International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.4, October 2011

ANALYSIS OF NAMES OF ORGANIC CHEMICAL COMPOUNDS BY USING PARSER COMBINATORS AND THE GENERATIVE LEXICON THEORY

Márcio de Souza Dias[1], Rita Maria Silva Julia[2] and Eduardo Costa Pereira[3]

[1]Department of Computer Science, Federal University of Goiás, Catalão-Goiás, Brazil
marcio.dias@catalao.ufg.br
[2]College of Computation, Federal University of Uberlândia, Uberlândia – Minas Gerais, Brazil
rita@ufu.br
[3]FEELT, Federal University of Uberlândia, Uberlândia - Minas Gerais, Brazil
costa@ufu.br

ABSTRACT

This work proposes OCLAS (Organic Chemistry Language Ambiguity Solver), an automatic system to analyze syntactically and semantically Organic Chemistry compound names and to generate the pictures of their chemical structures. If both parses detect that the input name corresponds to a theoretically possible organic chemical compound, the system generates its molecular structure picture, whether or not the name respects the current official nomenclature. This capacity of treating even names which, in spite of do not respect the constraints of the official nomenclatures, correspond to theoretically possible organic compound, represents an advance of OCLAS compared to other existing systems. OCLAS counts on the following tools: Generative Lexicon Theory (GLT), Parser Combinators and the Language Clean and an extension of the Xymtec package of Latex. The implemented system represents a helpful and friendly utilitarian as an automatic Organic Chemistry instructor.

KEYWORDS

Automatic Tutors for Organic Chemistry Nomenclature, Lexical Ambiguity, Computational Linguistics, Generative Lexicon Theory and Parser Combinators.

1. INTRODUCTION

All languages have ambiguities. In fact, some ambiguities are equivalent to paradoxes in logic systems. However, there are a few languages that come very close to eliminate all ambiguities due to syntaxes, morphology, and meaning (direct semantics). These languages are either artificial, or evolved in academic environment. The authors of the present paper use Parser Combinators and semantic tags to eliminate ambiguities in the Organic Chemistry language. The comprehension of the structures of the chemical compounds is fundamental in the context of the Chemistry, principally considering the relevance of domains such as provision and pharmaceutical industry in the modern world. Thus, the nomenclature adopted to name the chemical compounds must be seriously treated in order to allow coherent representations for them. The IUPAC (International Union of Pure and Applied Chemistry) is an organism responsible for establishing an official nomenclature for the chemical compounds [1].

In order to be able to treat chemical compound names, an automatic system must comprise appropriate terminologies and sets of syntactic and semantic rules to combine terms of the chemistry language such as to produce well formed sentences, that is, names for the chemical compounds which satisfy the constraints of the IUPAC nomenclature. To cope with this task,

the system must deal with the problem of the internal structure of chemical words and must examine the terms which are used to form simple words, complex words, or bigger grammatical units, so-called multi-word expressions or well formed sentences [2]. Further, the system must solve problems of lexical ambiguity. A lexical item is ambiguous when it has two or more possible readings, usually with distinct interpretation in a given context. The methods provided by the natural language processing (NLP) to treat sentences of the human languages can be successfully used as tool in several other related domains, such as: database interface [3], text mining [4] and technical language processing [2]. Particularly in this paper, they are used to deal with the task of detecting whether a name proposed to represent a chemical compound is coherent with the IUPAC nomenclature. Thus, one can count on syntactic and semantic parsers [5] [6] to analyse names of chemical compounds. The system OCLAS proposed here receives an organic compound name, analyses it syntactically and semantically and, whenever it represents a theoretically possible organic chemical compound, it generates a visual output for its chemical structure. An advance that the system shows in relation to other ones which also deal with chemical nomenclature consists on being able to analyse compound names that, in despite of do not respect the IUPAC nomenclature constraints, represent theoretically possible organic compounds. To succeed in this task, OCLAS must treat the problem of lexical ambiguity in the chemical language. The semantic and syntactic analysis of the chemical names are guided by the types of the terms which they are composed of. That is why the following suitable tools were used in the implementation of the system, obtaining very good results: Generative Lexicon Theory (GLT), Parser Combinators and the Functional Language Clean.
Another contribution of OCLAS is to extend the Xymtex package such as to use it as a tool for successfully generating clear and didactical pictures of the chemical structures. This paper presents OCLAS, compares it to other related works and shows that it can be a helpful utilitarian as an automatic instructor of Organic Chemistry Nomenclature. Preliminarily and for testing the proposed approach, the authors of OCLAS treated the alkanes, alkenes, alkynes, alkadyenes, alcohols and aldehydes. Throughout this paper, the following Definitions must be considered:

- Correct names: names that represent theoretically possible chemical compounds written according to the IUPAC Official Nomenclature Rules (IUPAC-ONR);
- Inadequate names: names that, in despite of do not respect the IUPAC-ONR, represent theoretically possible chemical compounds, that is, they satisfy all the chemical constraints related to the organic compounds (such as bonds, kind of atoms which can appear in the compounds etc);
- Incorrect Names: names that do not correspond to theoretically possible chemical compounds.

2. THEORETICAL BACKGROUND

2.1. Principles of Organic Chemistry

The organic chemistry is the branch of chemistry that studies the carbon based chemical compounds.

Carbon (C) is the main element that appears in the formation of organic compounds. The atoms that most frequently appear in these compounds, further than the carbon, are: hydrogen (H), oxygen (O), nitrogen (N), the halogens, the sulphur (S) and phosphorus (P). In chemistry, valency is a measure of the number of possible chemical bonds associated to the atoms of a given element [7]. Particularly, the carbon is a tetravalent element, as shown in Figure 1. A hydrocarbon is a chemical compound composed just of C and H.

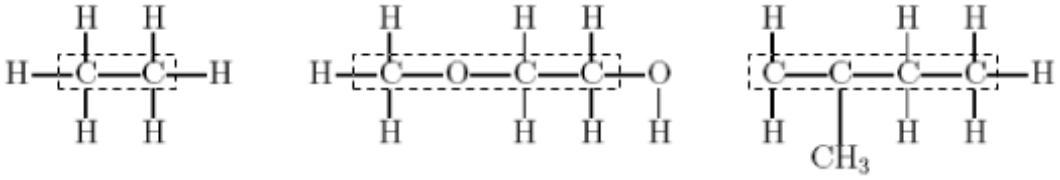

Figure 1. Types of carbon chains

2.2. Nomenclature (IUPAC System)

The IUPAC nomenclature system is a set of syntactical, lexical and pragmatic rules that organic chemists use to treat the chemical nomenclature. From these rules, given a structural formula, one is able to write a unique name corresponding to every distinct compound. In the same way, given an IUPAC name, one is able to write a structural formula. An IUPAC name has three essential features [8]: a root that indicates the longest continuous carbon atoms found in the molecular structure; a suffix and, possibly, other element(s) which designate functional groups that may appear in the compound; and, finally, names of substituent groups distinct from hydrogen that complete the molecular structure.

In the following subsections will show the nomenclature of some of the main organic functions treated by OCLAS.

2.2.1. Alkane hydrocarbons

The IUPAC rules establish the following steps to name the alkanes (hydrocarbons having only simple bonds) [9]:

- Select as main chain the longest continuous carbon chain (*Main Chain Rule*). For example, the carbon chain of Figure 2 represents the main chain of the compound *3-methyl-hexane*;

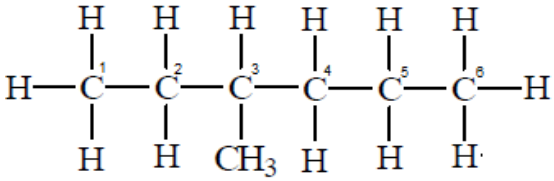

Figure 2. 3-methyl-hexane

- knowing that a substituent is an atom or group of atoms that replaces a hydrogen atom on the main chain of a hydrocarbon [10], number the carbons in the chain from either end, such that the substituents are given the lowest numbers possible (*Lowest Numbers Rule*) (see figure 3). These numbers are called “locants”.
- The substituents are assigned the number of the carbon to which they are attached. In Figure 2, the substituent CH_3 is assigned the number 3.
- The name of the compound is now composed of the name of the main chain preceded by the name and the number of the substituents, arranged in alphabetic order. For the same example, the name is thus 3-methylhexane.
- If a substituent occurs more than once in the molecule, the prefixes, “di-“, “tri-“, “tetra-“ etc., are used to indicate how many times it occurs.

