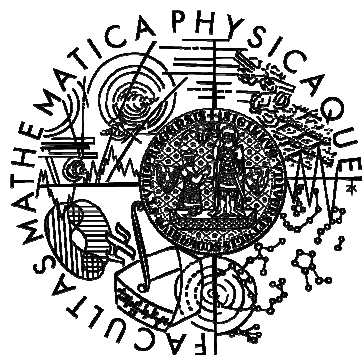


Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Procházka

Vizualizace molekul popsanych v ženevském názvosloví

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Pergel
Studijní program: Informatika, obecná informatika

2007

Věnováno Zdendovi Mertlíkovi

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne

Jan Procházka

Obsah

1. Úvod	5
2. Problematika organické chemie	7
2.1. Názvosloví	7
2.2. Grafická reprezentace molekul	9
3. Dekompozice	14
4. Technická realizace – jádro	16
4.1. Scanner	16
4.2. Parser	17
4.3. Semantika	19
4.4. Model	19
4.5. Grafika	21
5. Technická realizace – rozhraní	24
5.1. run-console	24
5.2. HTTP server	24
6. Problémy implementace	26
6.1. Přidávání triviálních názvů	26
6.2. Třída jazyka organické chemie	26
6.3. Názvy činící komplikace	27
6.4. Konstituce, konfigurace, konformace	27
7. Uživatelská dokumentace	28
7.1. Instalace	28
7.2. Obsluha přes WWW rozhraní	29
7.3. Obsluha z příkazové řádky	30
7.4. Konfigurace	31
8. Závěr	34
8.1. Možnosti rozšiřování	34
8.2. Alternativní implementace	34
8.3. Porovnání s existujícím SW	35
9. Použitá literatura	36
10. Přílohy	37
10.1. Sémantika – jazyk	37
10.2. index.htm	42
10.3. Skripty	43
10.4. Databázové soubory	46

Název práce: Vizualizace molekul popsaných v ženevském názvosloví
Autor: Jan Procházka
Katedra: Katedra aplikované matematiky
Vedoucí bakalářské práce: Mgr. Martin Pergel
e-mail vedoucího: perm@kam.mff.cuni.cz

Abstrakt: Předmětem této práce je implementace nástroje na automatické zpracování názvů organických sloučenin do podoby prostorového modelu molekuly. Pro popis modelu je použit jazyk VRML, což umožňuje mimo jiné následnou práci s ním ve většině nástrojů určených pro práci s prostorovými objekty. Program lze ovládat buďto z příkazové řádky, nebo přes WWW-formulář. Tím je dosaženo jak komfortu při běžném používání programu, tak možnosti jeho využití jinými programy. Důraz byl kladen na zpracování jazyka názvosloví organické chemie a vytvořený model není přesný, avšak pro získání představy o struktuře dostatečný.

Klíčová slova: chemie, názvosloví, modelování

Title: Visualization of Molecules Described in Geneva System
Author: Jan Procházka
Department: Department of Applied Mathematics
Supervisor: Mgr. Martin Pergel
Supervisor's e-mail address: perm@kam.mff.cuni.cz

Abstract: The subject of this work is to implement the tool for automatic processing of the names of the organic compounds into three-dimensional models of the molecules. The VRML language is used for description of the model. It allows (apart from other things) next processing in most tools designated for work with three-dimensional subject. It is possible to control the program from the command line or over the WWW-form. It causes both comfort during common use and possibility of next using by other programs. We focused on processing of the language of the organic chemistry, while the created model is not precise. However, for acquisition of general information about the structure it suffices.

Keywords: chemistry, nomenclature, modeling

1. Úvod

Chemie je dnes jedním z nejrychleji se rozvíjejících a nejvíce využívaných vědních oborů. Uplatnění nachází takřka všude, například v potravinářství (potravinářská chemie), v každé výrobní hale (chemie materiálů), při výrobě léčiv (farmakologická chemie), umělých cév, hnojiv, polovodičů nebo výbušnin.

Chemie se dělí na anorganickou a organickou. Původně toto rozdělení bylo velmi intuitivní, protože vzájemná přeměna mezi sloučeninami organickými (ze živé přírody) a anorganickými (z neživé) byla neznámá, ale od roku 1828 jsou tyto přeměny známy a hranice tedy musí vést jinudy. V současnosti je definována organická chemie jako *chemie sloučenin uhlíku*.

Velmi dlouhou dobu byla takřka veškerá pozornost soustředěna na chemii anorganickou. Poté, co nové znalosti vědecké činnosti v organické chemii začaly nacházet své uplatnění v komerční průmyslové praxi, zájem se přesunul k organické chemii. Dnes pozorujeme přesně opačný trend – neuvěřitelný rozvoj tohoto odvětví a jeho uplatňování v oborech, kde by to dříve nikdo nečekal (například izolační materiály ve stavitelství).

S tím souvisí potřeba rychlejší, ale zároveň přesné komunikace. Pojmenovat 13 milionů známých sloučenin tak, aby podle jména byla snadno rozpoznatelná vnitřní struktura a zároveň si při tom nechat prostor pro jména nově vznikajících sloučenin, není zrovna snadné. Pro tyto účely existuje systematické názvosloví. Vedle systematických názvů existují ještě názvy tradiční, semisystematické nebo triviální, které by sice bylo možné nahradit, ale za cenu ztráty srozumitelnosti (benzen by se označoval jako cyklohexa-1,3,5-trien).

Jako systematické názvosloví se označuje pojmenování sloučenin podle pravidel vytvořených a publikovaných IUPAC (Mezinárodní unie pro čistou a aplikovanou chemii). Český překlad této tzv. „Modré knihy“ (z anglického originálu) tato pravidla zároveň přizpůsobuje potřebám českého jazyka.

Poslední aktualizace systematického názvosloví proběhla v roce 1993 (česky 2000) [1,2]. Vedena byla snahou více zesystematizovat současné názvosloví, odstranit potenciální místa nedorozumění a hlavně sílící potřebou nejen jednoznačnosti, ale také jedinečnosti názvů. Současný stav, kdy pro většinu sloučenin existuje více variant pojmenování, dělá velké problémy například při sestavování rejstříků (vznik křížových odkazů a vícenásobných hesel) [3]. Český

překlad pak využil příležitosti „nutné“ aktualizace a začlenil navíc do nových pravidel změny zápisu tam, kde šlo o neopodstatněnou odchylku od anglické předlohy. Negativními důsledky jsou neznalost a nedodržování – zvláště když se jedná pouze o kodex a nikoliv o normu. Jen málo lidí dnes ovládá správné české organické názvosloví a s chybami je možné se setkat nejen u laické veřejnosti či publicistů, ale dokonce i u lidí přednášejících chemii na vysokých školách nebo na příbalových letácích léků.

Motivací pro vznik této práce byla výše zmíněná problematika. Cílem bylo vytvořit prostředí, ve kterém by si mohl uživatel zobrazit prostorový model organické sloučeniny podle jejího českého názvu. Přičemž práce neměla ambice poskytnout realistický model ani pokrýt celé organické názvosloví (protože obě tyto problematiky jsou značně rozsáhlé), ale byla vedena tak, aby mohla být do takového rozsahu dodatečně rozšířena.

2. Problematika organické chemie

2.1. Názvosloví

Organická chemie pracuje oproti anorganické s mnohem větším množstvím sloučenin. Ty mohou navíc tvořit výrazně složitější struktury. Z toho vyplývá, že i názvosloví je komplikovanější.




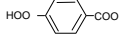
Názvosloví anorganické chemie je (speciálně v češtině) výrazně stručnější. Existuje jen několik málo druhů sloučenin (kyseliny, zásady, soli, ...), navíc tyto sloučeniny z pravidla obsahují jen malý počet atomů, takže není nutné blíže popisovat vnitřní strukturu (je totiž i tak jednoznačná). O tuto přehlednost se zasloužil Dr. Emil Votoček, který využil toho, že je čeština flexní typ syntetického jazyka a užil koncovek k zanesení informace o valencích atomů do názvu.

Názvy v organické chemii se dělí na triviální, semisystematické/semitriviální a systematické. Triviální název je název, jehož žádná část nepochází ze systematického názvosloví (např. močovina). Semisystematický název je název, v němž alespoň jedna část je vytvořena v systematickém smyslu (např. glycerol [ol], butan [an]). Systematický název na rozdíl od triviálního vyjadřuje systematické zařazení popisovaného jevu, v tomto případě se řídí chemickou strukturou látky. S pojmem struktura souvisí další pojmy, a to konstituce (vzájemné spojení atomů vazbami a typy vazeb), konfigurace (uspořádání atomů v prostoru, které není možné měnit volnou rotací kolem jednoduché vazby) a konformace (uspořádání atomů v prostoru, které je možné měnit volnou rotací kolem jednoduché vazby), jimž je pojem struktura nadřazen [4].

Systematické názvy se tvoří pomocí několika názvoslovných operací. Všechny pracují na stejném principu: pomocí předpony (popř. přípony, výjimečně vsuvky) aplikované na původní název, vzniká název sloučeniny po modifikaci příslušnou operací.

Názvoslovné operace se dělí na

- substituční – nahrazení jednoho nebo více atomů vodíku jiným atomem nebo skupinou atomů (upřednostňovaná operace)
(např. CH_4 [methan] \Rightarrow CH_3Cl [chlormethan]),

- záměnné – výměna jedné skupiny atomů nebo jednoho atomu jiného než vodíku za jinou skupinu atomů, resp. za jiný atom.
(např. $\text{CH}_3\text{--CH}_3$ [ethan] \Rightarrow $\text{SiH}_3\text{--CH}_3$ [silaethan]),
- aditivní – formální skládání názvu z částí beze ztráty atomů nebo skupin z kterékoli části
(např.  [naftalen] + 4H \Rightarrow  [1,2,3,4-tetrahydronaftalen]),
- konjunktivní – formální spojení názvů jednotlivých složek; odtržení stejného počtu atomů vodíku z každé složky v každém místě spojení
(např.  [benzen] + 2COOH [mravenčí kyselina] \Rightarrow  [benzen-1,4-dimravenčí kyselina]),
- subtraktivní – odstranění atomu(ů), iontu nebo skupiny, jež jsou implicitně zahrnuty v názvu výchozí sloučeniny
(např. CH_4 [methan] \Rightarrow $\text{CH}_3\text{--}$ [methyl]) a
- násobící – svazy obsahující dvoj- nebo vícevazné substituenty
(např. $\text{N}(\text{CHCl--COOH})_3$ [2,2',2''-trichlornitritrioctová kyselina]).

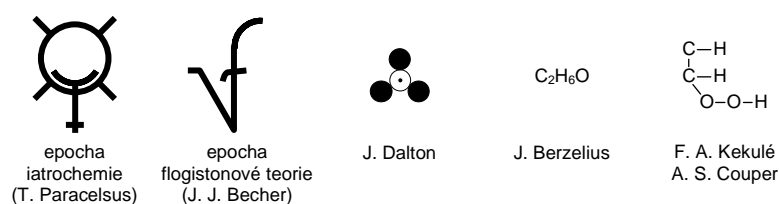
Vzhledem ke složitosti organické chemie (a tedy i jejího názvosloví), existuje několik skupin názvů, které se mohou zdát při nejmenším zavádějící. To samozřejmě komplikuje i jejich automatické zpracování. Patří mezi ně například:

- **rhodanid** – jedná se o zastaralé označení pro thiokyanát a souvislost s rhodiem je jen zdánlivá. (Dále pak existuje látka s triviálním názvem rhodanin, který je synonymem pro 2-thioxothiazolidin-4-on – tedy opět žádné rhodium).
- **per** – infix „-peroxo-“ znamená náhradu --O-- za --O--O-- , oproti tomu „per“ samo o sobě má význam zcela, takže předpona „perhydro-“ vyjadřuje proces nasycení všech vazeb. Druhý z významů je však podle doporučení 1993 již zastaralý a neměl by se tedy užívat.
- **S** – označuje stereodeskriptor chirálních sloučenin, ale také se jedná o značku síry.
- **fluoren** – triviální název tricyklické sloučeniny, který může budít dojem fluoru a dvojně vazby.

