Triangle: Language, Literature, Computation, n. 6, 2011
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

Aggregation with Recombination Patterns

M. Dolores Jiménez-López

Research Group on Mathematical Linguistics,
Rovira i Vigili University
Tarragona, Spain
E-mail: mariadolores.jimenez@urv.cat

Summary. In this paper, we show the commonalities between *aggregation* processes in Natural Language Generation and *recombination patterns*, a framework introduced recently as a way of generating complex sentences in natural languages using very simple recombination –and therefore biological– rules. By showing similarities between these two mechanisms, we suggest the possibility of carrying out aggregation by means of recombination patterns. We also refer to the possibility of using such a biological-motivated framework in the design of efficient and simple natural language generation devices.

1 Introduction

Natural Language Generation (NLG) may be seen as the technique of letting a computer automatically create natural language out of a computational representation. In order to generate natural language from computational representations, different processes must be carried out. First, from the knowledge base, it must be decided *what to say*: this is the so-called *content selection* phase. Then, during the so-called *text plan*, the order in which the sentences should be generated to make the text coherent is decided. Then *sentence planning* is carried out. And finally, in the so-called *surface*

generation, the syntactic structures and lexical choices (*how to say it*) are performed. Part of the sentence planning is the task of *aggregation*.

Aggregation has become a popular research topic in the field of NLG [8, 10]. It is usually described as a process in generation that consists of removing redundant information without losing information. Research on this topic has proved to be important in NLG because without aggregation automatically generated text is often very poor. In contrast, aggregation improves the quality of generated text by making it more fluent, concise and easy to read. Vastly different approaches to aggregation can be found in the literature and they are often not compatible. The definition of aggregation provided in [3], and reported below, is considered to be one of the best since it summarizes what other researchers consider this task to be:

> 'Functioning as one or a set of processes acting on some intermediate text structures in text planning, aggregation decides which pieces of structures can be combined together to be realized as complex sentences later on so that a concise and cohesive text can be generated while the meaning of the text is kept almost the same as that without aggregation.'

In this paper we want to show the commonalities between that important area in NLG and a formalism that have recently been defined in [9] with the aim of showing that the mechanism of recombination –found in Biology– also works in natural language. This new framework –called *Recombination Patterns*– shows that it is possible to generate a natural language (or an important part of it) by starting with a base –composed of a finite number of simple sentences and words– and by applying a small number of recombination rules. Initially, the formalism is defined for the generation of complex sentences in English, Italian and Spanish.

It is accepted that people do *aggregation* all the time to make natural language expressions shorter, non-redundant and easy to read. On the other hand, it seems that *recombination patterns* can explain in a very *natural* fashion the way in which people process natural language. So, both aggregation and recombination patterns could be found in the *human* processing of natural language. Now, since software engineering tools, data bases or expert systems are often highly redundant, aggregation processes have been defined and implemented for NLG devices. What we claim here is that recombination patterns also – such as aggregation– can also be applied to the

field of NLG and that such a new formalism may facilitate the definition of simple and efficient natural language generation devices.

The great deal of similarities between aggregation processes that have been defined in the area of NLG and recombination patterns lead us to suggest here that it may be possible to carry out aggregation by means of recombination patterns. Moreover, as we have already said, we suggest that the naturalness of recombination patterns may enable them to be used in the area of NLG.

The paper is divided into six sections. Section 2 and 3 briefly introduce aggregation and recombination patterns, respectively. Section 4 compares two issues. Section 5 suggests that it may be possible to used recombination patterns in natural language generation. Finally, some final remarks and basic references are presented.

2 An overview of aggregation

Researchers in the field have different understandings of what aggregation is. According to [11], aggregation is taken to be 1) redundancy elimination, 2) abbreviation, 3) text structure combination for concise and coherent text generation, 4) the combination of text components at any level to achieve a cohesive text, 5) the syntactic expression of concise and tightly constructed text and the generation of fluent, more readable and less boring text, etc. The scope of this section is, as far as possible, to shed light on this topic by providing answers to the questions: what, why, when and where aggregation. At the end of the section, the reader should have a clear idea of what aggregation is, and will be able to understand the comparison we will make with recombination patterns.