- If a substituent occurs twice on the same carbon, the number of the substituent is repeated.

2.2.2. Alkenes hydrocarbons

Hydrocarbons having at least one carbon-carbon double bond (C=C).

- Select as the main chain the longest continuous carbon chain that contains the carbon double bond (C=C). Replace "ane" with "ene" (see Figure 3).
- Number this chain from the end that will give the C atom starting the double bond the lowest number. Prefix the name with this number.
- Treat substituent as in alkanes.
- Dienes contain two double bonds, trienes have three, etc.

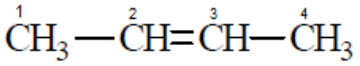

Figure 3. 2-butene

2.2.3. Alkynes hydrocarbon

The nomenclature of alkynes is similar to that of alkanes, but for the fact that the main chain must include the triple bond and be numbered in such a way that the functional group has the lowest position number. Further, one must substitute "yne" for "ane" and assign a position number to the first carbon of the triple bond (see Figure 4).

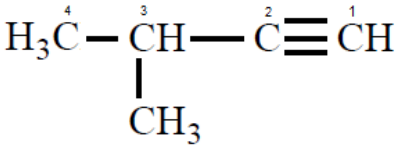

Figure 4. 3-methyl-1-butyne

2.3. TLG - The Generative Lexicon

This subsection presents a brief overview of the qualy structures used in the TLG to define a lexical item. Mores details can be found in [11].

Roles: the TLG uses the roles to characterize a lexical item. The principal roles in the context of OCLAS are:

- Formal: it establishes some characteristics that distinguish an object within a larger domain (Orientation, magnitude, shape, dimensionality, color, position etc).
- Telic: it describes the purpose of a lexical item.
- Agentive: It indicates whether and how a lexical item can be applied to another in order to generate a third lexical item. For instance, the *agentive* of *pent* is *assembly_function*, that is, a function that applies *pent* to another lexical item.
- **Qualia Structure:** a qualia structure used by the TLG uses to define a lexical item may be composed of:
- EVENSTR: it is used to define a lexical item that may be applied to another one, that is, a lexical item whose type is a *process*.

- ARGSTR: The argument structure (ARGSTR) of a lexical item L which is a *process* exhibits two kinds of arguments: first, the arguments that were involved in the earlier applications which originated L; second, the arguments (and their respective types) to which L can be applied in order to generate another lexical item.
- QUALIA: the field QUALIA of the *structure qualia of a lexical item L has as objective to characterize L, through the definition of its roles.*

2.4. Parser Combinators

The parser combinators are operators used to manipulate the parsers. The principal combinators used in OCLAS are (more details can be seen in [12] and [13]):

- <&>: it is called *sequential* operator. The expression P1 <&> P2, where P1 and P2 are parsers (and P2 is a lambda abstraction), is executed in the following way: P1 is applied to an input list L of lexical items. The combinator <&> passes to P2 the result and the difference list [14] obtained from this application (the result is passed as an argument to the parameter of P2).
- <&: This operator works in the same way as the operator <&>, except for one aspect: differently from the later, it discards the lexical item selected by P2.
- <@: it is a transformer combinator. Apart from the operators <&> and <!>, that combine parsers, transformer combinators modify existing parsers. The operator <@ applies a given function to the result parse trees of a given parser. Given a parser *p* and a function *f*, in *p* <@ *f*, the operator <@ returns a parser that does the same as *p*, but, in addition, applies *f* to the resulting parse tree obtained by the evaluation of *p*. In practice, the <@ operator is used to build a certain value during parsing. Put more generally: the operator <@ adds a *semantic function* to the parsers [13][14].
- <!>: It is an operator used for alternative composition (that is, it represents *choice*).

2.5. Least Upper Bound

Definition: Let S be a set with a partial order ≤. Then $a \in S$ is the *Least Upper Bound* of a subset X of S (denoted by LUB(X)) if $x \leq a$, for all $x \leq X$ [15].

Definition: Let S be a set with a partial order ≤. Then $a \in S$ is the *Least Upper Bound* of a subset X of S if *a* is an upper bound of X and, for all upper bounds *a'* of X, we have $a \leq a'$.

2.6. Xymtec

Xymtec is a demarcation package that combines files of style Latex developed to draw a wide variety of chemical structural formulas [16]. The commands of Xymtex have a group of systematic arguments to specify substitutions and their positions, internal cycles, double connection, triple connection and connection pattern (simple). In some cases, they have an additional argument to specify heteroatoms in the heterocycle vertexes. As a result of this systematic characteristic, Xymtec indeed works as a practical tool inside the independent device TEX [17].

2.6.1. Characteristics of Xymtec

Some of the main characteristics of Xymtec are resumed below:

- Xymtec only requests the illustration environment of Latex what assures portability;

- Structural formulae drawn in Xymtec present high level of quality due to the Latex sources.

2.6.2. The commands *Xymtec*

This subsection resumes the most important Xymtec commands used in the present work. The command **\tetrahedral**, by receiving as arguments the characters shown in table 1, draws a tetrahedral unit corresponding to a carbon atom. More details can be seen in [16].

Table 1. Arguments of The **\tetrahedral** Command

Character	Generated Structures
n or nS	inserts a simple bond in the n-th valency
nD	inserts a double bond in the n-th valency
nT	inserts a triple bond in the n-th valency
nA	simple bond alpha in the n-th valency
nB	simple bond beta in the n-th valency

For example, the commands below produces the pictures illustrated in figure 5:

```
\tetrahedral{0==C;1==H;2==OH;3==H;4==H}\qquad
\tetrahedral{0== C;1D==O;2==Cl;4==Cl}\qquad
\tetrahedral[{}{0+}]{0==N;1==H;2==CH$_{3}$;3==H; 4==H}\qquad
\tetrahedral{0==C;1==H;2==H;3==H;4==H}
```

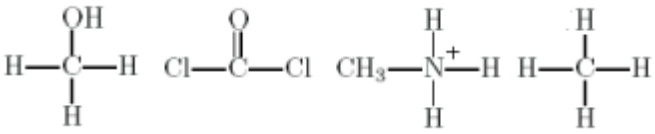

Figure 5. Drawings generated by the command **\tetrahedral**

The macros **\rtrigonal** and **\ltrigonal** are used to draw right-handed and left-handed trigonal units, respectively. For instance, the commands below output the pictures shown in Figure 6:

```
\rtrigonal{0==C;1==H;2D==O;3==H}\qquad
\ltrigonal{0==C;1==H;2D==O;3==H}
```

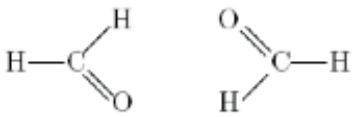

Figure 6. Drawings generated by the command \ltrigonal

Note that the original Xymtec resources allow drawing the chemical structure corresponding to just one atom with its bonds (it is not able to represent side-chains). In section 4, the authors

present the extensions introduced in the Xymtec package in order to enable OCLAS to represent complete chemical structure pictures.

3. RELATED WORKS

This section introduces some systems that treat the lexical ambiguity in Chemistry Languages. In [18], Frost, Hafiz and Callaghan propose a set of parser combinators that can be efficiently used for treating ambiguous grammar (even left-recursive grammars). Their algorithm combines memoization (a technique for storing the values of a function instead of re-computing them each time the function is called) with existing techniques for dealing with left recursion. It is relevant to point out that in Frost´s system the NL linguistic ambiguity is treated by combining the lexical items of the sentences under analysis in all possible ways. Subsection 4 shows that OCLAS, in order to be able to treat certain cases of ambiguity in the Organic Chemistry Language, must behave in a different may: it has to try to generate, from the original set of lexical items, a new one which corresponds to the ambiguous input name and which enables the system to produce the correct chemical structure that represents the input name.