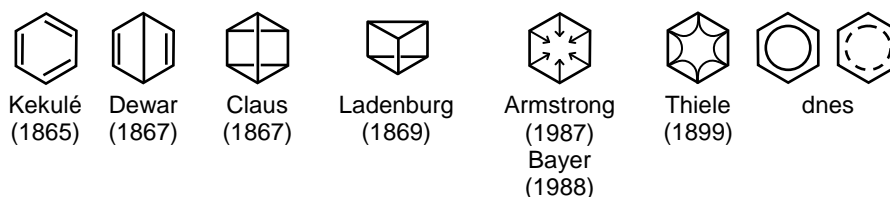
- **propadien** – tento název sice není zavádějící, přesto je problematický – „dien“ je totiž ekvivalentní název pro ethen. „propa-“ je pak možné chápat jako předponu.

2.2. Grafická reprezentace molekul [5]

Při snaze interpretovat chemický název vzniká otázka, co má být výsledkem. Existuje několik v praxi používaných možností. Obecně je možné rozdělit je na rovinné chemické vzorce (obvyklé v literatuře) a prostorové modely. Navíc existuje ještě jakýsi mezistupeň – chemické vzorce, které pomocí projekce znázorňují model.



Obr. 1: Vývoj symboliky sloučenin (jako příklad je uveden ethanol)[6]



Obr. 2: Návrhy vzorce pro molekulu benzenu.

Chemické vzorce jsou dvourozměrné znakové modely sloučenin. Umožňují vyjádřit kvalitativní složení, jak jsou jednotlivé atomy v molekule vzájemně vázány a jaké je jejich prostorové uspořádání. Strukturu lze však zachytit i pomocí jiných symbolů, jako například topologickou maticí orientovaného grafu nebo orbitálního diagramu.

Obecně lze vzorce rozdělit podle míry charakterizace složení a struktury na

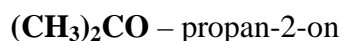
- stechiometrické (empirické) vzorce – vyjadřují pouze poměr, v jakém jsou atomy jednotlivých prvků v molekule zastoupeny (1 se neuvádí)

CH_2O – methanal, ethanová kyselina, glukosa, ...

- molekulové (sumární, souhrnné) vzorce – oproti stechiometrickým vzorcům nezachycují pouze poměry, ale skutečné počty atomů

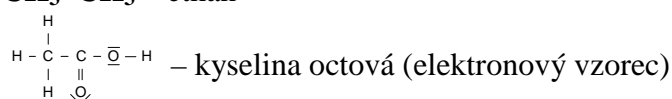
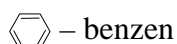
$C_6H_4ClNO_2$ – 2-chlornitrobenzen, 3-chlornitrobenzen, ...

- funkční (racionální) vzorce – vyjadřují jednotlivé charakteristické (dříve funkční) skupiny



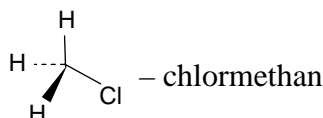
- strukturní vzorce – popisují strukturu, tj. vzájemné vazby a jejich uspořádání; podle toho, zda popisují konstituci, konfiguraci nebo i konformaci se rozlišují

- Konstituční vzorce

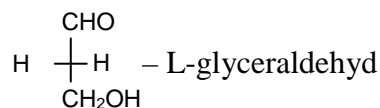


- Konfigurační vzorce

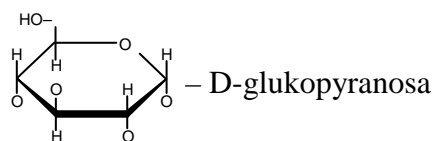
- **Bočná projekce** – směr dopředu je vyjádřen klínem, směr dozadu přerušovanou čarou



- **Fischerova projekce** – vazba směřující pod (nad) rovinu nákresny je vyjádřena vertikálou (horizontálou)

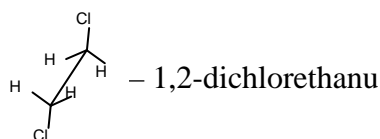


- **Haworthova projekce** – síla vazby (ve smyslu šířky) znázorňuje vzdálenost od pozorovatele

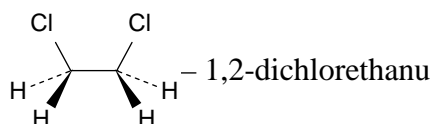


- Konformační vzorce

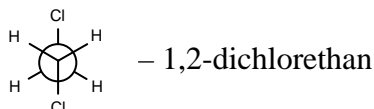
- **Perspektivní projekce**



- **Bočná projekce**



- **Newmanova projekce** – projekce ve směru vazby spojující uhlíkové atomy



- Jiné formy vyjádření struktury

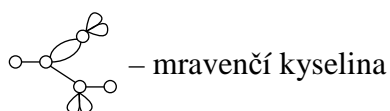
- Distanční matice

	C(1)	O(2)	O(3)	H(4)	H(5)	
C(1)	0	120	134	110	186	– mravenčí kyselina
O(2)		0	225	203	230	
O(3)			0	202	97	
H(4)				0	281	
H(5)					0	

- Incidenční matice

	C(1)	O(2)	O(3)	H(4)	H(5)	
C(1)	0	1	1	1	0	– mravenčí kyselina
O(2)		0	0	0	0	
O(3)			0	0	1	
H(4)				0	0	
H(5)					0	

- Grafy – vrcholy grafu jsou atomy a hrany jsou vazby mezi nimi, volné elektronové páry jsou vyjádřeny smyčkami



Modely molekul je možné rozdělit na materiální a počítačové. Stejně jako u vzorců záleží při výběru typu modelu na účelu jeho použití.

Základními typy materiálních modelů jsou:

- kuličkové modely – atomy jsou reprezentovány kuličkami různé barvy (podle prvku), vazby pak znázorňují tyčky příslušné délky mezi nimi,



– ethan

- kalotové modely – vznikají spojováním čepiček (fr. calotte), poloměry odpovídají van der Waalsovým efektivním poloměrům,



– ethan

- trubičkové modely – jedná se o soustavu trubiček nesoucích význam vazby, kde oblasti příslušící jednotlivým atomům jsou vyznačeny zbarvením,



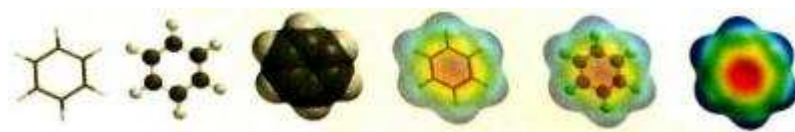
–butan

- Dreidingovy modely – atomy jsou představovány kroužky popř. kuličkami s výstupky, vazby pak trubičkami na ně nasazenými.



Počítačové modely se buď snaží napodobit modely materiální (pak se tedy opět jedná o modely kuličkové, kalotové, ...) nebo se snaží zobrazení kombinovat (např. poloprůhledný kalotový model, uvnitř kterého je zobrazen model trubičkový). Někdy jde ještě dál za „možnosti“ materiálního modelu (např. zobrazení mapy

elektrostatického potenciálu, hustoty elektronů na vazbách, atd.). I tyto modely je možné dále kombinovat mezi sebou.



Obr. 3: Přehled některých druhů počítačových modelů benzenu.

Ať už je k zachycení struktury molekuly použita jakákoliv metoda, je vždy problém u sloučenin se sdílenými elektrony. Jedná se totiž v podstatě o jednu vazbu mezi více než dvěma atomy. Pro benzenová jádra existuje u konstitučních vzorců speciální symbol – kroužek (viz obr. 2), ale například ozon již žádnou takovou značku nemá. Z nouze se pak v takových případech používá některý z tzv. „rezonančních hybridů“, což je ovšem trochu zavádějící (výsledná molekula je totiž kombinací všech těchto hybridů). Tento problém vychází z nedostatků Lewisovy teorie lokalizovaných vazeb, která je základem všech výše zmíněných chemických značení. Představou rezonance lze odstranit jen některé z problémů Lewisovy teorie, pro další lze použít představu hypervalence nebo elektronového deficitu, přesto však u některých molekul nastávají potíže [10].

3. Dekompozice

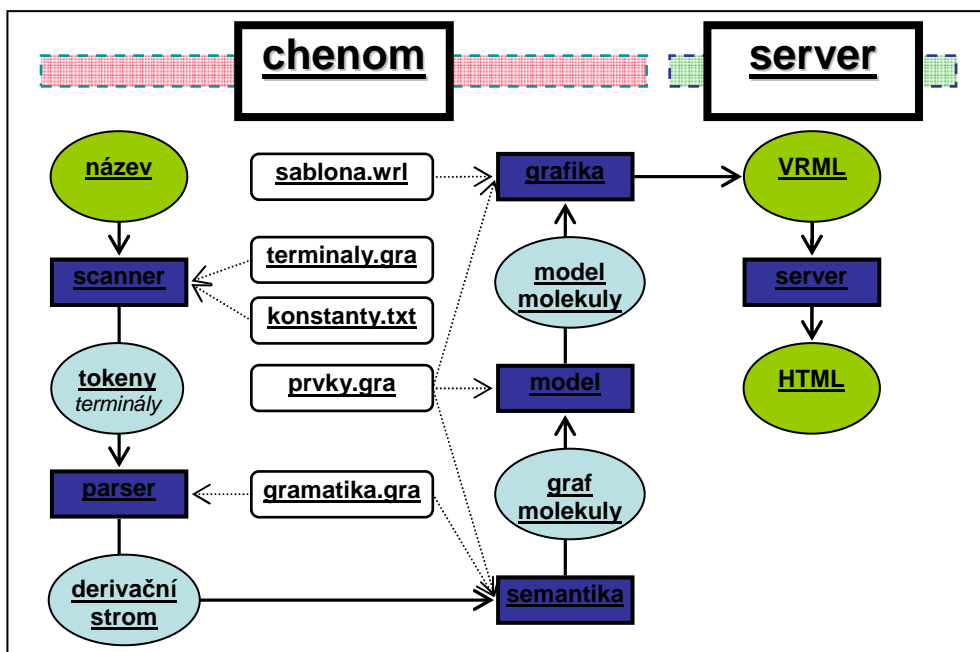
Protože je program rozsáhlý a jeden z vytyčených cílů byla modularita, byl rozčleněn na několik relativně (až na datové struktury a nutný interface) nezávislých částí. Svou koncepcí se program blíží překladači.

Vstup je nejprve zpracován modulem `Scanner` (lexikální analýza), který ho rozdělí na tokeny a vrací jejich číselné kódy definované v souboru „terminaly.gra“. Navíc řeší základní kontextové problémy (vypouštění a vkládání samohlásek), k čemuž využívá soubor „konstanty.txt“. Tokeny vstupují jako terminály do dalšího modulu zvaného `Parser` (syntaktická analýza). Zde je z nich vystavěn derivační strom dle gramatiky v souboru „gramatika.gra“. V tomto souboru navíc následuje za každým pravidlem sekvence instrukcí, které se v případě použití pravidla aplikují v příslušném uzlu. To zajistí modul `Semantika` (sémantická analýza). Výstupem tohoto modulu je již graf molekuly. Ten je třeba umístit do prostoru, aby vznikl model, což zajišťuje modul `Model`, který pomocí volání modulu `Grafika` zajistí i výstup ve formátu VRML [7] s parametry definovanými v souborech „prvky.gra“ (zde jsou veškeré potřebné parametry jednotlivých prvků – náboj, váha, přípustné elektronové konfigurace, poloměr a barva) a „sablon.wrl“ (zde je uložena celá hlavička výsledného dokumentu, to znamená Viewpoints, uzly PROTO pro atomy a vazby, zvolené kódování, atd.).