What is aggregation? Aggregation is the process of removing redundant information in a text without losing any information. In [11], two usages of the term 'aggregation' are pointed out:

1. the *narrow sense*: 'Aggregation is any process which maps one or more structures into another structure which gives rise to text which is more x-aggregated than would otherwise be the case. X-aggregated text is text which contains no multiple nonpronominal overt realizations of any propositional content.'

2. the *broad sense*: 'Aggregation is the combination of two or more linguistic structures into a single linguistic structure which contributes to sentence structuring and construction.'

According to [16], aggregation is not a process, a task, or a set of decisions in generation, but a *reason* for decisions to be made in a certain way. Aggregation is not an end in itself. Aggregation is an emergent phenomenon. From this perspective, aggregation is neither a process nor a goal of processes; rather, it is a characteristic of texts which have been generated properly.

Why is aggregation done? There are as many reasons as there are goals. People and systems must perform aggregation to make their text more readable, understandable, fluid, concise, coherent, cohesive...: not doing so risks the reader's misunderstanding or irritation. We assume axiomatically that shorter and less redundant text is better text.

When is aggregation done? Assuming that in NLG the tasks carried out are *content selection*, *sentence planning* and *surface generation*, we may say that aggregation takes place after content selection and *before surface generation.* So, aggregation is part of the *sentence planning*. Anyway it can be carried out whenever the appropriate structures arise, it can take place during every phase of NLG except during content selection and surface form generation. This leads to the view that there need not be a specific aggregation module in a generation pipeline. Rather aggregation can be done whenever possible on an opportunistic basis.

Where is aggregation done? Wherever the appropriate structures arise. In fact, there seems to be no aspect of language generation which can be excluded from a thorough consideration of aggregation.

What is aggregation done to/on? In order to answer this question, we can refer to the several types of aggregation that have been described. The typology suggests that it can be done at any level of linguistic representation. Types of aggregation differ from author to author. [10] distinguishes: 1) *Embedding or hypotactic aggregation*; 2) *paratactic aggregation*; and 3) *lexical aggregation*. A similar classification can be found in [3] where the types of aggregation are: 1) *Lexical aggregation*; 2) *embedding*; 3) *hypotactic aggregation*, and 4) *paratactic aggregation*. Now, combining the information from [11] and [5] we offer the following classification of general types of aggregation:

1. *Conceptual aggregation*. Conceptual aggregation typically reduces the number of propositions in the message while increasing the complexity of the value of some conceptual role.
2. *Discourse aggregation.* This reduces the complexity of rhetorical structure but increases the complexity of one of the propositional leaves.
3. *Semantic aggregation.* We take it that conceptual information is non-linguistic, language-independent, domain knowledge and that semantic information is linguistic, language-dependent representations of meaning. An example of semantic aggregation might be the mapping of the meanings of *'Jamie is Chris's sister'* and *'Chris is Jamie's brother'* to the meaning of *'Chris and Jamie are brother and sister'*.
4. *Syntactic aggregation*. This removes redundant information, but leaves (at least) one item in the text to carry the meaning explicitly. This is carried out at a pure syntactic level with no information loss about the content of the aggregation items. It is the most common form of aggregation. The two most common rules are (a) the subject grouping rule and (b) the predicate grouping rule.
5. *Elision*. This removes information that can be inferred and leaves no items in the text to carry the information explicitly but the information remains there implicitly.
6. *Lexical aggregation*. This replaces a set of items with a new item, while the overall meaning is kept intact.
7. *Referential aggregation*. This replaces redundant information with some sort of trace, such as a pronoun, to carry the information explicitly.