A more recent work in the area of Computational Linguistics applied to the Organic Chemistry was developed by Stefanie Anstein and Gerhard Kremer in 2005 [2]. They proposed a system for analysis of chemical terminology that is able to deal with systematic, trivial and semi-systematic chemical terms of organic substances, with chemical class names and with semi-systematic class names. The analysis is performed by a morph-semantic grammar developed according to IUPAC nomenclature. It yields an intermediate semantic representation that describes the information encoded in a name. The system outputs SMILE strings corresponding to the analysed terms and an appropriate classification for them. A smile string is a structural notation of a molecule that sequentially lists the main chain elements with their properties and branches. In the Anstein-Kremer's system, the basis for the generation of the SMILES strings is the semantic representation of the compound name, which describes the operations to be applied to nested semantic structures. The SMILE strings can be used to map the analysed term into its molecular structure. Systematic names are those expressed in terms of the official nomenclature, whereas trivial terms are usual designations for them. Semi-systematic names are a combination of trivial or class names and systematic names. Underspecification describes the fact that a certain linguistic entity to be definite and unambiguous is missing. The characteristics of the entity are thus not fully specified. Usually, the missing information can be deduced from the linguistic or other context (resolvable underspecification). In other cases, it is not possible (underspecification can not be resolved). For example, for the underspecified name *ethene (C=C)*, the position of the double bond is clear even though not indicated because there is only one posibility, whereas the underspecified name *butene* can be used to refer to either (in Smile notation) *C=CCC* or *CC=CC*. The ability to cope with underspecification and class names distinguishes Anstein-Kremer's system from other existing ones. Their system also allows that nomenclature-based synonyms are identified by either matching their semantic representation or their SMILES strings(2-pentulose and pent-2-ulose yield the same output). Anstein-Kremer's rules are only formulated for the purpose of analysis: their system is not meant for name generation from structures even though that would be theoretically possible. For testing their approach, Anstein-Kremer treated the carbohydrates (or *sugars*). Finally, Anstein-Kremer's system is able to analyse only certain types of embedded compound names, i.e., names that represent complete compounds themselves but that are part of other compound names (for example, all the alkanes, alkenes and alkynes are represented by embedded names). As shown in section 4, OCLAS extends the Anstein-Kremer's work, once it is capable of treating the inadequate names. Section 4 also shows that the use of the Xymtec package in OCLAS provides

a much more expressive representation for the figures of the chemical compounds than the smile structures outputed in Anstein-Kremer's system.

Abe's system treats the nomenclature of acyclic chemical compounds [19]. It receives as input a certain structural formula and outputs the official name (according to the IUPAC conventions)corresponding to this input. OCLAS and Abe's system work in different ways, since the first one has as input an organic chemical compound name and the second one a formula that corresponds to the chemical structure of a chemical compound. Further, distinctly from OCLAS, Abe's system is not able to treat ambiguous input (that is, whereas OCLAS is able to treat inadequate names, Abe's system just treats correct input formulae).

Raymond's software [20] helps beginner students of Organic Chemistry to learn how to use the IUPAC rules. The system receives as input a chemical compound name (alkanes, alkenes, alkynes, and halides) and, according to the IUPAC rules, outputs the main chain, the radicals, the suffix multipliers, their locations etc. Another functionality of Raymond's system is to allow that the user names the input structural formula. In this case, the system checks whether the proposed name is correct or not - if it is not, the system just informs the user that he has not correctly named the input structural formula, without proposing an alternative possible correct name for input names which are inadequate (distinctly from the behaviour of OCLAS).

4. THE SYSTEM OCLAS

The System OCLAS is a didactic tool for analysing names of Organic Chemistry compounds and for generating their corresponding chemical structure pictures.
Differently from other systems that process the chemical language (see section 3), OCLAS is also capable of analysing inadequate names. In this case, the system is able to output the correct chemical structure picture of the real compound that corresponds to the input name. To deal with this additional task, it must be capable of attacking and solving some kind of linguistic ambiguity problems, as shown below.

4.1. Ambiguity in Chemical Names

Incorrect or inadequate names generally appear when someone who does not keep down the IUPAC rules tries to name organic compounds. Whenever OCLAS detects that an input name does not respect the official rules, it tries to adjust it such as to generate a correct name from it. If the input name represents a theoretically possible chemical compound, OCLAS succeeds and outputs its corresponding chemical structure picture. Otherwise, the system warns the user that the name proposed does not represent a theoretically possible compound. Examples 1 and 2 below show situations in which inadequate names are submitted to OCLAS. In the examples, in the first phase of the analysis the system finds out that the input name violates at least one of the IUPAC rules; in the second phase, OCLAS succeeds in the task of adjusting the input names and infers the real chemical structures corresponding to them (it means that the input name is inadequate). This adjustment consists on determining the appropriate main chain and side-chains that can be retrieved from the lexical items that composes the input name. It is important to point out that this adjustment only succeeds when these lexical items (i.e., bonds, insaturations, number of carbon atoms, function identifier etc.) can be recombined in an alternative way that maps into a real chemical compound and into a correct name.

Example 1 - Analysis of *2-3-diethyl-4-4-dimethyl-3-pentanol*: First, OCLAS detects that this name does not respect the IUPAC rules, since it violates the Main Chain Rule (as shown in figure 8, which highlights the incorrect main chain that corresponds to the proposed name). Next, by taking into account the lexical items of the input name (that is, two radicals *ethyl* (with locants 2 and 3), two radicals *methyl* (both with locants 4), the carbon chain *pent*, the alkane

identifier *ane* and the alcohol identifier *ol* (with locant 3), OCLAS finds out that a correct compound, with an appropriate main chain and appropriate side chains can be retrieved from them.

This correct compound is *3-ethyl-2-2-4-trimethyl-3-hexanol*, whose molecular structure OCLAS outputs in figure 7.

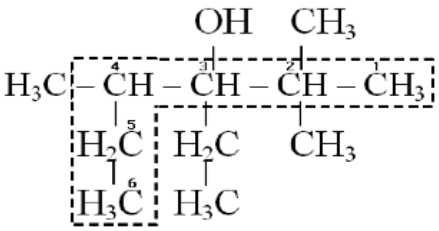

Figure 7. Respecting IUPAC Rules: 3-ethyl-2-2-4-trimethyl-3-hexanol

Example 2 - Analysis of *4-ethyl-3,5,5-trimethyl-4-hexanol*: after inferring that this name does not respect the IUPAC rules for violating the Lowest Number Rule (as shown in Figure 9), OCLAS adjusts the set of its lexical items *ethyl* (with locant 4), 3 radicals *methyl* (with locants 3, 5 and 5), *hexa*, *ane* and *ol* (with locant 4), and retrieves the same correct name *3-ethyl-2-2-4-trimethyl-3-hexanol* of the previous example (see figure *7*).

It is important to note that, in spite of being distinct, both inadequate names treated above represent the same real chemical compound illustrated in Figure 7. Further, although the sets of lexical items which correspond to the inadequate names are distinct one from the other, during the analysis OCLAS detects that, in fact, both represent the compound *3-ethyl-2-2-4-trimethyl-3-hexanol* whose lexical items are: 3 radicals *methyl* (all of them with locants 2), *hexa* and *ol* (with locant 3), and which presents the same chemical characteristics of the inadequate names analysed. It illustrates a very interesting case of lexical ambiguity solved by OCLAS during syntactic and semantic analysis. Solving this kind of ambiguity is not a trivial task, since analysis here does not consist just on detecting the lexical items of an input name *N* and on checking whether the way in which they are combined in N satisfies all the chemical constraints of these lexical items and all the concerning IUPAC nomenclature rules. More than this, whenever that combination does not succeed, the parser must try to retrieve from the original lexical items a new set of lexical symbols that can be combined such as to yield a real molecular structure with the same chemical characteristics expressed in N, as shown in more details in section 4.4. If the parser succeeds, it means that N is an inadequate name; otherwise, N is an incorrect one. Note that analogous problems of lexical ambiguity must be treated in Continuous Speech Recognition systems (which deal with speech signal in which the words are not isolated) and in Natural Language Translation systems. In the former ones, the difficulty consists on isolating the words, since the speech signal carries information about the speakers identity, his language, his physical and emotional state and his geographical and societal background [21]. In the later ones, the difficult consists on finding the appropriate words in the object language that represent the same meaning expressed in the words of the sentence in the source language [22].