Jména všech použitých souborů jsou definována v souboru „settings.ini“, který program dostane jako parametr.

Pro větší uživatelské pohodlí byla implementována dvě rozhraní. Konzolový skript „run-console.bat“ (popř. „run-console.sh“) přijímá název molekuly jako parametr a vytvořený VRML model zobrazí v aplikaci asociované s příponou „wrl“. Druhý interface pak tvoří jednoduchý HTTP server, který přijímá dotazy na stránku s modelem molekuly, poté zavolá skript „run.bat“ (popř. „run.sh“) s názvem v parametru. Pokud takový ještě nebyl vytvořen, tak ho vytvoří (pomocí volání výše popsaného jádra) a vrátí stránku, která model obsahuje.

Z výše uvedeného je zřejmé, že program potřebuje pro svou plnohodnotnou práci buď plugin pro zobrazování VRML do nějakého internetového prohlížeče (v případě užití druhého interface) nebo jakýkoliv jiný program schopný interpretace VRML souborů (v případě užití prvního – ale v podstatě i druhého – interface).



Obr. 4: Schéma zapojení programových modulů a jejich komunikace

4. Technická realizace – jádro

Tato kapitola obsahuje podrobný rozbor implementace modulů, ze kterých se skládá jádro programu, a jejich vzájemné komunikace. Těmito moduly jsou Scanner, Parser, Semantika, Model a Grafika.

4.1. Scanner

Jak již bylo řečeno, tento modul má za úkol provést lexikální analýzu požadovaného názvu. To není vhodné učinit standardním způsobem, totiž postavením automatu, který je resetován po každém přijatém tokenu, protože toto předzpracování by bylo (vzhledem k počtu tokenů, které budou rozpoznány) příliš časově náročné.

Bylo tedy použito jednodušší řešení. Program postaví z tokenů načtených ze souboru „terminaly.gra“ lexikální strom. Při zpracovávání názvu se v něm hledá vždy nejdelší cesta od kořene odpovídající dosud nepřečtenému tokenu. Při dosažení uzlu ve stromě, který již nemá přechod přes následující znak v názvu či při dosažení konce názvu se program vrátí zpět do posledního uzlu, který je označen jako výstupní. Z odpovídajícího místa v názvu pak pokračuje nové hledání. Pokud se stane, že je při návratu po větvi dosažen kořen, znamená to, že se zbývající část názvu nepodařilo určit. Program se vrátí do posledního určeného tokenu a pokračuje se v hledání (směrem ke kořenu). Pokud je dosažen kořen při hledání prvního tokenu, znamená to, že se programu nepodařilo název na tokeny rozložit a je vyvolána chybová hláška, v níž je uveden nejkratší sufix, kterému se nepodařilo porozumět.

Další zvláštností implementace tohoto modulu je, že nevyužívá sdružování tokenů. Buď by totiž muselo být jejich rozlišení v sémantické analýze implementováno pevně, nebo by to vyžadovalo rozšíření jazyka sémantické analýzy o jejich rozlišování a to kvůli ušetření jednoho (sdružovacího) neterminálu. Přesto je interface tohoto modulu pro sdružování připraven, protože používá soubor „konstanty.txt“, který přiřazuje intervalům tokenů jména.

Tento soubor je navíc využit k ošetření některých kontextových problémů. Prvním je vkládání a vypouštění samohlásek ve švu slova, tj. vypouštění koncového „a“ nebo „o“ v tokenu, popřípadě vkládání „o“ mezi tokeny. Současná implementace vypadá tak, že koncové „a“ je možné vypustit kdykoliv a „o“ je možné vložit do švu, pokud se jedná o jméno prvku, charakteristickou skupinu, řeckou číslovku nekončící samohláskou, kořen triviálního názvu nebo celý triviální název. Druhý kontextový

problém ošetřený v tomto modulu je existence samostatných hlásek jako tokenů (např. „a“ v „4a,9a-but[2]enoanthracen“ nebo v „dibenzo[a,j]anthracen“). Tyto tokeny nesmějí následovat po písmenu, protože to by umožňovalo vracet jako tokeny pouze jednotlivé hlásky (například „methan“ by bylo rozděleno jako „metha|n“ místo „meth|an“) a chyba by byla vždy vyvolána až v syntaktické analýze, namísto v lexikální.

Práce s tímto modulem probíhá pomocí několika funkcí. Nejprve je potřeba načíst intervaly tokenů z „konstanty.txt“, to zajistí procedura `scanner::inicializace_intervalu`, která v parametru očekává otevřený soubor. Následuje načtení seznamu tokenů, přičemž se z nich postaví lexikální strom. Pro tyto potřeby slouží procedura `scanner::inicializace`. Poté se volá procedura `scanner::parse` se zpracovávaným názvem v parametru. Ta ho rozdělí na tokeny, ale pouze v bufferu. Tyto tokeny jsou pak vráceny postupně včetně svých flagů a pořadového čísla ve funkci `scanner::yylex`. Obsaženy jsou ještě dvě funkce, `scanner::hodnota` a `scanner::text`, které převádí číselný kód terminálu na jeho jméno a naopak. To je využito hlavně v modulu `Parser` k načtení gramatiky.

4.2. *Parser*

`Parser` je modul zajišťující syntaktickou analýzu názvu. Nejprve se provede inicializace a to tak, že se načte ze souboru „gramatika.gra“ vše potřebné, tedy gramatická pravidla a jejich sémantický význam (sekvence příkazů). Při načítání se rovnou převádí hodnoty terminálů na kódy tokenů a stejně tak jména neterminálů. Těm je navíc potřeba ještě kódy přiřazovat, což se děje v pořadí, v jakém se vyskytují na vstupu. Sémantická pravidla se nijak nezpracovávají, ponechávají se ve své textové podobě. O jejich interpretaci se stará až v případě potřeby modul `Semantika`. Inicializace je zakončena testem, zda nejsou obsaženy neterminály, pro které neexistuje žádné odvozovací pravidlo a případně se tento nedostatek ohlásí uživateli.

Práce při hlavním volání vypadá následovně. Program dokola volá `scanner::yylex` dokud dostává tokeny a přitom z nich tvoří graf typu cesta, kde tokeny tvoří orientované hrany. Následuje budování derivačního stromu nad tímto grafem. To probíhá tak, že se postupně prochází pravidla a u každého se zjišťuje, zda je možné ho aplikovat, tj. zda pravá strana pravidla odpovídá orientované cestě v dosavadním grafu. Pokud ano, přidá se hrana spojující začátek a konec této cesty,

přičemž její hodnota odpovídá kódu neterminálu na levé straně pravidla. Takto se hledají hrany, dokud existuje možnost, že se nějaká přidá a zároveň se ještě nepodařilo aplikovat pravidlo, které by vytvořilo hranu počátečního neterminálu přes celou původní cestu.

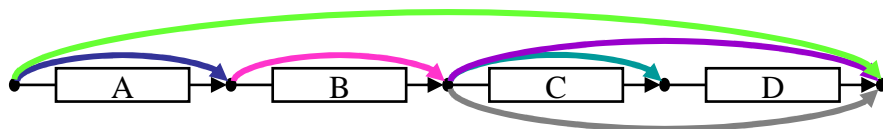
Zároveň s tímto grafem se buduje derivační strom a to tak, že vrcholy jsou tvořeny hranami původního grafu a hrany v něm vedou z A do B, pokud byla hrana odpovídající B použita při vytváření hrany odpovídající A. Zároveň se odstraní všechny vrcholy (a do nich, případně z nich, vedoucí hrany), které nejsou součástí hlavního stromu.

```

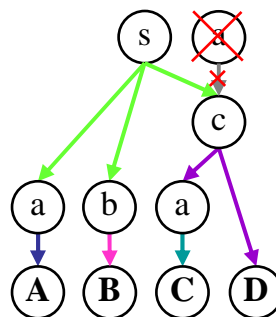
a ::= A | C | c ;
b ::= B+b | B ;
c ::= a+D ;
s ::= a+b+c ;

```

Obr. 5: Příklad gramatiky



Obr. 6: Vygenerovaný graf pro slovo ABCD podle gramatiky z obr. 5



Obr. 7: Výsledný derivační strom

S ostatními moduly probíhá komunikace následovně. K inicializaci slouží procedura `parser::inicializace`, která načte gramatiku ze souboru, který očekává otevřený ve svém parametru. Pro překlad kódu symbolu (terminálu nebo neterminálu) na jeho textovou reprezentaci slouží funkce `parser::kod2str`. To je využito pouze při chybových výstupech. Komunikaci s okolím zajišťuje ještě jedna funkce a tou je `parser::yyparse`, která vrací ukazatel na typ `parser::derivacni_strom::derivat`. Tato třída je velmi jednoduchá, obsahuje pouze seznam ukazatelů na potomky, iterátor na pravidlo gramatiky, které je daným

uzlem reprezentováno a rozsah indexů tokenů, jež do daného podstromu spadají – tedy veškeré informace potřebné k dalšímu zpracování.

4.3. *Semantika*

Tvorba modelu molekuly je započata vytvořením jejího grafu. To se děje v modulu *Semantika*. Ten nejprve zavolá funkci `parser::yyparse`, která vrátí derivační strom zpracovávaného názvu, ten vyhodnotí a strom zruší. Výsledek vyhodnocení je poté vrácen k dalšímu zpracování.

Vyhodnocování probíhá tak, že se volá rekurzivně na podstromy a z navrácených dat se staví pole. V aktuálně zpracovávaném uzlu se pak vezme sekvence příkazů jemu příslušící (z gramatiky v modulu *Parser*) a ta se počne aplikovat na předem vytvořené pole. Jak postupují mezivýsledky stromem nahoru, nakonec se dostane výpočet i do kořene, který svůj výsledek vrátí jako výsledek celého vyhodnocování. Návrátová hodnota je struktura, která mimo jiné obsahuje položku typu `semantika::skeleton`, ve které je uložena požadovaná molekula (tedy pouze její struktura, jména atomů, atd.).

Celkově se postupuje tak, že se molekula buduje bez vodíků a až když je vše ostatní hotovo, připojí se na „volné“ pozice vodíky. Pokud se stane, že pravidlo některého uzlu neobsahuje žádné příkazy, pak je na tuto situaci uživatel upozorněn. Jazyk sémantiky je velice jednoduchý, aby byla jeho interpretace co nejsnazší. Jeho podrobný popis je v Příloze 10.1.

Pro správnou interpretaci je nutné znát některé základní charakteristiky použitých prvků. Ty jsou načteny ze souboru „*prvky.gra*“ při inicializaci (procedura `semantika::inicializace`, která očekává v parametru otevřený soubor). Analýza se pak spouští pomocí volání funkce `semantika::yysem`, která vrací graf molekuly typu `semantika::skeleton`.

4.4. *Model*

Z grafu molekuly, který vrátí modul *Semantika* je potřeba vytvořit model. To znamená určit polohy jednotlivých atomů v prostoru, což zajišťuje tento modul. Zmíněnou úlohu je velmi obtížné přesně vyřešit a tak byl zvolen následující model.

V molekule dominují dvě síly. Atomy se v důsledku elektrického náboje odpuzují, ale jako protiváha je drží pohromadě vzájemné vazby. Stejně tak působí v použitém modelu dvě protichůdné síly. Každá dvojice atomů na sebe působí podle

Coulombova zákona, přičemž jejich náboje jsou přímo úměrné protonovému číslu Z (což odpovídá realitě). Vazebné síly jsou pak modelovány jako guma napínaná mezi atomy. Využity jsou tedy následující dva vzorce. Coulombův zákon:

$$F_e = \frac{1}{4\pi\epsilon} \frac{Q_1 Q_2}{r^2}, \text{ kde}$$

- ϵ je permitivita vakua (konstanta),
- Q_i jsou náboje jednotlivých atomů ($Q_i = e_0 Z_i$, kde e_0 je náboj elektronu),
- r je vzdálenost mezi atomy.

