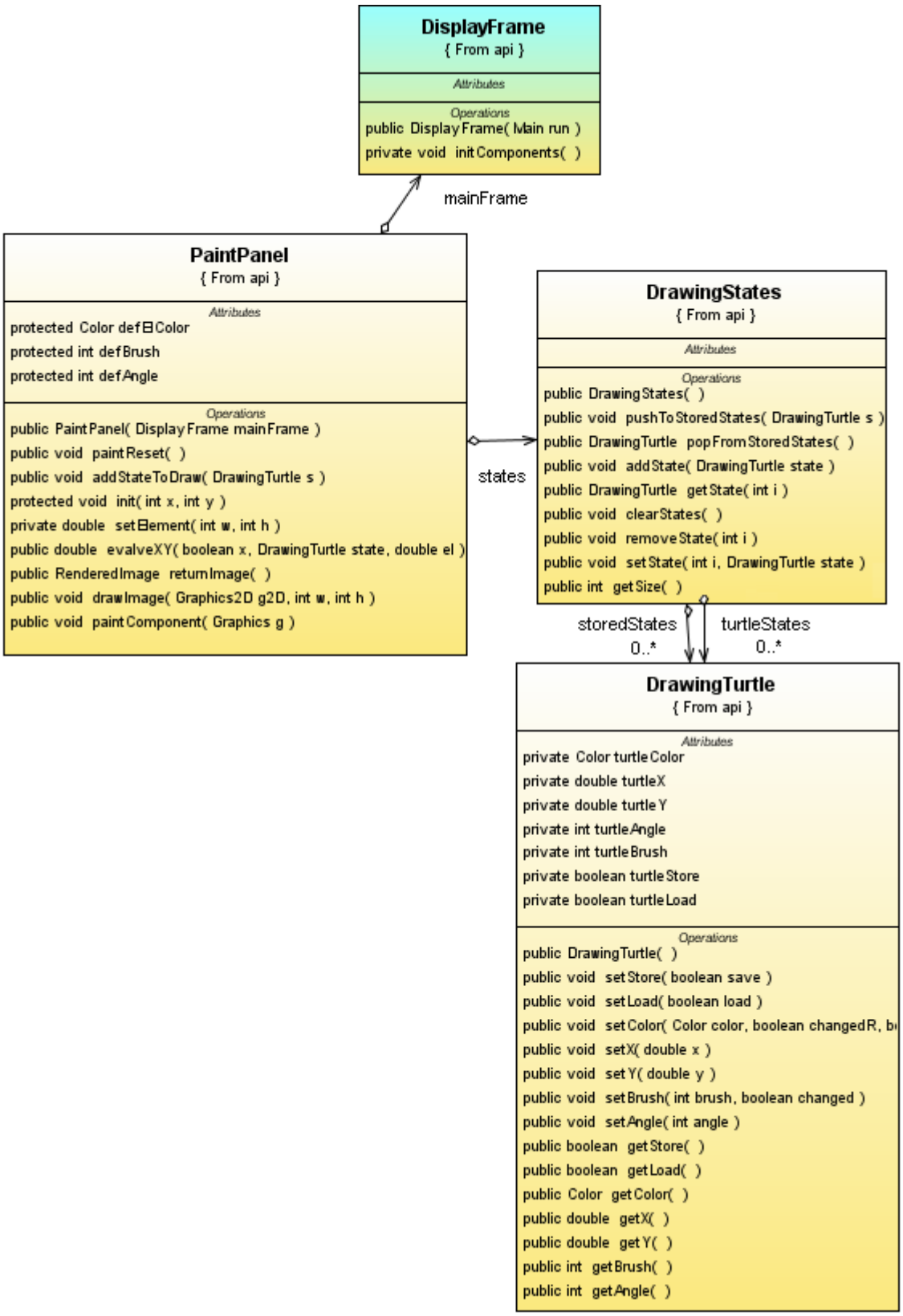
Obr. 6.4 – struktura balíčkového systému parseru (*package controller.parser*)

Základní struktura pro uživatelskou interakci formou konzolové aplikace je znázorněna na obr 6.5.



Obr. 6.5 – základní struktura balíčkového systému uživatelské interakce (*package view*)

Struktura rozšířeného uživatelského rozhraní (GUI) specifického pro tematickou oblast selektivního L-systému je na obr. 6.6 (pro zjednodušení nejsou zobrazeny všechny metody a atributy).



Obr. 6.6 – rozšířená struktura GUI specifická pro selektivní L-systém (*package view.api*)

6.3 Vlastnosti softwaru selektivního L-systému

V této kapitole se podrobněji zaměříme na specifické vlastnosti, výstupy implementovaného softwaru pro tematickou oblast navrženého selektivního L-systému.

6.3.1 Souhrnné informace

- Tematická oblast: implementace navrženého selektivního L-systému
- Řešená matematická úloha: Lindenmayerovy systémy s aplikací multigramatik
- Počet parametrů úlohy: 12
- Použité programové nástroje: Java, vývojové prostředí *NetBeans*
- Rozsah programového díla: 3800 řádků kódu
- Distribuce: *opensource* licence *Apache* 2.0
- Grafický formát výstupu: JPG, PNG nebo BMP
- Maximální rozměr výstupního obrazu: 1280x1024
- Počet ukázkových příkladů: 3 s rozměry 1280x1024, formát PNG (adresář demo)

6.3.2 Uživatelské rozhraní selektivního L-systému

Program byl navržen pro tvorbu a interpretaci selektivních L-systémů skrze snadno ovladatelné grafické prostředí. Nejprve se budeme zabývat základní funkcionalitou programu. Princip a tvorba vlastních vstupních systémů bude vysvětlen až dále. Základní menu zachycuje obr. 6.7.