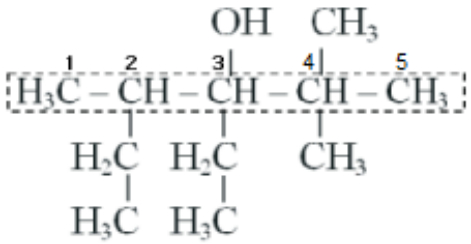

Figure 8. Violating Main-Chain IUPAC Rule: 2-3-diethyl-4-4-dimethyl-3-pentanol

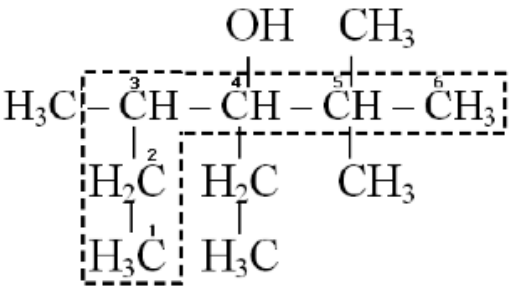

Figure 9. Violating Lowest-Number IUPAC Rule: 4-ethyl-3,5,5-trimethyl-4-hexanol

4.2. Main Tools Used in OCLAS

To cope with its objective of performing lexical, syntactic and semantic analysis of organic chemical compound names and, whenever this analysis succeeds, generating the pictures of their chemical structures, OCLAS counts on the following tools: the Generative Lexicon Theory (GLT), the Parser Combinators, the functional language CLEAN and the graphic pack Xymtex of Latex. As shown in section 2.3, the Generative Lexicon Theory performs analysis of sentences by trying to combine their lexical items according to their types. Such a strategy can be used to solve lexical ambiguity, since it allows to establish the meaning of an ambiguous lexical item by defining the type it must have in order to match the types of its complements in a sentence.

In this work, the GLT principals [11] are used to analyse sentences (names) of the Organic Chemistry Language taking into account the type of the lexical items that composes that sentences. These types are declared in the qualia structures that define the lexical items. In such a way, the Generative Lexicon Theory is used to solve ambiguity problems based on the type constraints expressed in the qualia structures of the lexical items. The relevance of the types in the process of analysis explains why OCLAS is implemented in the functional language

CLEAN, since it is extremely efficient to deal with types by virtue of its *uniqueness typing* and *transparency* proprieties [23].

Furthermore, the CLEAN counts on a friendly interface with the Parser Combinators used by OCLAS to combine lexical items in the syntactic and semantic analysis, as shown in section 2.3.

Finally, in order to endow OCLAS with the capacity of generating the pictures corresponding to the names of the organic compounds stored in the input file, the authors had to extend the graphic pack Xymtex of Latex, such as discussed later.

4.3. The Architecture of OCLAS

OCLAS is constructed according to the general architecture shown in the modules of figure 10.

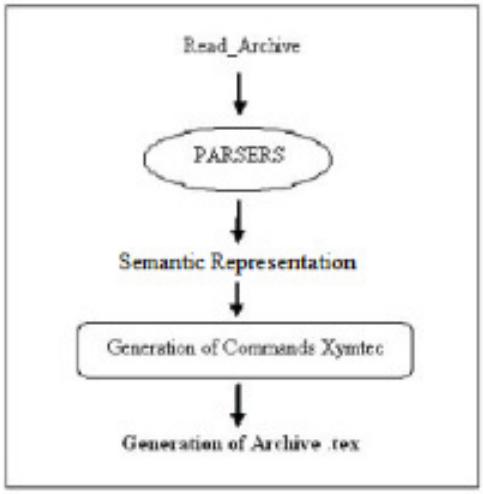

Figure 10. The OCLAS Architecture

The system performs the following sequences of actions: it reads the organic chemical compound names stored in the input file *test.pac* and generates, for each of them, a list of characters which the lexical, syntactic and semantic Parsers (module PARSERS) are able to manipulate. The lexical parser merely separates the lexical items of the current name. Next, in the syntactic analysis, the Parser Combinators tries to identify the category of each lexical item retrieved from the lexical analysis (prefixes, locants, main chain, side chains, insaturations and function identifier). The results obtained by the syntactic parser (lexical items and their respective categories and locants) are organized as data structures that will be passed as arguments to the functions responsible for the semantic analysis. The semantic parser tries to detect whether the lexical items, the categories and the positions (locants) received from the syntactic parser can be combined in such a way as to produce a correct name. If they can, the parser generates the semantic structure to be passed to the Xymtec Code Generator module. If

they can not, it tries to find an alternative combination of lexical items, categories and positions which will produce a correct name corresponding to the input name (obviously, both names must represent the same chemical compound). If the parser succeeds, it generates the semantic structure corresponding to the correct name obtained and passes it as an argument to the Xymtec Code Generator Module. Otherwise, the semantic parsing fails and OCLAS warn the user that he has proposed an incorrect name. As shown above, whenever the sematic parsing succeeds, the semantic structure produced by the parser will be passed as argument to the Xymtec Code Generator Module. This module, then, compiles the semantic structure received into Xymtec codes. This compilation process is another arduous work performed in the OCLAS environment, once it must produce a Xymtec code for each bond of the compound structure under analysis. Furthermore, in order to implement this compiler, the authors had to extend the Xymtec pack such as to make it able to represent the compiled codes (see 4.5.1). The execution of these codes by LATEX produces a visual representation of the chemical structure that corresponds to the compound name proposed by the user.

4.4. How OCLAS Utilizes the Principals of the TLG Qualia Structures

This section shows how the analysis performed by OCLAS fits the TLG formalism. The lexical items correspond to chemical terms such as: prefixes, suffixes, function identifier, main chains, side chains, radicals etc., which are represented by their qualia structures. Each compound name is obtained by combining theses qualia structures according to their type constraints, as exemplified below with the analysis of the name *3-ethyl-2-methyl-1-pentene*. In order to simplify the example and the comprehension of the analysis, the authors present in the qualia structures just the elements that are essential to explain the parsing process. Throughout the analysis, the lexical parser retrieves the locants *{1}*, *{2}* and *{3}* and the following lexical items: the insaturation suffix *ene*, the carbon chain *pent* and the radicals *2-methyl* and *3-ethyl*, whose qualia structures are shown in figures 11, 12, 14 and 16, respectively.

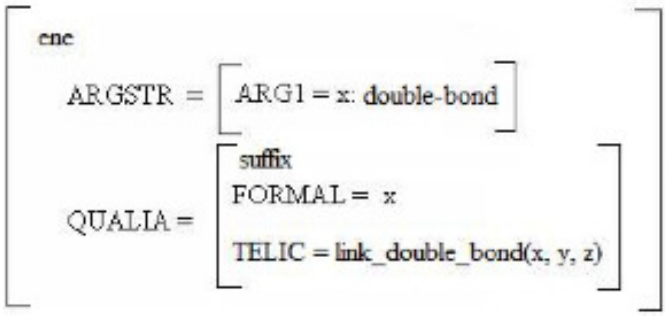

Figure 11. Qualia Structure of the suffix *ene*

In figure 11, *ene* is a lexical item (a suffix represented by x) characterized by the type *double-bond* (as indicated in the FORMAL role) to be passed as an argument to the qualia structure of a carbon chain *z* in order to generate an alkene. This application tries to insert a double bond in the y-th carbon atom of *z*, as shown in the expression of the TELIC role.