Druhý vzorec pak popisuje sílu mezi vázanými atomy následovně:

$$F_p = K \Delta y, \text{ kde}$$

- K je lineární tuhost (materiálová konstanta),
- Δy je vzdálenost mezi atomy zmenšená o délku gumy (ta je stanovena experimentálně na hodnotu 4,0).

Dalšího zjednodušení bylo dosaženo nahrazením spojitě simulace diskrétní variantou. Program v daném okamžiku vždy spočítá výslednici sil působících na atomy (F). Tuto sílu pak nechá konstantně působit po dobu t_i (jedná se tedy o rovnoměrně zrychlený/zpomalený pohyb). Takto je zjištěna nová poloha ($P_{i+1} = P_i + v_i t_i + F_i t_i^2 / 2m$) a rychlost ($v_{i+1} = v_i + F_i / m$) atomů a výpočet se opakuje, tentokrát se však síly nechají působit po dobu t_{i+1} . Posloupnost t_i je navržena tak, aby výpočet nutně konvergoval (použita byla geometrická posloupnost s kvocientem 0,9998 a hodnotou prvního členu 0,01).

Ačkoliv to není zřejmé, významný vliv na výsledek má i inicializace. Je potřeba, aby proběhla co nejrychleji, zároveň však musí umožnit rychlý přesun všech atomů do „správné“ (tj. v rámci modelu do nejkvalitnější možné) konfigurace. Nejjednodušší varianta, jak atomy rozmístit, je umístit je do řady vedle sebe (například na souřadnice $[4i, 0, 0]$). To by ale mělo za důsledek, že by všechny působící síly měly směr rovnoběžný s touto přímkou, tudíž by všechny atomy zůstaly na této přímce. Podobný výsledek by byl dosažen, pokud by všechny atomy ležely v jedné rovině. Naopak asi nejlepší výsledky by obecně dávalo rovnoměrné rozmístění na sféru (jejíž poloměr by ale musel nějak korelovat s počtem atomů).

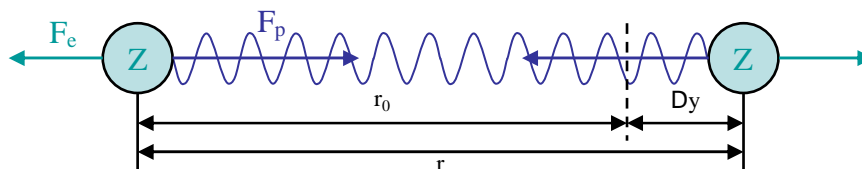
To by ale vyžadovalo zbytečně složité výpočty. Bylo tedy zvoleno a implementováno kompromisní řešení:

$\{[x_i, y_i, z_i]\}_i := [i, i(-1)^i, i^2(-1)^i]$, kde $[x_i, y_i, z_i]$ jsou souřadnice i -tého atomu.

Pro další zjednodušení modelu (resp. počítání v něm) se neuvažují skutečné hodnoty konstant (náboj, hmotnost, permitivita vakua, ...), ale jsou nahrazeny hodnotami vyššími. Tím, že jsou takto upraveny všechny konstanty ve správném poměru, je zachováno chování modelu, ale výpočty jsou přesunuty z řádu 10^{-27} do řádu jednotek, kde je dosahováno vyšší přesnosti. Výpočty tedy probíhají podle následujících vzorců:

$$F_e = \frac{Z_1 Z_2}{r^2}$$

$$F_p = r \left(1 - \frac{4,0}{r} \right) \quad (= \Delta y)$$



Obr. 8: Vzájemná interakce mezi dvěma atomy spojenými vazbou podle modelu

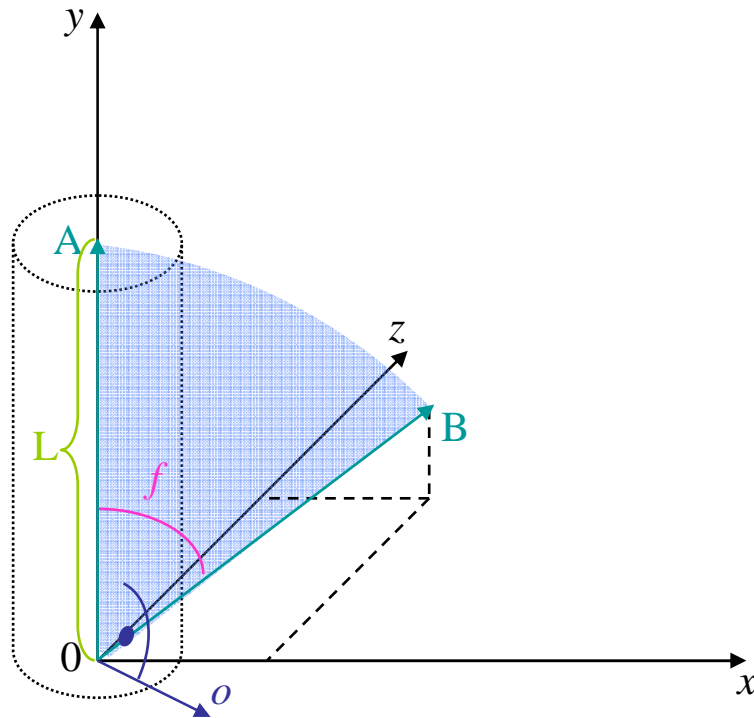
Interface tohoto modulu není příliš rozsáhlý. Hlavní je metoda `model::yymodel`, která vypíše pomocí následujícího modulu Grafika výsledek své práce – vytvořený model molekuly. Ten vznikne tak, že se načte graf molekuly z parametru do vlastní třídy `model::skeleton` a zavolá se její metoda `optimize`, která pracuje výše popsaným způsobem.

4.5. Grafika

Tento modul zajišťuje výstup. Slouží k tomu jeho funkce `grafika::tisk`, jejíž parametr je model a vrácená hodnota odpovídá jeho VRML zápisu.

V hlavičce (která je v souboru „sablona.wrl“ a načítá se do paměti funkcí `grafika::inicializace`) jsou předdefinovány pohledy (VIEWPOINT) a prototypy (PROTO) pro Atom (`Atom`), vazby jednoduchou, dvojnou, trojnou a chybnou (`Vazba1`, `Vazba2`, `Vazba3`, `Vazba_err`)(viz Příloha 10.4). Následují definice atomů skutečně použitých a to jako specifikace obecného Atom (je určena barva a velikost). Poté již je vypsána konkrétní molekula. Pro snazší zacházení nejsou struktury vnořeny (stejně by bylo obtížné určit, kterému ze dvou vázaných atomů vazba náleží).

Pro určení transformace vazby do požadované polohy, bylo třeba stanovit osu a úhel otočení ze své počáteční (s osou y rovnoběžné) pozice. Pro snazší manipulaci je vazba (válec/válce) nejprve posunuta dolní podstavou do počátku (v generované pozici leží počátek souřadného systému uprostřed vazby). Nakonec je posunut do prvního z vázaných atomů (zde se projeví výhoda prvního posunu).



Obr. 9: Výpočet parametrů pro otočení vazby do správného směru

Výpočet parametrů otočení (tedy osy a úhlu) je relativně jednoduchý a vychází z následujících předpokladů:

Značení: A – původní směr vazby, B – transformovaný směr vazby, o – směr osy rotace, φ – velikost rotace, L – délka vazby ($= \|A\|$).

$$o = B \times A$$

$$\cos(\varphi) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

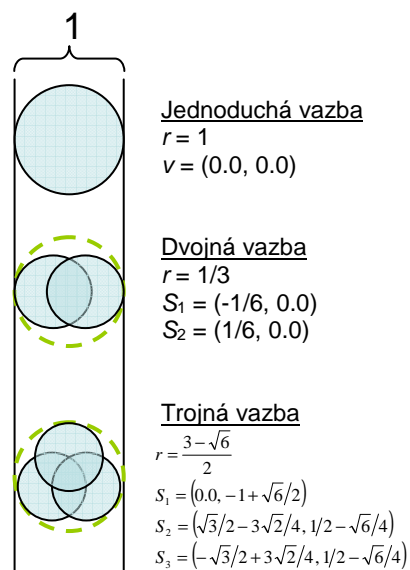
$$A \times B = [a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1]$$

$$A = [0, L, 0]$$

Po dosazení posledního vztahu do předchozích a následném vyjádření neznámých jsou získány kýžené parametry transformace:

- $o = (-b_3, 0, b_1)$ (u osy je nutné zajistit pouze směr vektoru, nikoliv jeho velikost, proto je možno celý výraz zjednodušit o jistě nenulovou hodnotu L),
- $\varphi = \arccos\left(\frac{b_2}{\|B\|}\right)$.

Násobnost vazby je vyznačena počtem válců. Aby byl výsledný vzhled přirozený, bylo potřeba dosáhnout opticky shodné síly vazeb, zároveň však bylo potřeba učinit je snadno rozlišitelnými. To se snad podařilo tak, že průřezu každé vazby lze opsat jednotkovou kružnicí a střed podstavy každého válce leží na hranici válců ostatních (viz obr. 10).



Obr. 10: Odvození tvaru násobných vazeb v modelu

5. Technická realizace – rozhraní

V předchozí kapitole je popsáno jádro celého programu. To ovšem pracuje pouze na příkazové řádce, což sice pro vstup nevádí, ale výstup, tj. 3D model molekuly, je potřeba nějak interpretovat. Implementována jsou dvě řešení.

5.1. *run-console*

Jedná se o velice jednoduchý skript, který očekává jako parametr jméno požadované molekuly. Zjistí, zda již model existuje ve složce „outs“. Pokud ne, je za pomoci jádra na tomto místě vytvořen. Za situace, že se toto podařilo, je vytvořený model otevřen definované aplikaci (schopné zobrazit soubory VRML). Defaultně je tímto programem internetový prohlížeč.

5.2. *HTTP server*

Ačkoliv se jedná spíše o koncový modul celého programu, byl pro vyšší variabilitu implementován jako samostatný program. Jedná se o malý a co do funkčnosti velice omezený HTTP server [8].

Při startu je načtena šablona stránky (tj. obsah souboru „index.htm“, kde v místech pro nadpis a pro tělo je text `$NAZEV$`, popř. `$TELO$`). Poté server začne poslouchat požadavky přicházející na port, jehož číslo bylo zadáno jako parametr (doporučeno je číslo běžně užívané pro HTTP servery – 80). Kladně jsou vyřízeny pouze následující požadavky:

- `GET /chenom`,
- `GET /chenom?mol=nazev_molekuly` – pokusí se vytvořit molekulu s daným názvem (pokud se to podaří, vrátí stránku, která molekulu obsahuje, jinak se na stránce zobrazí seznam vygenerovaných chybových hlášení),
- `GET /outs/nazev_molekuly.wrl` – vrátí model, který existuje,
- `GET /STOP` – ukončí server,
- `GET /RESET` – způsobí znovunačtení šablony ze souboru „index.htm“.

V ostatních případech je vrácena stránka s chybovou hláškou.

V druhé variantě dotazu je pro zajištění existence dotazovaného modelu molekuly použit skript „run“, který volá jádro `chenom` s volitelným parametrem `-e`. Ten způsobí, že je na vstupu očekáván oescapovaný (pomocí „`htmlencode`“) název. Výstup je poté uložen do stejnojmenného souboru do složky „outs“. Chyby

jsou ukládány do souboru „error.log“, jehož obsah je v případě neúspěchu generování zobrazen uživateli.