Obr. 6.7 – základní ovládací menu, práce se souborem, editace plochy, about

6.3.2.1 File menu

Program je distribuován již se skupinou konfiguračních souborů, které představují různé příklady vytvořených selektivních L-systémů. Stručně nyní popíšeme způsob načtení a uložení těchto souborů.

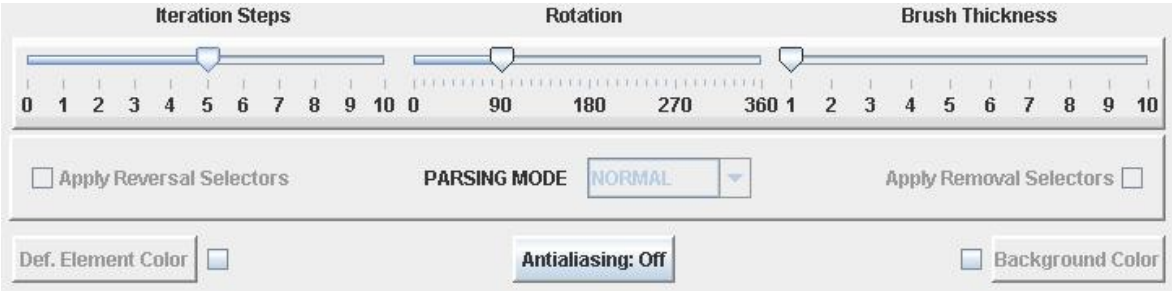
- **OPEN**: otevře vstupní textový soubor s obsahem definice pro selektivním L-systém a provede jeho interpretaci.
- **OPEN & EDIT**: totéž jako OPEN. Navíc otevře samostatné editační okno, kde je možno interaktivně provádět změny v definičním souboru a ihned je pak aplikovat.
- **SAVE AS**: provede uložení grafiky z kreslící oblasti do souboru. Na výběr jsou formáty JPG, PNG nebo BMP.
- **CLOSE**: ukončí práci s načteným systémem a překreslí plátno.
- **EXIT**: provede ukončení práce s celou aplikací.

6.3.2.2 Set, about menu

- **SCALE**: provede změnu měřítka na požadovaný násobek/zlomek, nejmenší měřítko je velikost počáteční.
- **HELP**: provede otevření tohoto manuálu přímo v prostředí programu. Možno také otevřít tento soubor samostatně v libovolném prohlížeči.
- **ABOUT**: informace o tomto programu.

6.3.2.3 Ovládací panel

Ovládací panel slouží k řízení dynamických změn parametrů selektivního L-systému. Jedná se zejména o parametr počtu iteračních kroků, parametr úhlu natočení, tloušťku štětce, aplikaci speciálních reverzních či vyjimacích selektorů a volbu způsobu interpretace (L-systém versus selektivní L-systém), viz obr 6.8.



Obr. 6.8 – ovládací panel pro řízení dynamických změn parametrů systému

- **Iteration Steps**: tento posuvník nastaví potřebný počet iterací (počet paralelních aplikací možného pravidla nad větnou formou).
- **Rotation**: změna hodnoty pro interpretační symboly L-systému "*p*" a "*m*".
- **Brush Thnickness**: nastaví tloušťku štětce pro všechny vykreslované elementy, které nemají definovanou jinou hodnotu tloušťky přímo ve větné formě nebo pravidle.

- **Parsing Mode**: provede přepnutí mezi parserem pro běžný L-systém a selektivním parserem (aplikace multigramatik). Podrobnosti viz níže "*Pokročilé ovládání programu*".
- **Apply Reversal Selectors**: provede aktivaci/deaktivaci reversních selektorů nad větou, která již prošla všemi iteračními kroky. Podrobnosti viz níže "*Pokročilé ovládání programu*".
- **Apply Removal Selectors**: provede aktivaci/deaktivaci vyjímacích selektorů nad větou, která již prošla všemi iteračními kroky. Podrobnosti viz níže "*Pokročilé ovládání programu*".
- **Define Element Color**: každému vykreslovanému elementu přiřadí tuto barvu, pokud nemá daný element definovanou jinou hodnotu barvy přímo ve větné formě nebo pravidle.
- **Background Color**: změna barvy pozadí na zvolenou hodnotu.
- **Antialiasing**: provede aktivaci/deaktivaci antialiasingu nad kreslícím plátnem.

6.3.3 Pokročilé vlastnosti selektivního L-systému

6.3.3.1 Terminologie selektivního L-systému

Následný text představuje hrubé seznámení s nezbytnou terminologií pro tvorbu nebo úpravu vstupních systémů v tomto programu. Kromě terminologie zmíněné níže se také předpokládá znalost principu multigramatik zmíněná např. v kapitole 2.

Jak již bylo nastíněno také výše, zde navržený program pro tematickou oblast selektivních L-systémů je také schopen vytvářet, interpretovat, editovat tzv. běžný L-systém. Dále se však zaměříme zejména na specifika selektivního L-systému (SELEKTIVNÍ MOD).

Iterační krok představuje přepisy ve větě provedené jednou aplikací každého z možných pravidel gramatiky. V situaci, kdy gramatika L-systému disponuje jedním pravidlem, pak počet iteračních kroků odpovídá počtu paralelních přepisů nad měnící se větnou formou. Získaná výsledná věta je na základě zavedené sémantiky jednotlivých symbolů graficky interpretována.