Figure 12 defines *pent* as a carbon chain *z* to be applied to the arguments *x, y* (that is, *z* is a process). The application is performed when the agentive role *assembly_function* is executed. According to the figure, the argument *x* of *pent* must belong to the type *TOP_1*. This type

corresponds to the LUB (see subsection 2.5) of the set of types which *pent* can be applied to. Considering the organic chemical functions which OCLAS deals with (see section 4.5) and respecting the constraints of the Organic Chemistry concerning the carbon chain *pent* (see section 2.6), *TOP_1* must be the disjunction (or union) of the types of the following lexical items: *ane, ene, yne, diene, diyne, ol, al* and *hyl*. The result of applying *pent* to *x* depends on the type of *x*, as shown below (remember that the type of *x* must belong to the set of types that composes *TOP_1*).

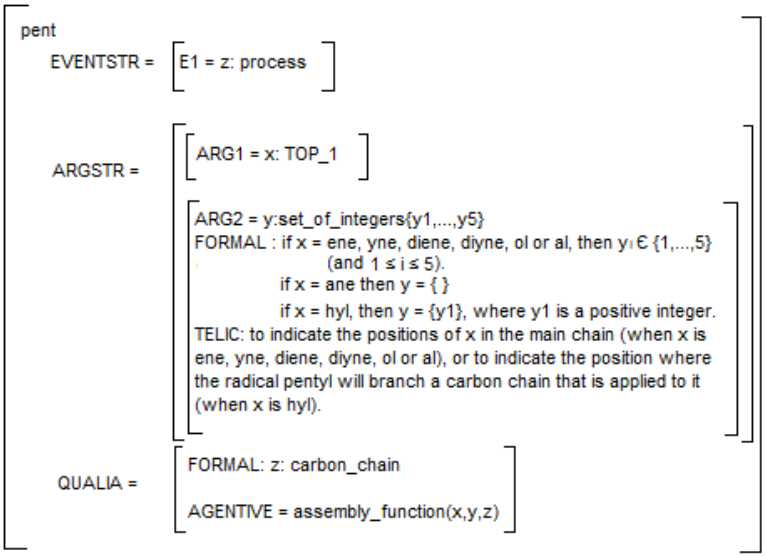

Figure 12. Qualia Structure of the carbon chain *pent*

If *x* is the suffix *ane, ene, yne, adyene, ol* or *al*, the application will produce a main chain corresponding to an alkane, an alkene, an alkyne, an alkadyene, an alcool or an aldehyde, respectively. That is why the argument *y* in the qualia structure of *pent* is a list of *n* integers $y_1,...,y_n$ indicating the positions in the carbon chain *z* (composed of 5 carbon atoms) where the appropriate elements corresponding to the suffix *x* must be inserted into. For example: if *x* is the suffix *adyene*, there are two elements to be inserted in *z*, that is, two double bonds (in this case, y_1 and y_2 will indicate the position of the first and the second double bonds, respectively, in *z*). Then, considering the original 12 free bonds of the five carbon atoms of *z* (that is, *pent*), it will remain just 8 ones.

If *x* is the function *ol*, there is just one element to be inserted in *z*: the alcohol identifier *OH*. Then, y_1 will indicate the position where the functional group *OH* (alcohol identifier) will be inserted into *z*. In this case, from the 12 free bonds of *z*, it will remain just 11. If *x* is the suffix *ane*, there is nothing to be inserted in *z* (then, *y* is an empty list). In this case, all the free bonds of *pent* will be filled in with hydrogen atoms and the application will produce an alkene.

If *x* is the suffix *hyl*, the application will generate a radical y_1-*pentyl* to be inserted in the y_1-*th* position of a carbon chain that is applied to it. Now, considering the qualia structures of *pent*

and *ene* summarized above, it is possible to go on with the analysis of the name *3-ethyl-2-methyl-1-pentene* proposed in the beginning of this section.

According to the qualia structures shown in figures 11 and 12, *pent* (that is, the process *z*) can be applied receiving as arguments the suffix *ene* (parameter *x*) and the set of integers {1} (parameter *y*), once *ene* matches the type *TOP_1* of *pent* and the element *1* of *y* (that is, y_1) belongs to the set {1, 2, 3, 4, 5}. This application is performed by the *agentive assembly_function* of *pent* and generates the lexical item *1-pentene* whose type is a process to be applied in order to generate another chemical structure. The qualia structure of *1-pentene* is illustrated in figure 13.

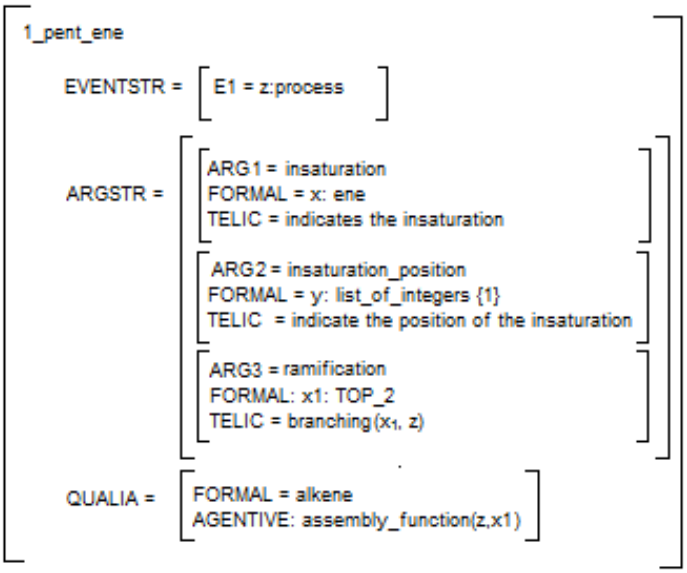

Figure 13. Qualia Structure of *1_pentene*

In its argument structure ARGSTR (see subsection 2.3), ARG1 and ARG2 correspond to the arguments *x* and *y*, respectively, received from the *assembly-function* (agentive) of *pent*. Then, *x* is *ene* and *y* is a list composed of the integer 1 (since $y_1 = 1$), which indicates that the double bond is placed in the first of the five carbon atoms of pent. ARG3 corresponds to the argument x_1 to which the lexical item 1-pentene (parameter z) can be applied, considering the constraints of the Organic Chemistry and the functions treated by OCLAS. Consequently, x_1 must have a type *TOP_2* that corresponds to the LUB of the types *M2, M3, M4, E2* and *E3*, which represent the types of the radicals *2-methyl, 3-methyl, 4-methyl, 2-ethyl and 3-ethyl*, respectively. As indicated in the TELIC role of the argument x_1, its objective is to branch the main chain *1-pentene* (represented by *z* in the figure). Note that the type of x_1 establishes the constraints which x_1 must satisfy in order to be an argument of *1-pentene*, that is, in order to branch it without violating the main chain rule. For example, a radical *4-ethyl* (type *E4*) cannot branch *1-pentene*, otherwise it would define a chain of 6 carbons that is longer than the carbon chain of *1-pentene* (that is why *E4* does not belong to *TOP_2*). The branching of *1-pentene* is performed when the agentive function *assembly-function applies pentene* (parameter, *z*) to the radical x_1. In

the given example (analysis of *2-methyl-3-ethyl-1-pentene*), x_1 will receive the value *2-methyl*. The lexical item *2-methyl* is obtained by applying the qualia structure of the carbon chain *met* (that is defined in the same way as *pent*) to the suffix *hyl* and to the set of integers {2} provided by the lexical parser, as shown in Figure 14.

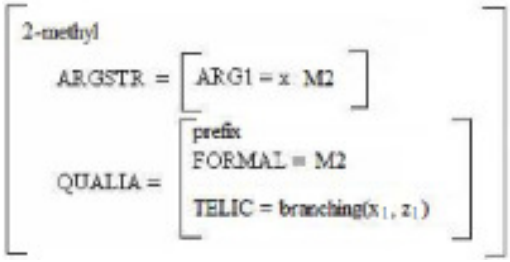

Figure 14. Qualia Structure of the radical *2-methyl*

That is why the first argument *y* of *2-methyl* is the list of integer *{2}*: it indicates that the radical *methyl* will be inserted into the second carbon atom of a carbon chain that is applied to it. As the type *M2* of the lexical item *2-methyl* is unifiable with the type *TOP_2* of x_1, the qualia structure of 1_pentene can be applied to the qualia structure of *2-methyl*. This application generates the lexical item *2-methyl-1-pentene* whose type is process, as illustrated in figure 15.