Server je navrhován jako localhostová aplikace a jako taková nemá velké nároky na provoz. Navíc zpracovávání dotazu může trvat i několik minut při 100% zátěži procesoru. Z těchto důvodů bylo při implementaci přistoupeno k jednovláknové variantě.

6. Problémy implementace

Je zřejmé, že není možné implementovat obecnou a úplnou interpretaci organického názvosloví. Při práci na programu CheNom však byla vyvinuta snaha o možnost pozdějšího rozšíření o všechny (více či méně dobrovolně) opomenuté aspekty a to nejlépe bez nutnosti měnit kód programu (a tedy recompileovat), ale pouhým zásahem do datových souborů. Z tohoto přístupu vplynuly následující problémy.

6.1. Přidávání triviálních názvů

Protože není možné do programu zanást všechny triviální názvy, je nutné umožnit jejich co nejsnazší přidávání. Tento požadavek přinesl několik komplikací. Modifikace zdrojového kódu programu rozhodně není snadné přidání. Je tedy nutné toto provádět ve vnějším (datovém) souboru. Nejedná se však pouze o triviální názvy, uživatel může potřebovat přidat třeba funkční skupinu. Pak je tedy potřeba umístit do vnějšího souboru celou gramatiku. Tím odpadá možnost využití pomocných nástrojů jako flex či bison. Navíc je nutné v tomto souboru s gramatikou popsat sémantiku jednotlivých pravidel. Pro tento jazyk je nutné vytvořit interpret, a proto je vhodné vytvořit malý, snadno analyzovatelný jazyk s výrazovými prostředky přesně vytvořenými pro potřeby popisu sémantiky názvotvorných operací.

6.2. Třída jazyka organické chemie

Existence úprav jako nahrazení koncovky „-ová kyselina“ koncovkou „-át“ nebo vkládání infixů do názvů (např. záměnný infix „-imido-“) dělá jazyk organické chemie výrazně kontextovým. Tato skutečnost velice komplikuje jak lexikální, tak syntaktickou analýzu. Situaci neprospívá ani nejednoznačnost pravidel. Například pravidlo R-0.1.7.3 popisuje vkládání „o“ následovně: „*Pro lepší výslovnost a libozvučnost se někdy mezi souhlásky předpony vkládá samohláska ,o‘*“ [2]. Také obraty jako „*zvažuje komise*“ nebo „*bude popsáno v budoucí publikaci*“ nejsou výjimečné [2].

Některé z těchto problémů se podařilo odstranit v lexikální analýze. V kondenzovaných polycyklech se používají jako lokanty písmena latinky. Při bezkontextovém zpracování je tak možné rozdělit každý název tak, že jednomu tokenu odpovídá jeden znak. Částečně lze tomuto zabránit tím, že se místo obvyklé

varianty použití prvního (nejkratšího) nalezeného tokenu použije nejdelší. I tak by ale například analýza názvu „methan“ vrátila tokeny „metha“ a „n“. Kontextové pravidlo umožňující vrátit token odpovídající jednomu písmenu latinky pouze v případě, že předchozí token nekončí písmenem, řeší i tento nedostatek. Dalším kontextovým pravidlem uplatněným v lexikální analýze je již zmíněné vypouštění a vkládání samohlásek.

Problémy s kontextovostí infixových názvotvorných operací se nepodařilo odstranit, a proto bylo od jejich implementace upuštěno.

Syntaktická analýza byla zvolena pro co nejširší třídu jazyků a tedy bezkontextová nedeterministická na způsob Q-systémů.

6.3. Názvy činící komplikace

Sestavení gramatiky názvosloví organické chemie tak, aby pokud možno neobsahovala žádné konflikty (a to ani po jejím rozšiřování) je velmi náročné. Například HI lze označit jako jodan, I⁻ pak jako jodyl. Jodid se zdá být obdobou jodylu a tedy Γ. Ve skutečnosti je však jodid název charakteristické skupiny I⁻ a pro Γ je přípustné pouze označení jodanid.

Větším problémem je spojení „karboxylová kyselina“. Jak je totiž methanal a methylaldehyd dvojí označení téhož, tak v důsledku shody funkčního skupinového názvu a substituční přípony charakteristické skupiny představují názvy „methankarboxylová kyselina“ a „methylkarboxylová kyselina“ tutéž sloučeninu. Ovšem byla-li by chápána druhá varianta ve smyslu první, pak by byla na připojovaném uhlíku vytvořena nevyužitá volná vazba (v takovém případě by bylo správnější označení „1-oylmethylkarboxylová kyselina“).

Mnoho komplikací je také způsobeno nevhodnými triviálními názvy. Například dvojice ethenyl (taktéž vinyl, CH₂=CH-) a ethylen (-CH₂-CH₂-).

6.4. Konstituce, konfigurace, konformace

Implementována musí být konstituce, protože jeden sumární vzorec může označovat velké množství látek naprosto odlišných vlastností. Aby mohla být implementována konfigurace, je nutné správně modelovat (nemá smysl řešit, zda má být atom umístěn nad nebo pod rovinou cyklu, pokud je při modelování umístěn do této roviny). Stejně jako konfigurace ani konformace nebyly implementovány. Hlavní důvod je ten, že v drtivé většině případů nemá názvosloví nástroje je od sebe odlišit (například vaničková a židličková konformace cyklohexanu).

7. Uživatelská dokumentace

Program CheNom je vytvořen za účelem vizualizace organických molekul podle jejich jména. Zdrojové kódy byly psány tak, aby bylo možné program provozovat jak pod operačními systémy Windows (98+), tak pod těmi z rodiny Linux.

Tato kapitola by měla seznámit čtenáře s možnostmi programu a jejich využíváním od instalace přes běžné použití až po pokročilou konfiguraci.

7.1. Instalace

K provozování programu je nutné mít nainstalovaný VRML prohlížeč, a to nejlépe integrovaný do internetového prohlížeče, který zvládá dobře interpretovat jména souborů s jinými než alfanumerickými znaky. To umožňuje použít WWW rozhraní programu. Pokud se jedná o standalone aplikaci, je možné program používat z příkazové řádky.

Windows

1. Nainstalujte internetový prohlížeč splňující výše uvedenou podmínku (Internet Explorer ne) včetně pluginu pro VRML.
2. Zkopírujte z příloženého CD složku `/windows/CheNom` na místo, kde si přejete program mít.
3. Pokud jste jako prohlížeč nezvolili Firefox, pak otevřete soubor `START.bat` v textovém editoru (např. Poznámkový blok) a slovo `firefox` nahraďte příkazem pro spuštění vašeho prohlížeče. Změny uložte.
4. Program se spouští pomocí obsaženého skriptu `START.bat`, popř. `run-console.bat` pro ovládání z příkazové řádky.

Linux

1. Nainstalujte VRML plugin pro váš internetový prohlížeč.
2. Zkopírujte z příloženého CD složku `/Linux/CheNom` na místo, kde si přejete program mít.
3. Spusťte v dané složce příkaz `./compile`.
4. Program se spouští pomocí obsaženého skriptu `START.sh`, popř. `run-console.sh` pro ovládání z příkazové řádky.

7.2. **Obsluha přes WWW rozhraní**

Po spuštění serveru je možné přistupovat k němu přes internetový prohlížeč. WWW adresa, na níž je služba dostupná je `http://pocitac:port/chenom`, tedy pro počítač, na němž je program spuštěn na portu 80 (standardní spuštění) je to **`http://127.0.0.1:80/chenom`**, ale protože port 80 se může vynechat (std. port pro HTTP protokol), stačí použít **`http://127.0.0.1/chenom`**. Pozor, na linuxových systémech smí na portech nižších než 1024 spouštět služby pouze správce! Pokud jím nejste, obvykle se postupuje tak, že se použije port 8080.

Obsluha je velmi jednoduchá. Stačí do bílé kolonky ve stránce vyplnit jméno sloučeniny a stisknout ENTER, případně tlačítko **Zobrazit**. Pro vkládání speciálních znaků a formátování jsou pod touto kolonkou tlačítka. Modrá tlačítka slouží k formátování (jmenováno zleva se jedná o kurzívu [`<i>`], dolní index [`<sub>`] a horní index [`<sup>`]) a k vkládání mezery [` `]. Přesně tak, jak se tyto symboly používají v HTML. Pro vkládání malých řeckých písmen slouží zelená tlačítka [`α`; – `ω`].

Po stisknutí odeslání dotazu je tento zpracován a nastat mohou dvě alternativy. Buď je zadané jméno vyhodnoceno jako chybné, není vygenerován žádný model a je vrácena stránka obsahující seznam všech vygenerovaných chybových hlášení, nebo vše proběhne úspěšně, podaří se vygenerovat model zadané molekuly a vrácena je stránka, která jej obsahuje v tagu `<embed>`.

Program se ukončuje kliknutím na červené tlačítko **KONEC** v pravém horním rohu. Jedná se o odkaz vedoucí na **/STOP**.

Zde vyplňte jméno molekuly určené k zobrazení: KONEC

Zobrazit i sub sup _

α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω

adamantan

Obr. 11: Takto vypadá vygenerovaná stránka

7.3. **Obsluha z příkazové řádky**

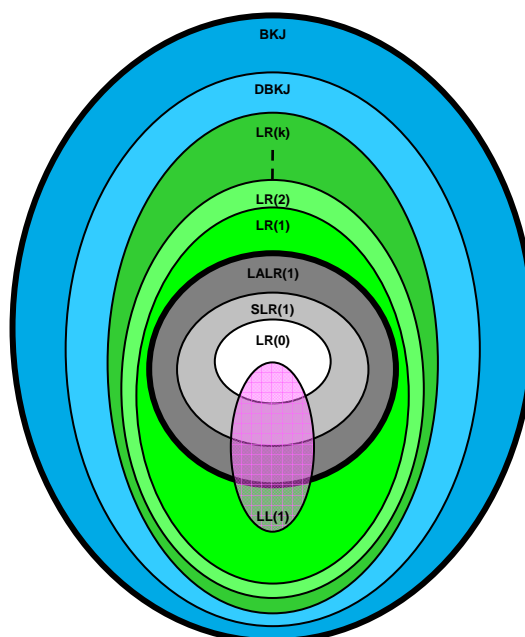
K obsluze z příkazové řádky slouží skript run-console. Použití je snadné, stačí zadat do parametru jméno požadované molekuly (může, ale nemusí být v uvozovkách, pozor však na meta znaky jako „<“, „>“, „;“, ...). Program následně otevře vytvořený model (pokud nebyl název vyhodnocen jako chybný) v asociované aplikaci a skript se ukončí.

7.4. Konfigurace

Celý program (ve smyslu každá jeho část) je konfigurovatelný, od změny vzhledu generovaných modelů až po provedení rozhraní.

Databáze názvů

Databázi názvů molekul, které je program schopný interpretovat, lze snadno a do libovolné míry rozšiřovat. Nadměrným rozšiřováním se však významně prodlužuje doba potřebná na vytvoření derivačního stromu a s ní i celková doba běhu programu. Při rozšiřování je také potřeba postupovat opatrně a dávat pozor, aby v gramatice nevznikl konflikt. Pro tyto účely je dobré vytvořit si sadu názvů se správnými výsledky a po každé významnější změně spustit skript, který odhalí případnou neshodu. Další možností je použít připravený skript, který převede soubor gramatiky do formátu používaného programem bison, který po spuštění ve tvaru `bison -n -r solved gramatika.y` vytvoří nepotřebný soubor „gramatika.tab.c“ a soubor „gramatika.output“, kde je mimo jiné popsán stavový automat přijímající danou gramatiku, ale hlavně seznam všech konfliktů. To usnadňuje jejich hledání a navíc na rozdíl od testů zaručuje nalezení všech. Naopak vzniká opačný problém – nalezení „falešných“ konfliktů, které ve skutečnosti žádnými konflikty nejsou (bison totiž umí pracovat pouze s gramatikami ze třídy LALR(1), ale gramatika chenu je bezkontextová (BKJ), což je výrazná nadmnožina [viz obr. 12]).