V tomto programu budeme pracovat celkem se třemi typy selektorů. Všechny selektory jsou vyjádřeny dle notace zápisu regulárních výrazů v programovacím jazyce Java. Prvním ze selektorů je tzv. **obecný selektor**, který je použit při aktivaci SELEKTIVNÍHO MODU. Tento selektor a akce s ním spojené přesně odpovídají chování selektoru dle definice multigramatik (viz kapitola 2). V selektivním modu nebude tedy možné provést přepis, pokud není taková akce explicitně povolena některým z těchto obecných selektorů.

Druhým typem selektoru je tzv. **vyjimací selektor** (*removal selector*). Jeho notace je pro svůj předem stanovený účel zjednodušena. Je také tvořen regulárním výrazem, který ale představuje jen jednu aktivní část. Jako levá část a pravá část kolem tohoto aktivního prvku jsou automaticky doplněny libovolné symboly, dle podoby dané větné formy. Zároveň zde neuvádíme speciální symboly pro oddělení aktivní části (díky zjednodušené formě, již není třeba oddělovat). Zatímco bychom museli

selektor běžně popsat např. takto: ".*<*a*+>.*", zde bude popis při zachování předchozího významu následující: "*a*+". Dále je zde předem nastavena pevná akce, kterou již napovídá samotné označení selektoru. Každá nalezená shoda bude znamenat vyjmutí z větné formy podřetězce v rozsahu této shody. Na rozdíl od obecných selektorů jsou, které jsou postupně aplikovány během parsování, dochází k aplikaci vyjímacích selektorů až na konci činnosti parseru, kdy je známa výsledná věta. Po aplikaci vyjimacích selektorů (*Apply Removal Selectors*) budou nad dosud získanou větnou formou aplikovány všechny vyjimací selektory nalezené v otevřeném vstupním souboru. Dochází tak k vyjmutí některých vhodně zvolených symbolů zapsaných těmito selektory z větné formy. Tím je ve výsledku grafický řetězec - věta patřičně redukována a tedy i ovlivněn celkový grafický výstup. Příklad takového výstupu bude prezentován níže společně s popisem tvorby selektivního L-systému.

Třetím typem selektoru je tzv. **reverzní selektor** (*reversal selector*). Pro jeho zápis a aplikaci platí totéž, co bylo zmíněno u vyjimacího selektoru. Avšak přednastavenou akcí zde není vyjmutí podřetězců věty v rozsahu shod, ale jak opět název napovídá, jedná se o operaci reverze nad těmito podřetězci. Pokud by nalezená shoda začínala některým z vyhrazených sémantických symbolů (viz níže) a chtěli bychom tento symbol z reverze vynechat, je toho možné dosáhnout pomocí speciálního symbolu "~" před definicí takového selektoru. Po aplikaci reversních selektorů (*Apply Reversal Selectors*) budou nad dosud získanou větnou formou aplikovány všechny reversní selektory nalezené v otevřeném vstupním souboru. Dochází tak k převrácení některých vhodně zvolených částí popsaných těmito selektory z větné formy. Tím je ve výsledku grafický řetězec - věta patřičně změněn a tedy i ovlivněn celkový grafický výstup. Příklad takového výstupu bude také prezentován níže společně s popisem tvorby selektivního L-systému.

6.3.3.2 Popis tvorby selektivního L-systému

Před uvedením syntaxe pro vytvoření selektivního systému v souboru zmíníme nejprve význam zavedených sémantických symbolů, které programem očekávány.

Jako neterminály gramatiky a větných forem lze použít libovolná velká písmena anglické abecedy. Základním symbolem pro vykreslení je symbol "***F***". Ostatní symboly (různé od "***F***"), které se před ním vyskytují, představují parametry k vykreslení tohoto elementu. Následující seznam popisuje jednotlivé vyhrazené sémantické symboly a jejich význam.

- "***F***" : základní element k vykreslení - úsečka, její délka je počítána dle aktuální velikosti okna.
- "***p***" : natoč želvu ve směru hodinových ručiček o úhel "*angle*".
- "***m***" : natoč želvu proti směru hodinových ručiček o úhel "*angle*".
- "***s***" : ulož na zásobník celkový úhel a polohu počátečního bodu tohoto elementu
- "***l***" : nastav celkový úhel a polohu v počátečním bodě tohoto elementu dle hodnot na vrcholu zásobníku.

- "***c***" : natoč želvu ve směru hodinových ručiček o úhel zadaný bezprostředně za tímto symbolem (velikost úhlu není omezena).
- "***t***" : nastav tloušťku štětce pro tento element na hodnotu zadanou bezprostředně za tímto symbolem (rozsah 1 - 10).
- "***r***" : nastav R (*red*) složku barvy elementu na hodnotu zadanou bezprostředně za tímto symbolem (rozsah 0 - 255).
- "***g***" : nastav G (*green*) složku barvy elementu na hodnotu zadanou bezprostředně za tímto symbolem (rozsah 0 - 255).
- "***b***" : nastav B (*blue*) složku barvy elementu na hodnotu zadanou bezprostředně za tímto symbolem (rozsah 0 - 255).
- "***h***" : pomocný symbol použitý k vyznačení shod u vybraných částí mezi selektorem a větnou formou.