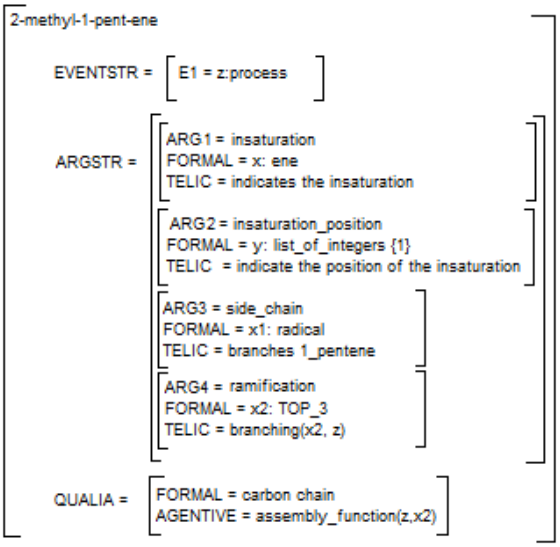

Figure 15. Qualia Structure of the compound *2-methyl-1-pent-ene*

As seen before (in the description of the qualia structure of *1-pentene*), the first, second and third arguments of *2-methyl-1-pentene* have been received from precedent applications and indicate the position of the insaturation, the insaturation itself (double bond) and the side chain

(radical *2-methyl*), respectively. The fourth argument x_2 corresponds to the one to which the process *2-methyl-1-pentene* can be applied in order to generate another possible chemical structure. The type of x_2 is *TOP_3*, which corresponds to the LUB of the types *M3, M4* and *E3*, respectively. Consequently, by using its *assembly-function*, *2-methyl-1-pentene* can be applied to the radical *3-ethyl* (whose type is *E3*) shown in figure 16.

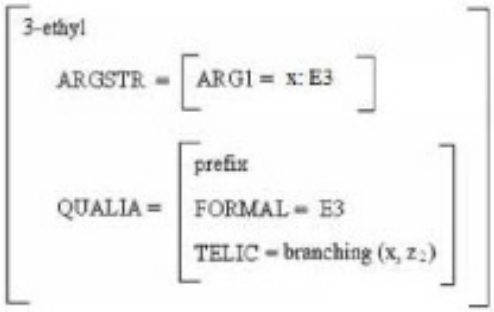

Figure 16. Qualia Structure of the radical *3_ethyl*

Finally, this application generates *3-ethyl-2-methyl-1-pentene* whose qualia structure is shown in figure 17.

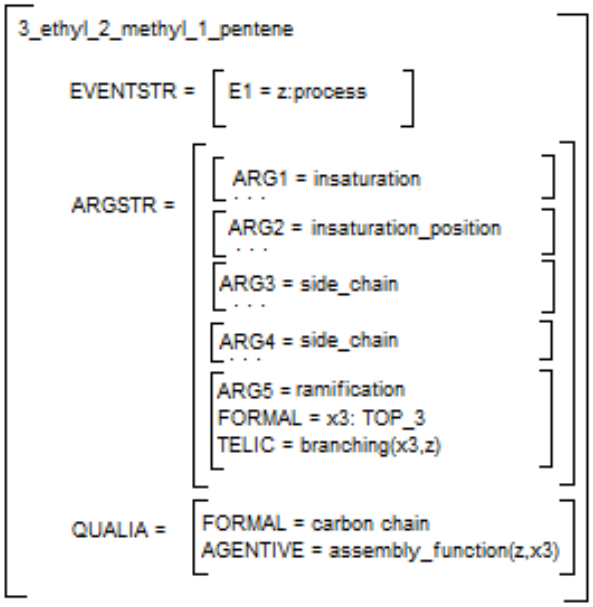

Figure 17. Qualia Structure of the compound *3_ethyl_2_methyl_1_pent_ene*

Therefore, *3-ethyl-2-methyl-1-pentene* is a correct name and the OCLAS is able to generate its chemical structure picture shown in figure 18.

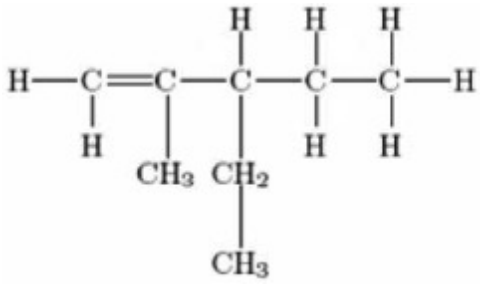

Figure 18. Picture of the compound *3_ethyl_2_methyl_1_pentene*

As said before, OCLAS is also able to analyse inadequate names and to correct them. It manages it by means of a set of rules for treating ambiguity that have been implemented to solve this problem, as explained below. Figure 17 show that *3-ethyl-2-methyl-1-pentene* can still be applied to arguments of type *M3, M4* or *E3* (which generates the LUB *TOP_3*), as shown in the FORMAL of ARG5. In this way, the analysis of the name *4-ethyl-3-ethyl-2-methyl-1-pentene* differs from the analysis of *3-ethyl-2-methyl-1-pentene* just at the end of the process, when the system tries to apply the qualia structure of the later name to the radical *4-ethyl* (type *E4*). However, *E4* does not unify with the type {*TOP_3* (otherwise, the result of the application would be a name that violates the main chain rule). It means that the semantic analysis of *4-ethyl-3-ethyl-2-methyl-1-pentene* has failed. Then OCLAS tries to find a disambiguation function capable of generating a correct name corresponding to it. In order to solve an inadequate name, the disambiguation functions take into account the set of lexical items corresponding to the input name and consider the constraints established by the main and lowest numbers rules. Therefore, particularly in the example above, the disambiguation function *f* that treats the input name retrieves as lexical items the locants *4*, *3*, *2* and *1*, the radicals *methyl* and *ethyl*, the carbon chain *pent* and the insaturation indicator *ene*. Further, guided by the constraints established by the main chain and lowest numbers rules, *f* detects that a radical *ethyl* can branch the 4-th carbon atom of the compound *3-ethyl-2-methyl-1-pentene* shown in figure 17, but, in this case, the following two modifications will occur: first, the carbon atoms of this radical *ethyl* will be incorporated into the main chain in order to compose a longer chain of 6 carbon atoms (then, this radical will not exist anymore and the new main chain becomes *1-hexene*); second, the 5-th carbon atom of *3-ethyl-2-methyl-1-pentene* becomes a radical *4-methyl*, since it does not belong to the main chain anymore. As a consequence of this second modification, *f* infers that now there are two radical *methyl* (*4-methyl* and *2-methyl*) branching the main chain *1-hexene*, what requires the insertion of the multiplying prefix *di* into the name.

From this point on, *f* is able to generate the appropriate set of lexical items that corresponds to the original set of lexical items presented above. This new set is composed of the following elements: the locants *4*, *3*, *2* and *1*; the multiplier *di*; one radical *ethyl*; two radicals *methyl*; the carbon chain *hexa*; and the insaturation indicator *ene*.

Finally, *f* is able to generate the correct structure *3-ethyl-2,4-dimethyl-1-hexene* corresponding to the inadequate name *4-ethyl-3-ethyl-2-methyl-1-pentene*, as shown in Figure 20. Figure 19 shows how this function *f* described above is implemented in OCLAS (section 4.5 gives more details about implementation in the system).

```
f "pent" [x,w,z][y] | ((getPos x)==4) && ((getPos w)==3)
                    && ((getPos z)==2) && (y==1)
                    && (equiv x (Ethyl 4))=
                   Alkene [C(WL, H, H, DL), C(WL, DL, z, SL),
                           C(WL, SL, w, SL),
                           C(H, SL, (Methyl 0), SL), C(H, SL, H, SL),
                           C(H, SL, H, H)];
```

Figure 19. The disambiguation function *f*

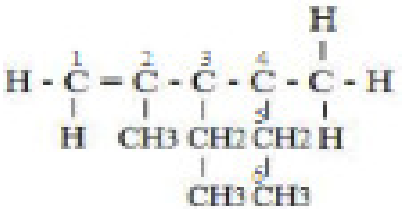

Figure 20. Picture of *3-ethyl-2,4-dimethyl-1-hexene*

Note that despite of the fact that OCLAS and the system proposed by Frost (see section 3) use both the resources of parser combinators to attack ambiguity problems, specific peculiarities of chemical languages force OCLAS to behave in a different way to cope with ambiguities. In fact, in addition to the lexical ambiguities treated by Frost, OCLAS also deals with a hard kind of lexical ambiguity that, in order to be solved, requires an extra expertise of the system: the ability of modifying and recombining the lexical items of an inadequate input name (provided by the lexical parser), such as to generate a new set of lexical items appropriated to produce a correct name which represents the real compound that corresponds to the input name. Remember that, as shown in section 3, the natural language ambiguities handled by Frost, Hafiz and Callaghan do not require from the system the ability of altering the lexical items of the ambiguous input sentences such as to produce a correct sentence. Instead of it, the system just tries to find all the possibilities to combine them for producing correct sentences.