Obr. 12: Hierarchie tříd gramatik [9]

Vlastní rozšiřování je snadné. Je-li přidán nový terminál, je potřeba ho zařadit do seznamu v souboru „terminaly.gra“ a to na správnou pozici. Každému terminálu

je přiřazeno číslo a to buď jeho zapsáním za něj odděleně sekvencí bílých znaků, nebo je to číslo o jedna větší než jaké bylo použito na předchozím řádku. Terminál musí být uveden v apostrofech, jinak je řádek považován za komentář (nesmí obsahovat mezeru), ale případné uvedené číslo je použito.

Správnou pozicí nového terminálu se myslí taková, kdy zůstanou zachovány příslušnosti k jednotlivým intervalům definovaným v souboru „konstanty.txt“. Ten vypadá tak, že je na každém řádku definován jeden interval. Definice má tvar dolní mez, libovolný znak, horní mez, sekvence bílých znaků (SBZ), pomlčka, SBZ, jméno intervalu, SBZ, popis bez mezer. Pokud je na místě pomlčky jiný znak, předpokládá se, že již následuje jen SBZ a řádek je ignorován.

Nejobtížnějším krokem rozšiřování gramatiky je přidání nového pravidla. To se provádí do souboru „gramatika.gra“. Jeho syntax je uvedena v Příloze 10.4.

Skripty

Spouštěcí skripty lze jednoduše modifikovat pro specifické potřeby uživatele pouhým odkomentováním/zakomentováním příslušného řádku. Bližší informace o možnostech jsou obsaženy přímo v komentářích jednotlivých skriptů (viz Příloha 10.3).

WWW rozhraní

Celý vzhled WWW stránky použité v tomto rozhraní lze změnit v souboru „index.htm“. Je pouze nutné zachovat několik následujících pravidel:

- Nejsou vkládány žádné soubory (obrázky, videa, skripty, styly, ...).
- Všechna místa, kde má být umístěno jméno zobrazené molekuly, jsou označena řetězcem `$NAZEV$`.
- Všechna místa, kde má být vložen výsledek zpracování názvu molekuly, jsou označena řetězcem `$TELO$`.
- Formulář pro odeslání dotazu ...
 - je odeslán metodou GET.
 - má atribut action nastaven na „/chenom“.
 - obsahuje jediný prvek (kromě tlačítka submit) – položku jménem „mol“, v níž je vyplněn název požadované molekuly.
- Formátování lze provést pomocí CSS, použity jsou dvě třídy:
 - error pro chybový výstup (element div),

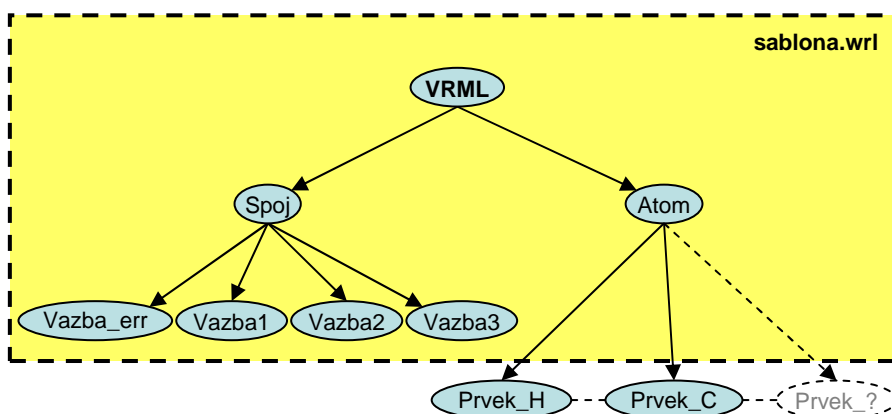
- výstup pro okno molekuly (element embed).

Vzhled modelu

Různé součásti vzhledu vytvořených modelů jsou definovány na různých místech. Pokud se má změna vzhledu projevit i v modelech, které již byly vytvořeny (při zapnutém cachování [viz Příloha 10.3]), je nutné buď změnu provést i v nich, nebo (což je obvykle jednodušší) smazat obsah složky outs a modely se vytvoří znovu při příštím požadavku na ně již se správným vzhledem.

Definice vlastností jednotlivých atomů (barva a poloměr) jsou umístěny v souboru „prvky.gra“. Jedná se o 5. (poloměr) a 7. (barva v HTML formátu) sloupec. Sloupce jsou odděleny pomocí SBZ. Pokud řádek nezačíná dvojicí znaků „>>“ (těsně předcházející prvnímu sloupci), je považován za komentář.

Všechny ostatní vlastnosti týkající se vzhledu jsou umístěny v souboru „sablon.wrl“. Jedná se o hlavičku výsledného VRML dokumentu – modelu. Jsou zde definovány případné Viewpointy, základní vzhled atomů a vazeb (jména těchto prototypů a jejich atributy musí být zachovány).



Obr. 13: Hierarchie prototypů v dokumentu modelu

8. Závěr

Cílem této práce bylo vytvořit program, který by pomohl především studentům středních škol při studiu názvosloví organické chemie. Tento vytyčený cíl byl snad programem CheNom naplněn. Přesto je možné mnohé věci vylepšit.

8.1. Možnosti rozšiřování

Určitá zlepšení je možné provést zpřesněním modelování, rozšířením o interpretaci názvů některých druhů sloučenin a stereochemických názvů nebo zjednodušením instalace. Některá rozšíření jsou již připravena. Výsledný VRML model molekuly by mohl být rozšířen o interaktivní zobrazování a skrývání vlastností jednotlivých jader (značka, náboj, seznam lokantů, které ho označují, ...). Pro tyto potřeby je seznam lokantů při sémantickém zpracování vytvářen, i když není dále využit.

Nepřesnosti v modelování spočívají v přílišném zjednodušení použitého modelu. Bylo by ji vhodné nahrať za některou chemiky používanou metodu, jako například hybridizace nebo VSEPR [10]. Systematické názvosloví heterocyklů a kondenzovaných polycyklů nebylo implementováno, protože má zcela odlišná pravidla tvoření názvů a často jsou tyto názvy chápány spíše jako triviální. Implementovat stereochemická pravidla nemá smysl, dokud je použit pro modelování současný model, který není schopen zaručit jejich dodržení.

Dalšího zpřesnění modelu by bylo dosaženo, kdyby byl program schopen při dalším dotazu na stejnou molekulu ji dále aproximovat (např. vždy po dobu dalších 10 sekund).

Pro větší pohodlí uživatele by bylo možné vytvořit instalátor / instalační skript, který by umožňoval nakonfigurovat program automaticky. Jednalo by se o umístění zástupce programu na plochu či do nabídky start, určení používaného interface, rozsahu cachování, atd.

8.2. Alternativní implementace

V průběhu tvorby programu byly zvažovány různé způsoby implementace jednotlivých modulů.

V modulu `Parser` přicházela v úvahu alternativa urychlující proces stavby derivačního stromu. Program by neprocházel stále dokola všechna pravidla gramatiky, ale místo toho použil zásobník (nebo frontu) na uložení těch pravidel,

kteřá by bylo možné použít. Seznam pravidel v zásobníku (frontě) nejprve tvoří všechna pravidla gramatiky. Při snaze o vytvoření derivačního stromu jsou nepoužitá pravidla odstraňována a naopak jsou přidávána pravidla použitá pravidla obsahující. Gramatika programu CheNom ovšem není natolik rozsáhlá, aby takto pracná implementace měla význam (doba výpočtu v modulu Model je totiž výrazně delší).

V modulu Server byla zvažována možnost vkládání různých souborů (obrázky, skripty, kaskádové styly, atd.) do WWW stránky rozhraní. Tato možnost byla z bezpečnostních důvodů zavrhnuta. Bylo by nutné ošetřit, aby uživatel nemohl program zneužít pro obejití nepovoleného přístupu k souborům.

8.3. Porovnání s existujícím SW

V současnosti je na trhu nepřehledné množství programů zaměřených na problematiku různých oblastí chemie. Co se týče interpretace českého názvu organických sloučenin, žádný takový software neexistuje.

Příkladem programu, který se také zabývá vizualizací molekul, je program ACD/Labs 8.0. Jeho předností je přesný model molekuly, ve kterém je možné také měřit vzdálenosti mezi atomy a vazebné úhly. Navíc umožňuje snadno přecházet mezi různými typy zobrazení (např. barvy a velikosti atomů). Naopak nevýhodami jsou možnost práce s názvem pouze ve směru od struktury, neschopnost programu v modelu graficky rozlišit násobnosti vazeb a jeho nedostupnost v češtině (a tedy ani programem generované názvy nejsou česky).

9. Použitá literatura

- [1] IUPAC, Commission on Nomenclature of Organic Chemistry: A Guide to IUPAC Nomenclature of Organic Compounds (Recommendations 1993), Blackwell Scientific publications, 1993
- [2] Kahovec, J. a kol.: Průvodce názvoslovím organických sloučenin podle IUPAC. Doporučení 1993, Academia, Praha 2000
- [3] Schejbalová, H., Stibor, I.: Úvod do studia organické a makromolekulární chemie, Vyd. TUL, Liberec 2004
- [4] Kolář, K. a kol.: Chemie /organická a biochemie/ II pro gymnázia, SPN, Praha 1997
- [5] Myšička, K. a kol.: Vzorce, modely a počítačová grafika ve výuce chemie, Gaudeamus, Hradec Králové 2006
- [6] Dalton, J.: A New System of Chemical Philosophy, John Weale, London 1842
- [7] Zrzavý, J.: VRML, Tvorba dokonalých WWW stránek, GRADA, Praha 1999
- [8] Berners-Lee T. a kol.: RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0, 1996
- [9] Yaghob J.: Principy překladačů, Lexikální a syntaktická analýza, [ftp://ulita.ms.mff.cuni.cz/pub/predn/pp/pp-03-lasxa.ppt], 2005
- [10] Mička Z. a kol.: Anorganická chemie I. Teoretická část, Karolinum, Praha 2003

10. Přílohy

10.1. Sémantika – jazyk

Konstanty

- **číslo:** #10 (nezáporné celé číslo)
- **náboj:** #2, #-2 (celé číslo)
- **řetězec:** ~text~
- **vazba:** - (jednoduchá), = (dvojná), % (trojná)

Parametry – reference

Syntaktický strom se prochází postfixově a v každém uzlu se vyhodnocuje sémantika na základě (již vyhodnocených) potomků. V právě vyhodnocovaném uzlu je vytvořeno pole obsahující výsledné hodnoty všech potomků a na nultou pozici se zapisuje vlastní výsledek. Tyto položky jsou dostupné pod svým indexem (tedy první syn odpovídá §1). Každý takový uzel je `struct`, na jehož položky se přistupuje standardní tečkovou notací. V každé instrukci má každý parametr nějakou defaultní položku, kterou není nutné (ale je možné) uvádět.

Položky jsou následující:

- **.int** – číslo
- **.str** – řetězec
- **.zdo** – seznam dvojic lokantů (pokud to není odkud-kam, ale pouze kde, tj. není to dvojice, pak se v definici neuvádí druhá položka a její hodnota je nastavena na prázdný řetězec)
 - **.z** – podpoložka odkud/kde (ze seznamu vrátí poslední hodnotu – RO!)
 - **.k** – podpoložka kam (ze seznamu vrátí poslední hodnotu – RO!)
- **.mol** – skelet – atomy a vazby mezi nimi, pokud je potřeba definovat jen některé jeho vlastnosti, vkládá se atom označený jako ~.~; jemu je pak možné nastavovat vlastnosti jako vaznost, náboj, ...; pokud je potřeba definovat pouze vazbu, pak volná vede z 1 do 0 a každá jiná z 1 do 1

Instrukce

Jazyk obsahuje několik tříd instrukcí, přístup se provádí přes standardní čtyřtečkovou (`class::method`) konstrukci. Každá instrukce se ukončuje středníkem.