Pokud chceme sestavit vlastní vstupní soubor s definicí selektivního L-systému, je třeba nejprve vytvořit nový textový soubor (přípona **.cfg*) a v něm provést popis jeho vlastností dle zavedené syntaxe. Jako šablonu lze použít některý ze stávajících vytvořených souborů, které jsou distribuovány s programem (adresář *cfg*). Tyto soubory jsou také doprovázeny komentáři a to u každé definované vlastnosti. Abychom si ale udělali snazší představu o vytvoření takového selektivního L-systému, budeme si nejprve demonstrovat příklad.

Příklad 6.1 - Uvažme vytvoření selektivního L-systému s následujícími vlastnostmi.

- Počáteční vstupní větná forma: např. "*input = FsmhFhFlphFhF*"
- Pravidla gramatiky: např.
  "*Grammar.prod.0 = F > t2Fsmr255g250b19FhFlpFr255g250b19hFt2*"
- Obecné selektory (pokud chceme použít selektivní mód): např. "*Grammar.sel.0 = .*h<0>.**"
- Vyjimací selektory (pokud chceme použít): např. "*Grammar.sel.rem.0 = t\\d+*"
- Reversní selektory (pokud chceme použít): např. "*Grammar.sel.rev.0 = ~g\\d+*"
- Počáteční počet iteračních kroků: např. "*steps = 4*"
- Definice počátečních složek barvy elementu: např. pro R "*color.r = 128*", G a B podobně
- Definice hodnoty pro symboly '*p*' a '*m*': např. "*angle = 350*"
- Počáteční hodnota tloušťky štětce: např. "*brush = 1*"

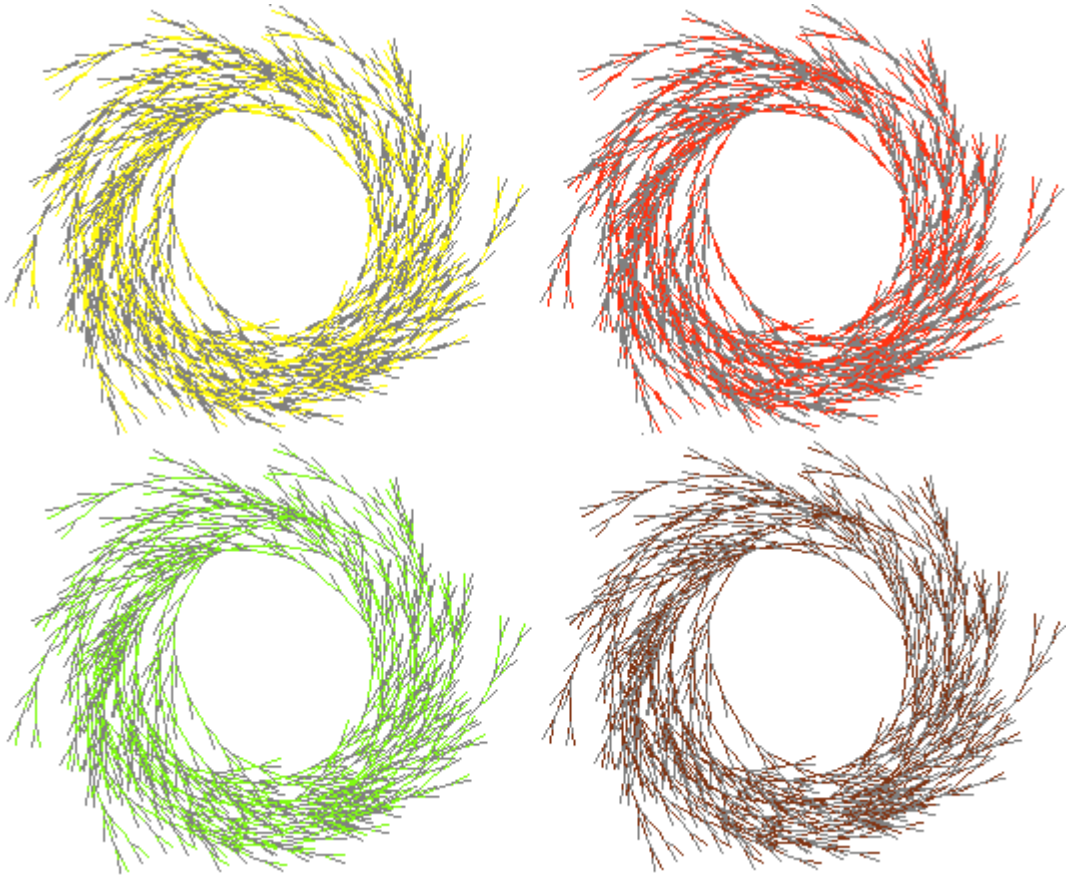
Jak můžeme v seznamu vidět, každé vlastnosti (daného pojmenování) je přiřazena požadovaná hodnota, přičemž pořadí vlastností není nikterak omezeno. Je třeba dodržet jen pojmenování jednotlivých vlastností a formát pro zápis jejich hodnoty.