The systems which implement aggregation are designed in a number of different ways. Their architectures can be classified under the following four categories, on the basis of where aggregation is performed:

1. *Independent sentence-planning module*. This treats aggregation as a microplanning subtask which happens after text planning and before surface realization. When it runs, the full contents have been selected and the rhetorical structure of the text is determined. The module attempts to improve the text quality by performing aggregation on propositions which are either adjacent or close enough to be brought together, but the interaction with other NLG tasks is minimal.

2. *Opportunistic text planning*. This views aggregation as a tool used by text planning or content selection. It is used to generate concise summaries of some input data. The first planning task is to determine what information is essential, and what information is optional. All essential information is included in the text plan, but then aggregation, especially embedding, is used to determine which optional facts should be included. If an optional fact can be expressed concisely, it is included; otherwise it is left out.
3. *Discourse Organization Module*. This approach combines ideas from the first two approaches. It assumes that the content is selected as a first independent step and that all facts passed on must be included, but then it uses the technique of opportunistic planning to build the text structure, yielding a concise and flowing text. It is one of the best ways to perform aggregation when content determination can be done up-front without regard to realization issues. This approach gives the application a high degree of flexibility.
4. *Revision-based generation.* This does not contrast with the former types; rather it is a particular feature of some systems in the other categories. It is based on human writing analysis which concludes that writing is usually done in three heavily interwoven (and recursively nested) phrases: planning, sentence generation, revision. Since humans generate multiple drafts before producing a final text, it is reasonable for NLG systems to do the same.

3 Introducing recombination patterns

In [9], we define *recombination patterns* for natural language syntax with the aim of showing that the mechanism by which genetic material is merged –that is, recombination– works and is also valid in natural languages. The main scope of the idea we propose is not so much to explain the human processing of language as to be able to offer a formalism that may simplify generation mechanisms for natural language; that is to define a formalism that may allow the construction of natural language processing systems that are as simple and efficient as possible. We are somehow convinced that recombination may well explain the way in which speakers combine linguistic elements (words, phrases, sentences) that they already know in order to construct new linguistic structures. More specifically, our research focus on

the possibility of generating a natural language (or an important part of it) using a finite number of words, phrases, sentences... and by only applying recombination rules. In order to show that recombination may well describe natural syntax we have focused, up to now, on the study of complex sentences in three languages: English, Italian and Spanish. We have defined different formulae –called *patterns*– for coordinate and subordinate sentences in these three natural languages. Up to now, we have defined 379 patterns that allow us to generate complex sentences in English, Italian and Spanish by the *recombination* of simple sentences. Each pattern is a very simple formalism divided into three different parts:

1. The first one –called *pattern recognition*– defines in terms of some basic categories –proposition, substantive, noun, etc.– the two propositions we always have at hand in order to generate complex sentences.
2. The second part –called *pattern recombination*– is where the application of different recombination rules (insertion, deletion, transposition, transformation, etc.) takes place. Taking into account that in every pattern we start by having at hand two propositions, this phase starts by placing Proposition 2 in some place with reference to Proposition 1. It does so by performing either
   - an *initial insertion*. In this type of insertion, Proposition 2 is situated before Proposition 1. The typical instruction here says: `Insert Prop.2 before Prop.1;`
   - a *median insertion*. Here, Proposition 2 is placed after the Subject of Proposition 1, the instruction being something like `Insert Prop.2 after Subject in Prop.1;` or
   - a *final insertion*. Final insertion situates Proposition 2 after Proposition 1. So, the instruction here says: `Insert Prop.2 after Prop.1.`

   After the positioning of Proposition 2 with respect to Proposition 1, some recombination rules are applied.
3. And finally, in the last part –called *grammatical adaptation*– necessary changes (insertion of elements, transformation of verbal tenses and modes...) in order to obtain grammatical sentences are made.