4.5. The Implementation

This section resumes how OCLAS uses the Parser Combinators to implement the analysis process described in the previous section. The prototype that has been implemented treats the following chemical functions: hydrocarbons (alkanes, alkenes, alkynes and alkadienes), alcohols and aldehydes. Each of these chemical functions is treated by a set of Clean functions implemented according to its specificities. The Parser Combinators are used to combine the results obtained during the lexical, syntactic and semantic analysis. In order to illustrate this process, it follows a brief explanation of how the compound name *3-ethyl-1,2-pentadyene* is analysed. Initially, the Lexical Parser retrieves the set of locants {3} and {1,2} and the tokens {*et, hyl, 1,2, pent* and *diene* which compose the name. These elements are passed as arguments to the function *chain* shown below:

```
(1) chain = alkaneMainChain  <!> alkeneMainChain <!>
            alkyneMainChain <!> alkadyeneMainChain <!>
            alcoholMainChain <!> aldehydeMainChain;
```

The function *chain* uses the combinator <!> (see section 2.3) to combine the parsers alkaneMainChain, alkeneMainChain etc, which represent the chemical functions that have been implemented (alkane, alkene etc). Each parser comprises the set of Clean functions necessary to identify and to analyse chemical compound names belonging to a chemical function. Therefore, these parsers are responsible for performing the actions that the system must execute in order to

analyse correct, incorrect or inadequate names (these actions are described in section 4.3 and 4.4). Thus, in the example proposed above, when the function *chain* receives from the Lexical Parser the tokens corresponding to *3-ethyl-1,2-pentadyene*, only the parser *AlkadyeneMainChain* succeeds in the task of recognizing and assembling them such as to generate a semantic representation. Note that this parser is composed of several other parsers. The parser of *AlkadyeneMainChain* that can analyse the name of the example is called *withoutMultAlkadyenes* (once it is able to deal with alkadyenes that present no multiplying prefix) and is shown in (2) below:

```
(2) withoutMultAlkadyene = radicalsAlkadyene <&> \s-> (posLinkDyene) <&>
                                                    \j-> (alkadyeneCarbonChain <@
                                                    (\x-> (mkAlkadyene x s j)))
                                                    <& alkadyeneFunction;
```

In (2), *alkadyeneCarbonChain*, *radicalsAlkadyene* and *pos-LinkDyene* are syntactic parsers. The parser *alkadyeneCarbonChain* tries to identify the main chain of the input name. The parsers *radicalsAlkadyene* and *posLinkDyene* check whether and where there are radicals or double bonds, respectively, in the main chain. *AlkadyeneFunction* is also a syntactic parser whose role is to check whether the chemical name represents an alkadyene. *MkAlkadyene* is a semantic parser whose objective is to try to build the semantic structure corresponding to the alkadyene under analysis. Thus, *radicalsAlkadyene*, *posLinkDyene* and *alkadyeneCarbonChain* output the list *[(ethyl 3)]*, the list *[1, 2]* and the string *pent*, respectively. The role of the syntactic parser *alkadyeneFunction* is to confirm whether the name is really an alkadyene (by trying to find the suffix *adyene*). The analysis of *3-ethyl-1,2-pentadyene* performed by *WithoutMultAlkadyene* can be resumed as it follows (see subsection 2.3): the combinator <&> passes the output *[(ethyl 3)]* of *radicalsAlkadyene* as an argument to the lambda abstraction that represents *posLinkDyene* (by means of the parameter \s); furthermore, the difference-list corresponding to the expression *1,2-pentadyene* that remains from the execution of *radicalsAlkadyene* is passed to *posLynkDyene*. Next, the list *[1, 2]* of *posLinkDyene* and the difference-list composed of *pentadyene* are passed to *alkadyeneCarbonChain* (the list *[1, 2]* is passed by means of the parameter \j). The parser *alkadyeneCarbonChain*, then, selects in its input difference-list the carbon chain *pent* and transfers a new difference-list containing *adyene* to the last syntactic parser *alkadyeneFunction*, which simply checks whether this remaining lexical item represent an alkadyene. Note that the lexical item *adyene* selected by *alkadyeneFunction* is merely used to identify the function *alkadyene* to which the chemical name belongs. Thus, after the identification, it can be discarded, that is, the combinator <& does not passes it to the parser *mkAlkadyene*. Further, the parser *alkadyeneCarbonChain* passes its output *pent* to the semantic parser *mkAlkadyene* by means of the parameter *x*. This later parser has as role to check whether the outputs *x*, *s* and *j* of the syntactic parsers *radicalsAlkadyene posLinkDyene*, *alkadyeneCarbonChain*, respectively, can be semantically combined. In other words, it checks whether the radical *3-ethyl* can branch a chain of five carbon atoms that has two double bonds: one placed in the first carbon atom and the other in the second one. Thus, the semantic parser *mkAlkadyene* is a kind of *assembly-function* shown in the previous subsection, that is, it performs the role agentive expressed in the qualia structure of the lexical item *pentadyene* (whose type is *process*).

The semantic analysis is started by the combinator <&, which commands the application of the semantic parsers to the results obtained by the syntactic parsers (as illustrated in (2)). Whenever this application fails, OCLAS activates the disambiguation functions in order to try to find a correct name that corresponds to the input name, as shows in subsection 4.4. If it succeeds, it generates the semantic structure and the picture of the correct compound name. Otherwise, it

warns the user that the name that he has proposed is incorrect. The syntactic analysis of *3-ethyl-1,2-pentadyene* can be resumed as shown in Figure 21.

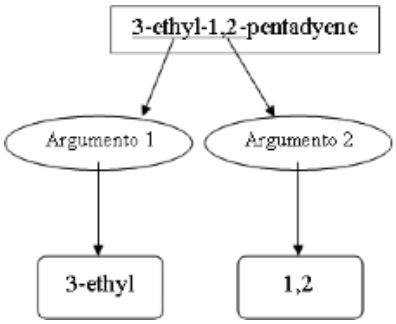

Figure 21. Syntactic Analysis of *3-ethyl-2,4-dimethyl-1-hexene*

The application of the semantic parser *mkAlkadyene* to the results of the syntactic parsers can be seen in the expression (3) below.

(3) mkAlkadyene "pent" [x][y,z] | ((getPos x)==3) && (y==1) && (z==2) = Alkadyene [C(WL, H, H, DL),C(WL, DL, WL, DL), C(WL, DL, x, SL),C(H, SL, H, SL), C(H, SL, H, H)];

In (3), the 4 arguments of each carbon atom correspond to its bonds, where, WL indicates "absence of branching", H represents a simple bond with a hydrogen atom, DL indicates a double bond with a carbon atom, SL indicates a single bond with a carbon atom and *x* corresponds to the prefix *ethyl* (to be placed in the third carbon atom, as indicated). Furthermore, *y* and *z* (here, equal to 1 and 2, respectively) indicate in which carbon atoms the two double bonds will be placed (in this case, in the first and the second carbon atoms).