Úplný zápis příkladu 6.1 selektivního L-systému do souboru je znázorněn na obr. 6.9 (včetně textů s komentáři).

```
# SPRING RIM

# set grammar productions
Grammar.prod.0 = F > t2F smr255g250b19F hF lpF r255g250b19hF t2

# set list of general selectors
Grammar.sel.0 = .*h<0>.*

# set list of removals selectors
Grammar.sel.rem.0 = t\\d+
Grammar.sel.rem.1 = r\\d+

# set list of reversal selectors
Grammar.sel.rev.0 = ~g\\d+

# set input sentence (initial sential form)
input = F smhF hF lphF hF

# set initial derivation steps (number of iterations)
steps = 4

# set default color as RGB model
color.r = 128
color.g = 128
color.b = 128

# set default angle (degrees, with right-rotation notation)
angle = 350

# set default brush thickeness
brush = 1
```

Obr. 6.9 – příklad definice selektivního L-systému v textovém souboru (viz příklad 6.1)

Pokud provedeme programem interpretaci (*file-open*) takto definovaného selektivního L-systému, obdržíme výsledek na obr. 6.10 (vlevo nahoře). Aplikací (*Apply reversal selectors*) reverzních selektorů získáme výstup na obr. 6.10 (vpravo nahoře). Pokud doplníme také aplikaci vyjimacích selektorů (*Apply removal selectors*, aplikace na sebe navazují) dostaneme výstup jako je na obr. 6.10 (vpravo dole). Ponecháním jen aplikace vyjimacích selektorů získáme výstup na obr. 6.10 (vlevo dole). Úpravou dalších parametrů skrze ovládací panel (počet kroků, nastavení úhlu, tloušťky štětce) lze v reálném čase dynamicky sledovat další změny v grafickém výstupu.



Obr. 6.10 – grafická interpretace příkladu 6.1 s využitím reverzních a vyjimacích selektorů

Další ukázky výstupů z vytvořeného programu navrženého selektivního L-systému jsou součástí přílohy 1. této práce.

6.4 Vlastnosti softwaru analyzátoru jazyka

V této kapitole se podrobněji zaměříme na specifické vlastnosti, výstupy implementovaného softwaru pro tematickou oblast zpracování přirozeného jazyka.

6.4.1 Souhrnné informace

- Tematická oblast: aplikace multigramatik pro analyzátor přirozeného jazyka
- Použité programové nástroje: Java, vývojové prostředí *NetBeans*
- Rozsah programového díla: 1300 řádků kódu
- Distribuce: *opensource* licence *Apache* 2.0
- Textový formát výstupu: konzolová aplikace, dle zavedené notace

6.4.2 Pokročilé vlastnosti analyzátoru

Cílem tohoto analyzátoru je zachycení kontextových vazeb, tak jak se mohou v běžných větách jazyka vyskytovat. Vstup je reprezentován posloupností lexikálních jednotek, jejichž vazby chceme ověřit. Zpravidla se jedná o vazby mezi slovními druhy, případně větnými členy. Analyzátor je navržen na abstraktní úrovni, není tedy svázán jen s prostředky jednoho jazyka. Tento přístup tedy předpokládá znalost analyzovaného jazyka a schopnost sestavit pravidla, která popisují kontextové vazby mezi zkoumanými lexikálními jednotkami.

Podotkněme jen, že tyto vazby jsou za hranicí rodiny jazyků bezkontextových, využíváme zde tedy výpočetní síly modelu multigramatik. Na rozdíl od modelu kontextových gramatik podporuje tento model paralelní zpracování a tvorba parseru navíc umožňuje využití již známých přístupů z prostředí současných překladačů (založených na modelu bezkontextových gramatik), neboť tvar pravidel multigramatik je založen právě na jednoduchosti bezkontextových pravidel.

Dále v porovnání s „*čistě*“ paralelními gramatikami, které přepisují všechny části větné formy dle použitého pravidla, v případě částečně paralelních gramatik (multigramatik) jsou přepsány jen některé z nich s ohledem na použitý selektor. V syntaktické analýze pak zpravidla požadujeme přepis jen některých sekvencí, zatímco jiné zůstávají beze změny. Z tohoto pohledu je vhodnější využít modelu částečně paralelních gramatik (bližší paralelismu reálného chování) – multigramatik.

6.4.2.1 Tvorba a příprava vstupu pro analyzátor

Abychom mohli připravit vstup analyzátor, je třeba nejprve připravit vstupní věty a definovat mutligramatiku k zachycení jejich kontextových vazeb. Analyzátor pak rozhoduje syntaktickou korektnost vět (s ohledem na definované vazby), případně detekuje výskyt chyby. Vstupní věty jsou načítány ze souboru „*input[utf8].txt*“ (adresář *cfg*, přepokládané kódováni je UTF-8). Definiční multigramatika je načítána ze souboru „*Grammar.cfg*“ (adresář *cfg*). Demonstrujme přípravu vstupu na následném příkladu.

Příklad 6.2 – příprava vstupu pro analyzátor, nechť je dán následující vstup:

```
(příklad dvou vět)
abcccbbab. abccbbabrt.
```

Komentář je uveden v závorce. Každá věta představuje posloupnost symbolů, kde každý symbol reprezentuje určitý druh lexému (např. slovní druh) jehož vazby s jinými lexémy chceme pokrýt.
Tento obsah uložíme do vstupního souboru vstupu („*input[utf8].txt*“ v adresáři *cfg*). Pro definici vazeb volme například multigramatiku dle obr. 6.11.

```
# Sentece context definition

# set the sign of starting symbol

Grammar.start= P

# set list of CFG like productions used by multigrammar

Grammar.prod.0= P > SCBD
Grammar.prod.1= S > abc
Grammar.prod.2= D > ab
Grammar.prod.3= B > bb
Grammar.prod.4= C > cc

# set list of selectors mapped to the list of productions

Grammar.sel.0= .*a*<1>c*<3>.*
Grammar.sel.1= <0>
Grammar.sel.2= .*S*<4>B*.*
Grammar.sel.3= ggg<3>hhh
Grammar.sel.4= SCB<2>
```