Recombination rules applied during the *pattern recombination* and the *grammatical adaptation* phases are only of the following five types:

1. *Insertion*, where some element or structure is inserted at some point of the string.
2. *Deletion*, where some elements of the structure are deleted.
3. *Transposition*, where some elements are shifted to another location in the string.
4. *Inversion*, where elements in the string can be reversed.
5. *Transformation*, where some elements change their form.

One of our goals in defining such formulae –even though not the most important one– is to do without the traditional grammatical division of complex sentences and to speak just of patterns that recombine some acceptable structures and give some other acceptable structures. Patterns defined, then, give up the usual terminology (relative sentences, noun sentences and the like) and are organized according the following criteria:

- *Variables used*. We define patterns by specifying the variables used. We specify the variables at hand when we start the application of the rules sequence (in) and the result obtained after the application of such a sequence (out).
- *Type of insertion*. Taking into account that in every pattern we start by having at hand two propositions, by type of insertion we mean the place –the beginning, middle, end– where the second proposition is situated with respect to the first one.
- *Rules sequence*. Sequence of rules applied in order to obtain the complex sentence from the two simple sentences considered.
- *Element introduced*. The word we introduce in order to generate a complex sentence from the recombination of two simple ones.

The following example shows what patterns look like. In this example, the four criteria above can be observed. Notice, also, that the three different parts mentioned above are distinguished: (1) pattern recognition (roman); (2) *pattern recombination* (italics); and (3) **grammatical adaptation** (boldface). Moreover, we consider that each of the simple sentences in the so-called *pattern recognition* phase belongs to a Corpus ($\mathcal{C}$).

By using the pattern in Table 1, sentences like *'The book which he read belonged to John'*, *'The umbrella which she took is mine'*, *'The report which Karen submitted implicated several of her friends'* or *'The books which he had recommended were unobtainable'* can be generated. In order to show how patterns work, we show how to generate the first sentence. The reader can easily

Pattern: (in: x, y, w, z, s; out: x WHICH w s y)
`Rules Sequence: Median Insertion, Transposition, Deletion, Insertion`
English: WHICH

xy = Prop. 1: Proposition $\in \mathcal{C}$
wz = Prop. 2: Proposition $\in \mathcal{C}$
x: Substantive $\in \mathcal{C}$
w: Substantive $\in \mathcal{C}$
y: Predicate $\in \mathcal{C}$
z = s x: Predicate $\in \mathcal{C}$
s: Verb $\in \mathcal{C}$

x (w s x) y: Quasi Proposition `Insert Prop.2 after` *x* `in Prop.1`
x (x w s) y: Quasi Proposition `Transpose` *x* `in Prop.2 to initial position in Prop.2`

x WHICH w s y: Proposition `Delete` *x* `in Prop.2 & Insert WHICH`

Table 1. Example of a Recombination Pattern.

check how to obtain the other three –and sentences of the same type– by using the same pattern. As shown in the first part of the pattern in Table 1, we start by identifying two simple sentences and by defining them in terms of basic categories as shown below:

- The book belonged to John:= x y
- He read the book:= w z; z = s x
  - The book:= x
  - belonged to John:= y
  - He:= w
  - read the book:= z
  - read:= s

Now, once we know the elements we have at hand, we start the phase of pattern recombination. In our example, we perform the two recombination rules –`Insert Prop.2 after` *x* `in Prop.1; Transpose` *x* `in Prop.2 to initial position in Prop.2`– indicated in the pattern above:

The book (he read the book) belonged to John:= x (w s x) y
The book (the book he read) belonged to John:= x (x w s) y

And finally, in the grammatical adaptation phase we introduce the elements required in order to obtain a grammatical sentence. By performing the rule indicated in the pattern –`Delete` *x* `in Prop.2 & Insert WHICH`– we obtain the following structure:

The book WHICH he read belonged to John:= x which w s y

The patterns we have defined show how to combine and modify simple sentences in order to generate complex ones. What is important in our formalism is that we can generate complex structures by using simple sentences and by applying only five types of very simple rules: *insertion, inversion, deletion, transposition* and *transformation*. This simplifies the generation of natural languages quite considerably. From the patterns defined it may not be difficult to define an algorithm that specifies the steps required to generate complex sentences from simple ones. We also have the intuition that such an algorithm may be language independent, since patterns defined for the three languages considered are very similar –they only have significant differences in the so-called grammatical adaptation phase. The definition of an algorithm of these characteristics may have important implications in the fields of natural language generation and language universals, since it shows that formal methods can be defined to generate natural languages independently of specific languages. Such a formalism may also have interesting implications in the areas of cognitive science or machine learning theory since –as pointed out by [1]– intuitively, maybe recombination rules are more feasible than rewriting systems to explain how humans process language.

4 Aggregation versus recombination patterns

The task of aggregation is to combine simple representations to form a complex one. Recombination patterns fit with this broad sense of aggregation understood as the 'combination' of any two linguistic structures to produce a third more 'complex' structure.

According to [11], regardless of the level of representation, aggregation is performed on linguistic structures (unless they are predominantly conceptual). Thus, theories of coordination and subordination are relevant. However, there is almost no mention of linguistic theory in the aggregation literature. There are several theories of coordination and subordination

in the linguistics literature which could be used, but this appears not to have been done. Instead, relatively simple and superficial approaches, especially in syntactic aggregation, have been adopted. Indeed, compared to the linguistic theories available, the treatments of coordination in particular implemented NLG systems have been relatively trivial. The conclusion we draw from this is that any successful system which achieves 'aggregated text' will have to incorporate linguistic knowledge about coordination and extraction, ellipsis, focus, centering and discourse and lexical semantics. Recombination patterns can offer a very simple linguistic theory of syntax that may contribute to the improvement of aggregation processes in NLG by incorporating a natural and implementable treatment of coordination and subordination.

In [12], Casper (Clause Aggregation in Sentence PlannER) is introduced as a sentence planner which focuses on generating concise sentences –an expression is more concise than another expression if it conveys the same amount of information in fewer words. Complex sentences combine clauses and are more concise than the corresponding simple sentences because multiple references to the recurring entities are removed. Clause aggregation can happen at three levels: 1) inferential, 2) rhetorical, and 3) linguistic. Here we are interested only in the linguistic level. At this level, lexical and syntactic information are used to combine clauses. We can distinguish the following two types of linguistic aggregation operators:

1. *Hypotactic operators*. The term hypotaxis describes the relation between a dependent element and its dominant clause. To aggregate two propositions using hypotactic operators, the propositions must share some entities in common. When they do, hypotactic operators try to transform one of the clauses into a modifier. Since the goal is to generate concise text, CASPER prefers to transform a proposition into an adjective if possible, then a PP, a participle clause, and if all else fails, a relative clause.
2. *Paratactic operators*. Paratactic aggregation operators combine clauses using constructions of equal status, such as coordination. Two approaches to combining propositions using coordinate constructions can be distinguished. In the first approach, adjacent propositions that have only one slot containing distinct elements are collapsed into one proposition with one conjoined slot containing the distinct elements. In the second approach, the conjoined propositions have distinct elements in more than one slot. To combine them, each conjoined proposition is generated, but

deletion rules are used to ensure the resulting sentence has the correct ellipsis.

If we consider the above two types of operators used in linguistic aggregation and we look at some examples of aggregation rules, we will realize the similarities between the technique used in Natural Language Generation and recombination patterns. Those similarities could support our idea of applying our framework to the field of aggregation. Moreover, we claim that recombination patterns provide a simple and natural linguistic theory of coordination and subordination that can be easily used in those syntactic aggregation processes.