For instance, in the third carbon atom C(WL, DL, x, SL) of (3), there is a double bond connecting it to the second carbon atom and three simple bonds connecting it to a hydrogen atom, to a radical *ethyl* and to a carbon atom, respectively. The result of the application (3) corresponds to a semantic structure that is passed as argument to the compiler LATEX. The output of this compilation process is the following XYMTEC object code:

```
\begin{picture}(1200, 600)(0,0)
\put(0,0){\tetrahedral{0==C;2==H;3==H;4D==}}
\put(240,0){\tetrahedral{0==C;2D==;4D==}}
\put(480,0){\tetrahedral{0==C;2D==;3==
\put(-260, -355){\tetrahedral{0==CH2;3==;1==}}
\put(-260,-650){\tetrahedral{0==CH3;1==}};4==}}
\put(720,0){\tetrahedral{0==C;1==H;2==;3==H;4==}}
\put(960,0){\tetrahedral{0==C;1==H;2==;3==H;4==H}}
\end{picture}
```

Finally, the execution of this code produces the picture corresponding *3_ethyl_1,2_pentadyene* shown in Figure 22.

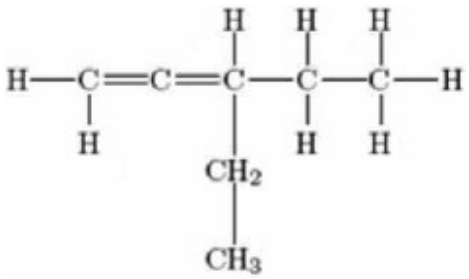

Figure 22. OCLAS representation for *3-ethyl-1,2-pentadyene*

As said before, the Xymtec pictures of the chemical structures yielded by OCLAS is similar to the ones found in the didactic books of Organic Chemistry, that is, they are very clear and illustrative, which is a fundamental requirement for an automatic instructor that intends to be a helpful and friendly utilitarian. In order to show the advantage of using Xymtec as a tool for generating chemical structure representations, it is interesting to compare the representations produced by OCLAS (using Xymtec) and by the Anstein-Kremer's system (using Smile strings). Figure 23 shows the picture produced by the latter to represent the compound *3-ethyl-1,2-pentadiene*. In fact, the picture generated by OCLAS for the same compound (see Figure 22) is much more clear and didactic than the one generated by the Anstein's system.

C=C=C(CC)CC

Figure 23. Anstein's representation for *3-ethyl-1,2-pentadyene*

Subsection 4.5.1 show in more details how OCLAS extends XYMTEC in order to process the chemical formulae.

In the same way that OCLAS counts on the semantic parsers *mkAlkadyene* to analyse the alkadyenes (see example above), it counts on other semantic parses to deal which the remaining functions. For instance, it defines *mkAlkyne*, *mkAlcohol* and *mkAldeyde* to treat the akynes, the alcohols and the aldeydes, respectively.

4.5.1. Extending Xymtec in Order to Generate Chemical Structure Pictures

Xymtec is a very useful tool for drawing chemical compounds, once it generates illustrative and clear pictures very suitable to didactic purposes. The chemical structure picture corresponding to a correct or to an inadequate name analysed by OCLAS is obtained whenever the *Xymtec* code produced by the analysis is executed. In order to cope with this task, in OCLAS the original *Xymtec* library needed to be extended, once it does not own commands capable of drawing the branchings of a main carbon with their side-chains in the positions specified by the locants (see section 2.6.1). To solve this limitation, it was necessary to treat the prefixes (locants and radicals), so that they could be correctly connected to the main chains. It was achieved by manipulating the values of the coordinates of the command **\put** of Latex. This command allows to insert pictures into determined positions defined by its parameters. For instance, this command combined, with the command **\tetrahedral** (see section 2.6.2), allows the insertion of a tetrahedral in the positions indicated by the coordinates of the command **\put** (see Figure 22).

5. CONCLUSIONS AND FUTURE WORKS

The system OCLAS presented here is a very useful automatic Organic Chemistry instructor specialized in the analysis of names of organic chemical compounds and in the generation of their chemical structure pictures. OCLAS is able to analyse correct names as well as incorrect or

inadequate names. In this last case, it must be able to solve complex ambiguity problems related to Organic Chemistry terminology. The abilities of analysing and correcting the ambiguities of the inadequate names and of using an optimized extension of *Xymtec* to represent the pictures of the chemical structures in a very friendly and didactic way, distinguish OCLAS from the other existing similar systems. The prototype implemented is able to treat six chemical functions: the alkanes, the alkenes, the alkynes, the alkadyenes, the alcohols and the aldheydes. Neverthless, it can be easily extended such as to deal with other chemical functions. As future works, the authors intend to extend the abilities of OCLAS in the following ways: making the system able to analyse other chemical functions (including cyclic organic compounds); making the system able to analyse results obtained by vibration spectrometers such as to infer the chemical functional group to which a compound belongs from the bands of spectral absorption of their molecules.

REFERENCES

[1] http://www.iupac.org.

[2] S. Anstein and G. Kremer. (2005) Analysing names of organic chemical compounds from morpho-semantics to smiles strings and classes. Master's thesis, Universitat Stuttgart.

[3] A. J. Agrawal and O. G. Kakde. (2011) *Object-Relational Database Based Category Data Model for Natural Language Interface to Database*. International Journal of Artificial Intelligence & Application (IJAIA), Vol. 2, N. 01, pages 35-41.

[4] D. Jiao and D. J. Wild (2009) Extraction of CYP Chemical Interactions from Biomedical Literature Using Natural Language Processing Methods. J. Chem. Inf. Model. 49, pages 263–269.

[5] J. Allen.(1987) *Natural Language Understanding*. The Benjamin/Cummings Publishing Company.

[6] S. J. Russell and P. Norvig. (2009) *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 3. edition.

[7] J. R. Partington. (1989) *A Short History of Chemistry*. Dover Publications.

[8] IUPAC. (1993) *A Guide to IUPAC Nomenclature of Organic Compounds*. Blackwell Scientific.

[9] M. Fogiel. (2000) *Organic Chemistry I - Super Review. All you need to know!* Research and Education Association.

[10] P. Ertl. (2003) *Cheminformatics analysis of organic substituents: Identification of the most common substituents, calculation of substituent properties, and automatic identification of drug-like bioisosteric groups*. J. Chem. Inf. Comput. Sci., 43(2):374˝U380.

[11] J. Pustejovsky. (1995) *The Generative Lexicon*. The MIT Express.

[12] P. K. et al. (1997) *Parser combinators*. Web -ftp://ftp.cs.kun.nl/pub/Clean/papers/cleanbook/.

[13] Springer. (1995) *Functional Pasers*. Tutorial text of the First international spring school on advanced functional programming techniques.

[14] P. K. et al. (1997) Parser Combinators.

[15] J. W. Lloyd. (1947) *Foundations of Logic Programming*. Springer-Verlag Berlin Heidelberg, New York.

[16] S. Fujita. (1993) *Xymtex: A macro package for typesetting chemical structural formulas.*

[17] S. Fujita. (1994) *Typesetting structural formulae with the text formatter tex/latex*. Computers & Chemistry, 18(2):109–116.

[18] R. A. Frost, R. Hafiz, and P. Callaghan. (2008) *Parser combinators for ambiguous left-recursive grammars*. In PADL, pages 167–181.

[19] H. Abe, S. Takahashi, and S. ichi Sasaki. (1991) *Computer-aided generation of iupac nomenclatures for acyclic compounds*. Journal of Mathematical Chemistry.

[20] K. W. Raymond. (1991) *A lisp program for the generation of iupac names from chemical structures*.

[21] D. Jurafsky and J. H. Martin. (2000) *Speech and Language Processing: An Introduction to Natural Language Processing*, Computational Linguistics and Speech Recognition. Prentice Hall.

[22] B.S.Pederson.(1997)Lexical Ambiguity in Machine Tanslation:Expressing Regularities in the Polysemy of Danish Motion Verbs.Phd thesis, Copenhagen-DK.

[23] R. P. Edsko de Vries and D. Abrahamson. (2008) *Uniqueness typing simplified*. Olaf Chitil, ZoltÅLan HorvÅLath and ViktÅLoria ZsÅLok (Eds.): IFL 2007, LNCS 5083, pages 201–218.