Obr. 6.11 – definice multigramatiky pro zachycení kontextových vazeb vět

Notace zápisu multigramatiky odpovídá její definici, jak bylo uvedeno například v kapitole 2. Výjimkou jen zápis selektorů, kde aktivní části jsou odděleny pomocí symbolů “<” a “>”. Obsahem aktivní části je odkaz přímo na číslo pravidla, s nímž selektor vytváří vazbu a pasivní části jsou zapsány přímo prostřednictvím regulárních výrazů dle notace užité v programovacím jazyce Java (podobně jako v případě notace použité při implementaci selektivního L-systému). Obsah definice z obr. 6.11 uložme do textového souboru pro vstupní multigramatiku („*Grammar.cfg*“ v adresáři *cfg*). Spuštěním programu jako konzolové aplikace pak obdržíme následující textový výstup.

```
- INIT ENTRY -
Start symbol is: P
<List of productions>:
      P>SCBD
      S>abc
      D>ab
      B>bb
      C>cc
<List of selectors>:
      .*a*<1>c*<3>.* | 2
      <0> | 1
      .*S*<4>B*.* | 1
      SCB<2> | 1
```

Jak můžeme ve výpisu vidět, v rámci inicializace je proveden výpis načtené definice multigramatiky. U výpisu selektorů je navíc za symbolem "|" vyjádřen počet jeho aktivních částí. Po inicializaci následuje proces lexikální analýzy. Zde jsou vypuštěny komentáře a rozpoznány lexikální jednotky.

```
- LEXICAL ENTRY -
<PREPROCES OUTPUT>:
      abcccbbab.
      abccbbabrt.
<ANALYSE>:
      Recognised senteces: 2
```

Navazuje proces syntaktické analýzy (parseru), v našem případě jsou kontextové vazby první věty v pořádku (derivační proces končí úspěchem), zatímco u věty druhé je derivace zablokována (není možná žádná další aplikace pravidla), věta je tedy (s ohledem na definici kontextu lexikálních jednotek v multigramatice) syntakticky chybná.

```
- PARSER ENTRY -
1st sentence:
      <Selector application number>: 1
            application at #: #ccbbab
            Selector of replacement: abc
            application at #: abccc#ab
            Selector of replacement: bb
            Derivation step => SccBab
      <Selector application number>: 3
            application at #: S#Bab
            Selector of replacement: cc
            Derivation step => SCBab
      <Selector application number>: 5
            application at #: SCB#
            Selector of replacement: ab
            Derivation step => SCBD
      <Selector application number>: 2
            application at #: #
            Selector of replacement: SCBD
            Derivation step => P
_____________________________________
Result: sentence context is correct.
```

```
2nd sentence:
      <Selector application number>: 1
            application at #: #cbbabrt
            Selector of replacement: abc
            application at #: abcc#abrt
            Selector of replacement: bb
            Derivation step => ScBabrt

_____________________________________
Result: sentence context is incorrect.
```

Demonstrovali jsme příklad vstupních vět a definice jejich kontextových vazeb formou multigramatik. Postavení multigramatik tak zde mimo úlohu výpočetního modelu pro syntaktický analyzátor plní také úlohu specifikace (definici kontextu vět analyzovaného jazyka).

Podobně můžeme vytvářet komplexnější definice, tak abychom pokryli všechny případy kontextových vazeb cílového jazyka a mohli tak v důsledku rozhodnout o správnosti zadaného vstupu.

Závěr

Stěžejní zhodnocení bylo uvedeno v rámci každé zkoumané tematické oblasti, viz kapitoly 4.3 a 5.3. Zaměříme se zde nyní proto jen na přehled dosažených výsledků.

Přínos práce spočívá ve studii teorie multigramatik a jejich následné aplikaci ve vhodných tematických oblastech. V oblasti L-systémů navazuje zavedením selektivních L-systémů (také s rozšířením typů selektorů) a jejich využitím zejména pro potřeby biomatematiky a výtvarné informatiky.

V oblasti zpracování přirozených jazyků pak pokračuje zavedením abstraktního jazyka (se schopností získávání gramatických charakteristik zejména na základě vhodně navržené slovní syntaxe namísto korpusového přístupu), posílením deterministického návrhu pro syntaktický nízkoúrovňový parser a postihnutím kontextových vazeb u rodiny přirozených jazyků za hranicí bezkontextové třídy.

Práce je dále rozšířena o analýzu a návrh softwarového produktu pro každou tematickou oblast (zahrnuje návrh vhodných paralelních algoritmů) za účelem demonstrace dosažených výsledků. Navržená řešení a příklady byly koncipovány zejména s ohledem na potřeby zvolených tematických oblastí, ačkoli lze obecně některé poznatky využít i pro příbuzné oblasti, mezi které také bezpochyby patří výstavba překladačů.

Tvé slzy se třpytí, oživené bolestí.
Srp měsíce visí v minulosti bledý v nemoci.
Chladná noc příliš dlouhá mění se v mráz.
Kdo je ve věži zmrazený v zoufalství?
Déšť lehce bubnuje na karmínová okna.
Můj osud je napsán na papíře třepotajícím se ve větru.
Vzdálené sny stoupají jak kadidlo.
Rozpouštějí se v noci jako tvůj obraz.
Chryzantémy padají plačící na zem.
Obrys tvého úsměvu na žlutém svitku, tvé srdce je teď prázdné.
Mé myšlenky tiše odpočívají, severní vítr fouká.
Ještě nesvítá, tvůj stín je tak čistý a blízký.
Jediný společník mé duše, na jezeře se zrcadlí pár.

<< 菊花台>>

Literatura

[1] Rozenberg, G., Klein, H. C. M.: Multigrammars,
International Journal of Computer Mathematics, s 177-201, 1983.