We can distinguish between embedding and paratactic aggregation. Embedding is considered the most interesting and powerful type of aggregation. It can be used to express multiple facts about an entity concisely, and it is an effective tool in opportunistic generation. Three types of embedding are given in [15]: *nominal, adjectival*, and *adverbial*. Each type has a number of possible realizations with different complexities:

1. *Nominal*. A nominal can be embedded as a noun or an appositive phrase. For example, given *'King made this jewel'; 'King is a Scottish designer'*, the second proposition can be embedded as a noun: *'The Scottish designer King made this jewel'*, or as a appositive phrase: *'King, a Scottish designer, made this jewel'*.
2. *Adjectival*. An adjectival can be embedded as an adjective, a prepositional phrase or a relative clause. For example, given *'A man bought the picture'; 'The man had blond hair'*, the second proposition can be embedded as an adjective: *'A blond man bought the picture'*, as a prepositional phrase: *'A blond man with blond hair bought the picture'*, or as a relative clause: *'A man who had blond hair bought the picture'*.
3. *Adverbial*. An adverbial can be realized as an adverb or a prepositional phrase. For example, given *'Paula danced with Peter'; 'She was willing'*, the second proposition can be embedded as an adverb: *'Paula danced with Peter willingly'*, or as a prepositional phrase: *'Paula danced with Peter with willingness'*.

When many forms of embedding are possible for the same proposition, prefer the simplest one.

In [9] we have defined recombination patterns that can account in a very simple way for the three types of embedding referred in [15]. However, in

order to show the similarities between aggregation processes and recombination patterns we focus here on paratactic aggregation. Paratactic aggregation consists of conjoining two or more propositions with the help of a coordinating conjunction. Redundant information can be elided so that the conjunction can be found at any depth in the sentence structure. For example: *'John has a car'; 'John drives to school'* can be aggregated to *'John has a car and drives to school'*. At a deeper nesting level, *'John walks with Mary'; 'John walks with Jane'* can be aggregated to *'John walks with Mary and Jane'*.

In what follows we will use the following abbreviations: S = Subject; P = Predicate; Do = Direct object; Pc = Predicative Subject Complement; Conn = Connectives, i.e. And, Or, Xor; Xor = exclusive or. Moreover, we assume that a normal English clause has the following order *S P Do*. Two examples of aggregation rules are shown in Tables 2 and 3.

Definition of Predicate and Direct Object grouping (PDO-grouping):

$$S_1\mathbf{P}Do\ Conn\ S_2\mathbf{P}Do\ Conn\ S_n\mathbf{P}Do \longrightarrow S_1\ Conn\ S_2\ Conn...\ S_n\mathbf{P}Do$$

Table 2. Example of an Aggregation Rule: PDO-grouping.

Definition of Subject and Predicate grouping (SP-grouping):

$$\mathbf{SP}(Do_1\ Xor\ Pc_1)\ Conn\ \mathbf{SP}(Do_1\ Xor\ Pc_2)\ Conn...\ \mathbf{SP}(Do_n\ Xor\ Pc_n) \longrightarrow \mathbf{SP}(Do_1\ Xor\ Pc_1)\ Conn\ (Do_2\ Xor\ Pc_2)\ Conn\ (Do_n\ Xor\ Pc_n)$$

Table 3. Example of an Aggregation Rule: SP-grouping.

If we apply the aggregation rule in Table 2 to the following two sentences: 1) *'John is a student'* and 2) *'Mary is a student'*, the aggregated sentence

obtained is: *'John and Mary are students'*, where the grouped phrase is: *'are students'*.

Now, if we apply the aggregation rule in Table 3 to: *'John is a boy'* and *'John is tall'*, the result after aggregation with Subject and Predicate Grouping is : *'John is a boy and tall'*, where the grouped phrase is: *'John is'*.

If we compare the aggregation rules in Tables 2 and 3 with the recombination patterns in Tables 4 and 5, the similarities are evident.