Konvence:

- Nepovinné parametry jsou v hranaté závorce (tedy $XXX(a[, b])$) znamená buď $XXX(a, b)$, nebo $XXX(a)$.
- Jména parametrů jsou volena podle přípustného obsahu a to podle následujícího seznamu:
 - n – nezáporné celé číslo
 - m – reference na n (defaultní položka je `.int`)
 - N – n nebo rn
 - z – celé číslo
 - rz – reference na z (defaultní položka je `.int`)
 - Z – z nebo rz
 - s – řetězec
 - rs – reference na s (defaultní a jediná položka je `.str`)
 - S – s nebo rs
 - l – seznam (dvojic) lokantů
 - rl – reference na l (defaultní položka je `.zdo`)
 - L – l nebo rl
 - m – skelet -- zapisuje se následujícím způsobem:
 - definice se skládá ze dvou částí oddělených dvojtečkou
 - první část obsahuje definice atomů, tj. seznam definic atomů oddělených pomlčkou uzavřený do hranatých závorek
 - definice atomu vypadá takto:
(~znacka~, #nasobnost, seznam_jmen)
 - v seznam_jmen se alternativní názvy oddělují čárkou a celý je uzavřen do složených závorek
 - druhá část obsahuje definice vazeb, tj. seznam definic vazeb oddělených čárkou uzavřený do hranatých závorek
 - definice vazby vypadá následovně: zV_k , kde z a k jsou pořadová čísla vázaných atomů z první části definice (0 znamená volný konec) a V je typ vazby (tj. jeden ze symbolů `- = %`)
 - **Př.:** $[(\sim P \sim , \# 3 , \{ \sim 1 \sim \}) - (\sim N \sim , \# 5 , \{ \sim 2 \sim , \sim a \sim \})] : [1 \% 2 , 2 = 0]$
 - m – reference na m (defaultní položka je `.mol`)

- $M - m$ nebo rm

INT – číslo

- $NEW(N)$ – dosadí do $\$0.int$ hodnotu N

Př.: `INT::NEW(#3);INT::NEW($1.int);INT::NEW($1);`

- $ADD(rn, N)$ – položku rn zvětší o hodnotu N

Př.: `INT::ADD($0.int,#3);INT::ADD($0,$2.int);INT::ADD($1,$2);`

- $MUL(rn, N)$ – položku rn zvětší N -krát

Př.: `INT::MUL($0.int,#10);INT::MUL($0,$2.int);INT::MUL($1,$2);`

STR – řetězec

- $NEW(S)$ – dosadí do $\$0.str$ hodnotu S

Př.: `INT::NEW(~abc~);INT::NEW($1.str);INT::NEW($1);`

- $APP(rs, S)$ – za rs připojí řetězec S

Př.: `INT::ADD($0.str,~ab~);INT::ADD($0,$2.str);INT::ADD($1,$2);`

MOL – skelet

- $ZDO(rl, S_1[, S_2])$ – na konec rl připojí nově vytvořený lokant buď z S_1 (kde), nebo z S_1 (odkud) do S_2 (kam)

Př.: `MOL::ZDO($0.zdo,$1.str,~1~);MOL::ZDO($0,$1);`

- $LAP(rl, L)$ – za seznam lokantů rl připojí seznam lokantů L

Př.: `MOL::LAP($0.zdo,$1.zdo);MOL::LAP($1,$2);`

- $NEW(m)$ – dosadí do $\$0.mol$ hodnotu m

Př.:

`MOL::NEW([(~N~, #3, {~1~}) - (~N~, #5, {~2~, ~a~\})] : [1\%2, 2=0]);`

- $CPY(rm)$ – dosadí do $\$0.mol$ hodnotu rm

Př.: `MOL::CPY($1.mol);`

- $NWA(S)$ – dosadí do $\$0.mol$ atom (se standardní vazností), jehož značka je S a přiřadí mu lokant $\sim 1 \sim$

Př.: `MOL::NWA(~Na~);MOL::NWA($1.str);MOL::NWA($1);`

- $SUB(rm_1, rm_2, N[, L])$ – v rm_1 nahradí atomy na pozicích L atomem rm_2 (skelet obsahující právě jeden atom), zároveň otestuje, zda jich je N ; pokud L chybí, nahradí se všechny atomy v rm_1 , kterých je v tom případě očekáváno N

Př.: `MOL::SUB($5.mol,$4.mol,$3.int,$1.zdo);`

- $DLA(rm, N, L)$ – v rm odstraní atomy na pozicích L (pokud je atom připojen jen jednou vazbou – kromě vodíku –, tak je tato pouze zrušena, pokud dvěma stejné násobnosti, tak jsou tyto „sousedé“ propojeni), zároveň otestuje, zda jich je N

Př.: `MOL::DLA($4.mol,$3.int,$1.zdo);`

- $DLX(rm, S)$ – v rm odstraní atom na pozici S (všechny vazby jsou ponechány jako volné)

Př.: `MOL::DLX($4.mol,~1~);`

- $DLV(rm, N, L)$ – v rm odstraní všechny vazby dle L ($.z - .k$), zároveň otestuje, zda jich je N

Př.: `MOL::DLV($3.mol,$2.int,$1.zdo);MOL::DLV($3,$2,$1);`

- $DEC(rm, N, L)$ – v rm sníží násobnosti vazeb o 1 (případně je zruší) dle L , zároveň otestuje, zda jich je N (pokud je v L u všech položek $.zdo.k$ prázdný řetězec, pak se požaduje dvojnásobná velikost L oproti N a položky se berou po dvou)

Př.: `MOL::DEC($3.mol,$2.int,$1.zdo);MOL::DEC($3,$2,$1);`

- $INC(rm, N[, L])$ – opak DEC , tedy v rm zvýší násobnosti vazeb o 1 (případně je vytvoří) dle L , zároveň otestuje, zda jich je N (pokud je v L u všech položek $.zdo.k$ prázdný řetězec, pak se požaduje dvojnásobná velikost L oproti N a položky se berou po dvou); pokud je N nula, pak se spojuje první atom s posledním a L se neuvádí

Př.: `MOL::INC($3.mol,$2.int,$1.zdo);MOL::INC($3,#0);`

- $MVV(rm, S_1, S_2, S_3, S_4)$ – v rm přesune vazbu mezi S_1 a S_2 na vazbu mezi S_3 a S_4

Př.: `MOL::MVV($10.mol,$1.zdo.z,$3.zdo.z,$1.zdo.z,$5.zdo.z);`

- $ION(rm, Z)$ – do torza skeletu rm přidá (ve smyslu součtu) náboj Z

Př.: `MOL::ION($1.mol,#-2);`

- $APL(rm_1, rm_2, N[, L])$ – v rm_1 aplikuje vše platné z rm_2 (nesmí být atomy, jen vazba, náboj, arita, ...) na pozice L (kterých se otestuje, jestli je N); pokud rm_2 obsahuje i vazbu, která není volná (nevede do 0), pak má tato druhý konec buď dle L v $.zdo.k$, nebo v následujícím prvku po $.zdo.z$; pokud L chybí, tak se aplikuje dle N na všechny atomy v rm_1 (pokud jich je N), pokud je N o 1 menší, pak je vynechán poslední atom, pokud je $N = 1$, tak se aplikuje na první atom

Př.: `MOL::APL($1.mol,$6.mol,$5.int,$3.zdo);`

- `APP(rm1, rm2, N, L)` – za atomy rm_1 (do rm_1) připojí rm_2 a spojí první atom rm_2 s atomem dle L (kde), toto provede pro všechny položky L (otestuje, zda jich je N)

Př.: `MOL::APP($5.mol,$4.mol,$3.int,$1.zdo);`

- `CON(rm1, rm2, N[, L])` – za atomy rm_1 (do rm_1) připojí rm_2 a spojí všechny volné vazby rm_2 s atomem dle L , toto provede pro všechny položky L (otestuje, zda jich je N), pokud L není definován, pak se připojení provede N -krát na atom $\sim 1 \sim$

Př.: `MOL::CON($5,$4,$3,$1);`

- `LIN(rm1, rm2, N)` – vezme rm_2 , dá ji N -krát za sebe (spojí vždy první atom nové kopie s posledním atomem předchozí kopie), s tím, že v každé kopii zruší všechna jména, výsledek uloží do rm_1 a očísluje (od jedničky)

Př.: `MOL::LIN($0.mol,$0.mol,$1.mol);`

- `CHN(rm1, rm2, N[, L])` – vezme rm_2 , dá ji N -krát za sebe, s tím, že v každé kopii mají všechny lokanty přidánu čárku (oproti předchozí kopii), a poté spojí dvojice dle L (pokud je uvedeno, jinak atomy se jménem 1 a počtem čárek lišícím se o jedna, tj. 1-1', 1'-1", ...) jednoduchou vazbou

Př.: `MOL::CHN($0.mol,$2.mol,$1.int);`

- `ADD(rm1, rm2, N)` – k torzu skeletu rm_1 , přidá N -krát torzo skeletu rm_2 (tj. sečtou se náboje a sloučí se seznamy vazeb)

Př.: `MOL::ADD($0.mol,$2.mol,#1);`

- `FIN(rm)` – doplní každému atomu vodíky tak, aby mu nic nechybělo

$$\left(\text{vaznost} = \left(\sum_{\text{vazby}} \text{nasobnost} \right) + |\text{naboj}| \right), \text{ pokud toto splnit nelze (atom již má více}$$

vazeb, než je jeho vaznost), je vyvolána chyba

Př.: `MOL::FIN($1.mol);`

10.2. index.htm

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type"
      content="text/html; charset=windows-1250"
/>
<title>CheNom - $NAZEVŠ</title>
<style>
  /* popis formátování */
  ...
</style>

<script>
  /* javascript obsluhující tlačítka formuláře */
  ...
</script>
</head>

<body>
<!-- tlačítka na ukončení serveru -->
<div class="konec">
  <a href="/STOP">KONEC</a>
</div>

<!-- formulář pro zadání dotazu -->
<form action="/chenom"          method="GET"
      enctype="multipart/form-data" name="formular"
>
  <input type="text" name="mol" id="vstup" />
  <input type="submit" value="Zobrazit" id="odeslat" />
  <!-- tlačítka pro formátování a vkládání spec. znaků -->
  ...
</form>

<hr />

<h1>$NAZEVŠ</h1>
$TELOŠ

</body>
</html>
```

10.3. Skripty

START.bat

```
@echo off
cd Complet\
@echo on
start server 80
start firefox "http://127.0.0.1/chenom"
```

START.sh

```
#!/bin/sh
cd Complet/
./server 8080&
firefox "http://127.0.0.1:8080/chenom"
```

run.bat

```
rem @echo off
set jmeno=%1
echo %jmeno% >input
set vystup="outs\%jmeno:~1,-1%.wrl"
if EXIST %vystup% goto cached

rem -----
rem necachovat modely:
rem del /F /Q "outs\*.wrl"
rem -----

chenom.exe DATA\cp1250\settings.ini -e >%vystup% 2>error.log < input
if ERRORLEVEL 1 (
    del input

rem -----
rem podrobny chybovy vystup:
echo ===== >> error.log
type %vystup% >> error.log
rem -----

    del %vystup%
    rem pause
    @echo on
    exit 3
)
:cached
del input
@echo on
```

run.sh

```
#!/bin/sh
vstup=input
vystup="outs/$@.wrl"

echo $@ >$vstup

# -----
# necachovat modely:
rm -f outs/*.wrl
```

```

# -----
if [ ! -e $vystup ]; then
  if ! ( \
    ./chenom DATA/utf8/settings.ini -e <$vstup >$vstup 2>error.log \
  ); then
    # -----
    # podrobny chybovy vystup:
    echo ===== >> error.log
    cat $vstup >> error.log
    # -----
    rm -f $vstup
    rm -f $vystup
    exit 3
  fi
fi
rm -f $vstup

```

run-console.bat