[2] Meduna, A.: Six-Nonterminal Multi-Sequential Grammars Characterize the Family of Recursively Enumerable Languages,
International Journal of Computer Mathematics, s. 179-189, 1997.

[3] Meduna, A.: Descriptional Complexity of Multi-Continues Grammars,
Acta Cybernetica, s. 375-384, 1998.

[4] Meduna, A., Vurm P.: Multisequential Grammars with Homogeneous Selectors,
International Journal of Computer Mathematics, s. 6, 2001.

[5] Meduna, A., Lukáš R., Krajíček J.: Homogeneous Multicontinuous Grammars and Their Reduction, zasláno do *Fundamenta Iformaticae*, 2007.

[6] Dassow, J., Paun, G.: *Regulated Rewriting in Formal Language Theory*. Springer,
New York, 1989, ISBN 0-38751-414-7.

[7] Meduna A., Švec M.: *Grammars with Context Conditions and Their Applications*,
Wiley, New York, US, WILEY, 2005, ISBN 0-471-71831-9.

[8] Mandelbrot, B. B.: *The Fractal Geometry of Nature*, San Francisco: Freeman, 1982

[9] Thompson, A.: *On Growth and Form: The Complete Revised Edition*,
Dover Publications, 1992, ISBN-10: 0486671356.

[10] Abelson, H., diSessa, A.: *Turtle Geometry*, MIT Press, Cambridge,
Massachusetts (USA), 1992, ISBN 0-262-51037-5.

[11] Sun Microsystems, *Java Documentation: Regex Pattern*, dokument dostupný na:
http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html (2006).

[12] Čermák, F.: *Český národní korpus* [akademický projekt], Praha, Filozofická fakulta Univerzity Karlovy (ÚCNK) 2002. Dokument dostupný na: http://ucnk.ff.cuni.cz/.

[13] Krajíček. J.: *Analýza přirozených jazyků* [bakalářská práce], FIT VUT Brno, 2005.

[14] Sipser, M.: *Introduction to the Theory of Computation*. PWS Publishing,
kapitola 2.2: Pushdown Automata, s.101–114. ISBN 0-534-94728-X, 1997

[15] Aho, A. V., Sethi, R., Ullman, J. D.: *Compilers: Principles, Techniques, and Tools*,
Addison-Wesley, 1986. ISBN 0-201-10088-6

[16] Aycock, J., Horspool, R.N.: Practical Earley Parsing. *The Computer Journal*, 2002.

[17] Morneau, R. A.: *The Lexical Semantics of Machina Translation Interlingua*, 2006.
Dokument dostupný na: http://www.eskimo.com/~ram/lexical_semantics.html

[18] Mackenzie, C. E.: *Coded Character Sets, History and Developement*, Boston, MA, USA,
1980, ISBN 0-201-14460-3

[19] Gillam, R.: *Unicode Demystified*, Boston, MA, USA, Addison-Wesley, 2003, ISBN 0-201-70052-2

[20] Krajíček, J., Lodrová, D., Melkes, Z.: *Senzory scannerů v aplikaci OCR* [akademický projekt], FIT VUT Brno, 2006

[21] Bird, S., Klein, E., Loper, E.: *Natural Language Toolkit Documentaion: Chunk Parsing*, 2006. Dokument dostupný pod: http://nltk.sourceforge.net/lite/doc/en/chunk.html.

[22] Martín-Vide C., Mateescu, A., Salomaa, A.: Sewing Contexts and Mildly Context-Sensitive Languages, *Turku Centre for Computer Science*, 1999

[23] Rozenberg G., Salomaa, A.: *Handbook of Formal Language*. SpringerVerlag, Berlin Heidelberg, 1997.

[24] Bird, S., Klein, E., Loper, E.: *Natural Language Toolkit Documentaion: Chart Parsing*, 2006. Dokument dostupný pod: http://nltk.sourceforge.net/lite/doc/en/chart.html.

[25] Wikipedia, the free encyclopedia: *L-system*, 2006. Dokument dostupný pod: http://en.wikipedia.org/wiki/L-System.

[26] Lindenmayer, A.: Mathematical models for cellular interaction in development, *Journal of Theoretical Biology,* s. 280-315, 1968.

[27] Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants (The Virtual Laboratory). Springer-Verlag, 1990, ISBN 0-387-97297-8.

[28] Aaby, A. A.: *Syntax*. Walla Walla College, Walla Walla, Washington, USA, 1998

[29] Serba, I.: *Výtvarná informatika* [přednáškové texty], FIT VUT, Brno, 2006.

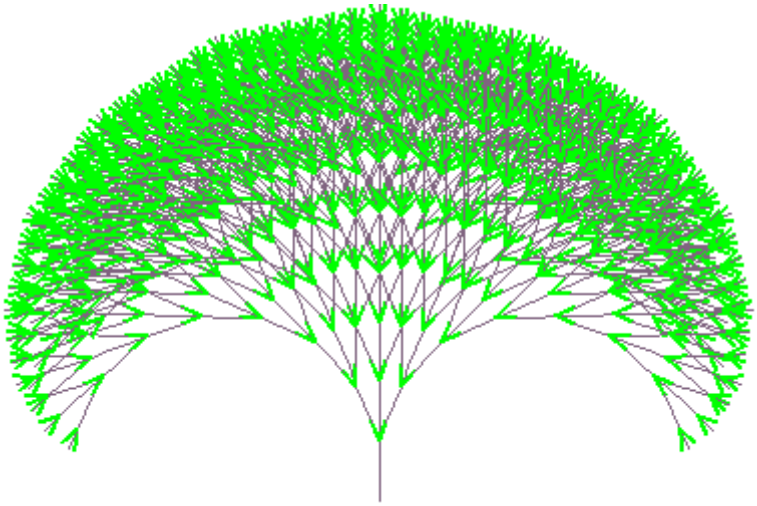
[30] Beck, K.: *Programování řízené testy*. Grada, 2004

Seznam příloh

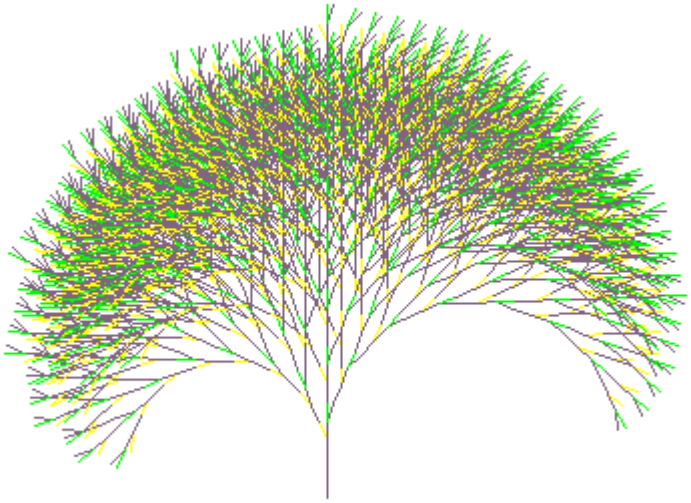
Příloha 1. Ukázky výstupů z implementace zavedeného selektivního L-systému.

Příloha 2. CD, obsahuje zdrojový text této práce a veškeré zdrojové soubory k vytvořenému softwaru a to včetně programové dokumentace.

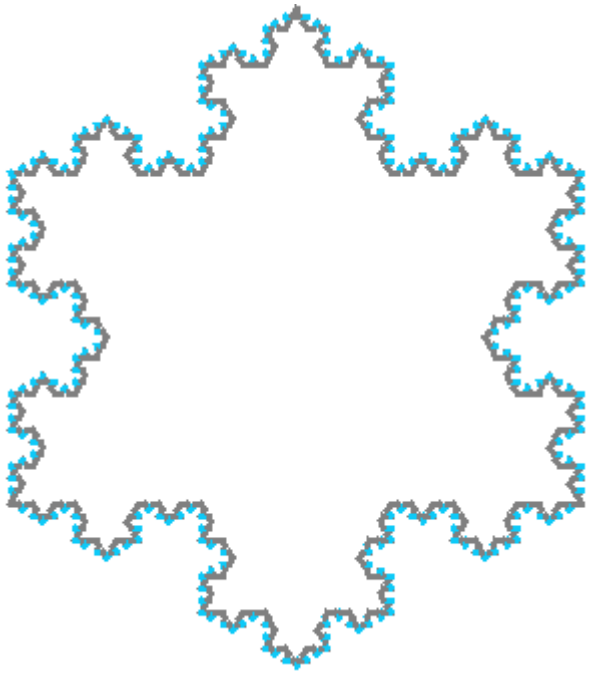
Příloha 1. (výstupy z implementace selektivního L-systému)



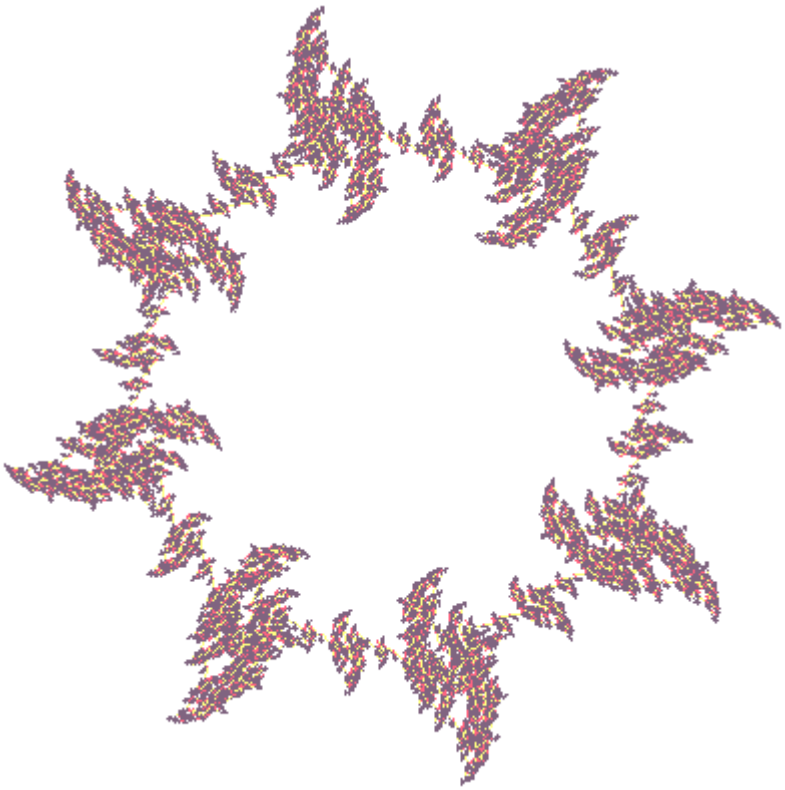
Obr. P. 1 – efflorescent celeriac



Obr. P. 2 – flower mist



Obr. P. 3 – winter Koch curve



Obr. P. 4 – crimson sun