```

@echo off
set jmeno=%1
echo %jmeno% >input
set vystup="outs\%jmeno:~1,-1%.wrl"
if EXIST %vystup% goto cached

rem -----
rem necachovat modely:
rem del /F /Q "outs\*.wrl"
rem -----

chenom.exe DATA\cp1250\settings.ini >%vystup% < input
set chyba=%ERRORLEVEL%
if ERRORLEVEL 1 (
  del input

  rem -----
  rem podrobny chybovy vystup:
  echo ===== >> error.log
  type %vystup% >> error.log
  rem -----

  del %vystup%
  pause
  @echo on
  exit /B %chyba%
)
:cached

del input
rem -----
rem zde napiste prikaz, který otevře model ve vašem prohlizeci
rem cesta k souboru je %vystup%
rem explorer %vystup%
%vystup%
rem -----
rem necachovat (druha varianta):
rem del %vystup%
rem -----
@echo on

```

run-console.sh

```
#!/bin/sh
echo $@ >input
vstup=input
vystup="outs/$@.wrl"

# -----
# necachovat modely:
rm -f outs/*.wrl
# -----

if [ ! -e $vystup ]; then
  if ! ( \
    ./chenom DATA/utf8/settings.ini <$vstup >$vystup 2>error.log \
  ); then
    # -----
    # podrobny chybovy vystup:
    echo ===== >> error.log
    cat $vystup >> error.log
    # -----
    rm -f $vstup
    rm -f $vystup
    exit 3
  fi
fi
rm -f $vstup

# zde napiste prikaz, který otevře model ve vašem prohlizeci
# cesta k souboru je $vystup
```

10.4. Databázové soubory

settings.ini

```
[Soubory]
Konstanty = konstanty.txt
Terminaly = terminaly.gra
Gramatika = gramatika.gra
Prvky = prvky.gra
Vystup = sablona.wrl
```

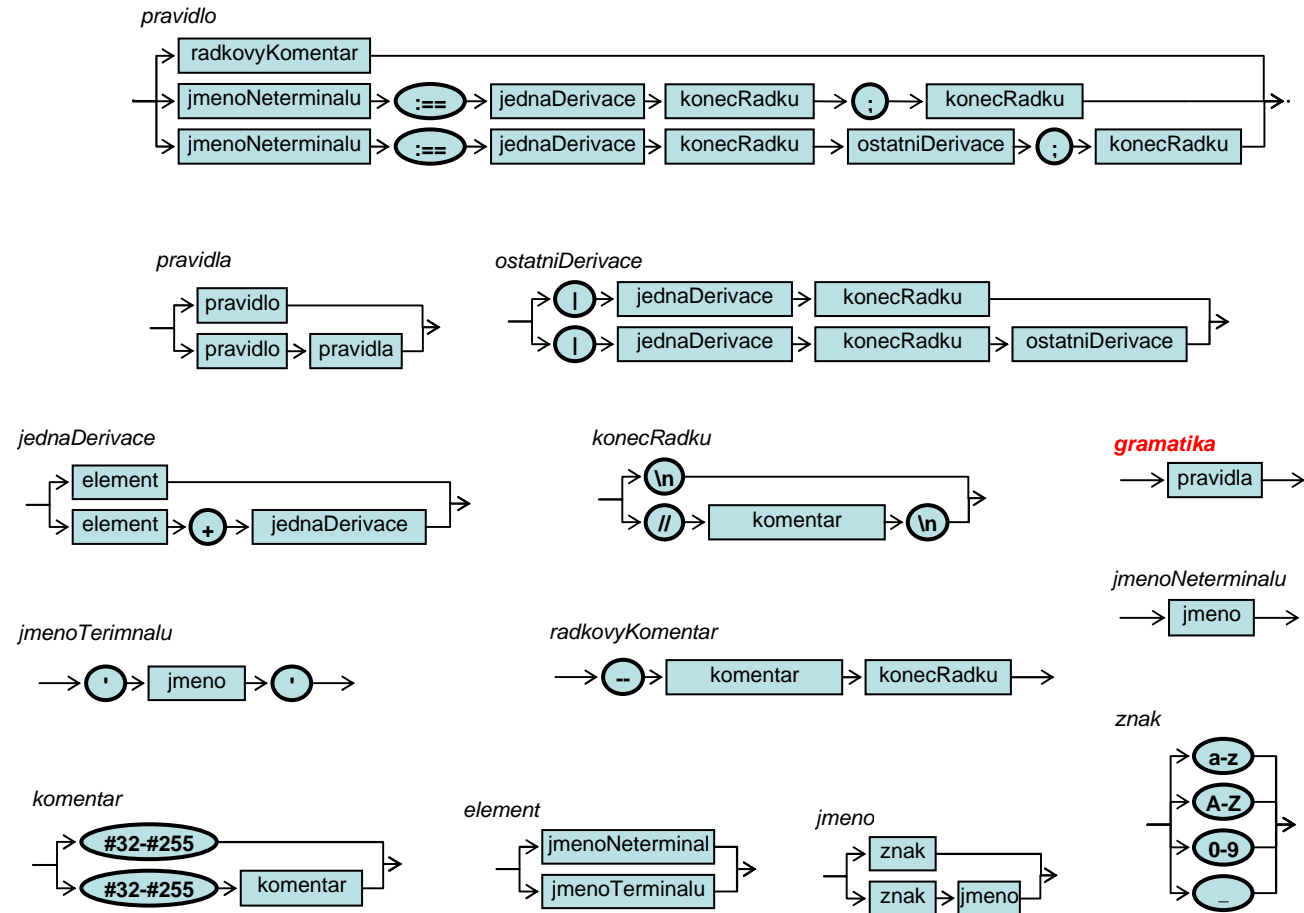
Jméno souboru nesmí obsahovat mezeru, pokud je použita relativní cesta, nesmí být z bezpečnostních důvodů použit odkaz do nadřazené složky. Cesta je vztažena ke složce, v níž se nachází tento soubor. Tedy pokud je absolutní cesta k tomuto souboru `C:\chenom\DATA\settings.ini`, ukazuje cesta `v2\prvky` na soubor `C:\chenom\DATA\v2\prvky`.

konstanty.txt

```
0000-0006 - Separator (.,,,:,,;,-,->)
0007-0007 - Carka (')
0008-0019 <
0020-0029 - Zavoroka ((,],[,},{,})
0030-0039 - Operace (de,nor,anhydro,cyklo,seko,abeo,retro)
0040-0049 <
0050-0059 - Formatovani (<i>,<sub>,...)
0060-0099 <
0100-0199 - CharakteristickaSkupina (yl,an,en,yn,...)
0200-0299 - Stereodeskripty (cis,trans,meso,...)
0300-0499 <
0500-0529 - AnorgPripona (ný,natý,...)
0530-0999 <
1000-1009 - Cislice (0123456789)
1010-1099 <
1100-1199 - Alfabeto (αβ...)
1200-1229 - Latinka (abcd...)
1230-1249 - Samohlaska (aáeéěííóóúúýý)
1250-1399 <
1400-1433 - ReckePocty (#lokantů--přidat_násobnost_lokantů,..)
1434-1499 - LatinskePocty (počty_opakování_struktur,...)
1500-1999 <
2000-2199 - ZnackaPrvku (H,He,...)
2200-2999 - JmenoPrvku (oxa,sila,...)
3000-3999 - TypSlouceniny (aren,oxokyselina,keton,...)
4000-4999 - TrivKoren (bora,oxida,joda,metha,...)
5000-9999 - TrivNazvy (acetylen,ethan,...)
```

terminaly.gra

```
===== 0000
'-'
';'
':'
','
'.'
'-'>'
'''
----- 0020
'('
')'
 '['
']'
 '{'
}'
----- 0030
'de'
'nor'
'anhydro'
'cyklo'
'seko'
'abeo'
'retro'
'per'
----- 0050
'&nbsp;';'
'<sup>'
'</sup>'
'<sub>'
'</sub>'
'<i>'
'</i>'
===== 0100
'yl'
'yliden'
'ylidyn'
'ium'
'ylum'
'id'
'it'
'át'
'an'
'en'
'yn'
'al'
'ol'
'on'
'lat'
'at'
'oyl'
'imin'
'amoyl'
'oát'
'os'
.
.
.
```



prvky.gra

--Zn	Z	arita	hybridizace	poloměr	hmotnost	barva
>>H	1	1	s1	1	1.E-3	#0000ff
--He	2	-1	-	1	1.E-3	#ffffff
--Li	3	-1	-	1	1.E-3	#ffffff
--Be	4	-1	-	1	1.E-3	#ffffff
--B	5	3	-	1	1.E-3	#ffffff
>>C	6	4	slp3	1	1.E-2	#ff0000
>>N	7	3	slp2	1	1.E-3	#00ff00
>>N	7	5	slp3d1	1	1.E-3	#00ff00
>>O	8	2	slp1	1	1.E-3	#ffffff
>>F	9	1	s1	1	1.E-3	#00ffff
--Ne	10	-1	-	1	1.E-3	#ffffff
.
.
.

sablona.wrl

```

#VRML V2.0 utf8
NavigationInfo { type "EXAMINE" }
##### pohledy #####
Viewpoint {
  position 0 0 50
  description "above"
}
# ...

##### atom - vzor #####
PROTO Atom[
  field SFCOLOR dCol 1.0 1.0 1.0
  field SFFloat fRad 1.
]{
  Transform{
    children[
      Shape{
        geometry Sphere {
          radius .5          #polomer vnitřni části splyvajici s vazbami
        }
        appearance
        Appearance{
          material Material {
            diffuseColor IS dCol
            transparency 0.   #vnitřni část - viditelnost
          }
        }
      }
      Shape{
        geometry Sphere {
          radius IS fRad
        }
        appearance
        Appearance{
          material Material {
            diffuseColor IS dCol
            transparency .5   #vnější část - viditelnost
          }
        }
      }
    ]
  }
}

##### vazby - vzory #####
PROTO Spoj[
  field SFCOLOR dCol 1.0 1.0 1.0
  field SFFloat fLen 2.
  field SFVec3f fPos 0 1 0. #0 fLen/2 0

```

```

    field SFFloat fRad .5
  ]{
    Transform{
      translation IS fPos
      children[
        Shape{
          geometry Cylinder {
            height IS fLen
            radius IS fRad
          }
          appearance
          Appearance{
            material Material {
              diffuseColor IS dCol
              transparency 0
            }
          }
        }
      ]
    }
  }
}

PROTO Vazba_err [
  field SFFloat fDist 2.
  field SFVec3f fPoz 0 1 0.
] {
  Spoj {
    dCol 1.0 0.0 0.0
    fLen IS fDist
    fPos IS fPoz
    fRad .25
  }
}

PROTO Vazba1 [
  field SFFloat fDist 2.
  field SFVec3f fPoz 0 1 0.
] {
  Spoj {
    dCol 1.0 1.0 1.0
    fLen IS fDist
    fPos IS fPoz
  }
}

PROTO Vazba2 [
  field SFFloat fDist 2.
  field SFVec3f fPoz 0 1 0.
] {
  Transform{
    children[
      Transform{
        translation .16666666 0 0
        children Spoj {
          dCol 0 1 1
          fRad .33333333
          fLen IS fDist
          fPos IS fPoz
        }
      }
      Transform{
        translation -.16666666 0 0
        children Spoj {
          dCol 0 1 1
          fRad .33333333
          fLen IS fDist
          fPos IS fPoz
        }
      }
    ]
  }
}

```

```
# PROTO Vazba3 - obdobne
```

```
#####
```