PATTERN: AND `(Median Insertion)`

x y: Proposition $\in \mathcal{C}$
w y: Proposition $\in \mathcal{C}$
x: Substantive $\in \mathcal{C}$
w: Substantive $\in \mathcal{C}$

x (w y) y: Quasi Proposition `Insert Prop.2 after Subject in Prop.1`

x AND (w y) y: Quasi Proposition `Insert AND before Prop.2`
x AND w y: Proposition `Delete in Prop.2 elements present in Prop.1`

Table 4. Recombination Pattern: AND (Median Insertion).

By using the recombination pattern in Table 4, we can obtain an aggregated sentence like *'Gordon and Shirley missed the meeting'* by performing the steps established in the pattern as follows:

- Gordon missed the meeting:= x y
- Shirley missed the meeting:= w y

  Gordon:= x
  missed the meeting:= y
  Shirley:= w

  Gordon (Shirley missed the meeting) missed the meeting:= x (w y) y

  Gordon AND (Shirley missed the meeting) missed the meeting:= x and (w y) y

PATTERN: AND `(Final Insertion)`

x y: Proposition $\in \mathcal{C}$
x z: Proposition $\in \mathcal{C}$
x: Substantive $\in \mathcal{C}$

x y (x z): Quasi Proposition `Insert Prop.2 after Prop.1`

x y AND (x z): Quasi Proposition `Insert AND after Prop.1`
x y AND z: Proposition `Delete in Prop.2 elements present in Prop.1`

Table 5. Recombination Pattern: AND (Final Insertion).

Gordon AND Shirley missed the meeting:= x and w y

Now if we apply the recombination pattern in Table 5, we can obtain paratactic aggregations such as the one in *'Smith hit the ball and ran to first base'* by performing the rules stated in the pattern:

- Smith hit the ball:= x y
- Smith ran to first base:= x z

  Smith:= x
  hit the ball:= y
  ran to first base:= z

Smith hit the ball (Smith ran to first base):= x y (x z)

Smith hit the ball AND (Smith ran to first base):= x y and (x z)
Smith hit the ball AND ran to first base:= x y and z

What we have tried to show in this section is that recombination patterns and rules proposed in the field of aggregation are very similar. In fact, the same result can be obtained by applying an aggregation rule, such as the ones in Tables 2 and 3, or by applying a recombination pattern such as the ones defined in Tables 4 and 5. Therefore, what we claim here is that aggregation may be carried out by using recombination patterns, an integrative approach to natural syntax coming from formal languages, biology and linguistics. By offering a simple, natural and implementable new approach to

syntax, recombination patterns might provide the missing linguistic theory in most of the implemented aggregation systems.

5 NLG with recombination patterns

In this section we would like to suggest the possibility of applying recombination patterns to natural language generation. What we present here is just a preliminary intuition about this possibility, nothing else. A great deal of research should be done in this direction in order to actually define an NLG system based on the formal framework introduced in this paper.

In general it is accepted that in a Natural Language Generation System the following phases can be distinguished:

- *Content selection*: from the abundant knowledge base we select *what to say*. This phase packages information as verb-based, clause-sized propositions, each of which is realized as a single sentence. Many of these propositions share common features, such as the entity being described. If a generation system simply generates each proposition as a sentence, the output will contain many repetitive and redundant references to common features.
- *Text plan*: decides the order in which the sentences should be generated to make the text coherent.
- *Sentence planning:* its main task is aggregation –the combining of semantically related propositions in order to produce concise and fluent expressions. The goal or the sentence planner is to transform a set of input propositions into a minimum number of words under lexical, grammatical and pragmatic constraints. This transformation process occurs in multiple stages. In each stage, a set of combining operators is applied to the propositions.
- *Surface generation:* it has to be decided *how to say it*, i.e., the realization of the syntactic structures and lexical choices.

Now let us take a slightly simplified view of the text generation process as a pipeline of three stages: 1) *Text planning* (which determines the content and overall discourse structure of the text material), followed by 2) *sentence planning* (which decides on the sentence structure and scope), which in turn is followed by 3) *surface form realization* (which is based on syntax). So, the steps in a natural language generation system using aggregation techniques can be the ones in Figure 1.

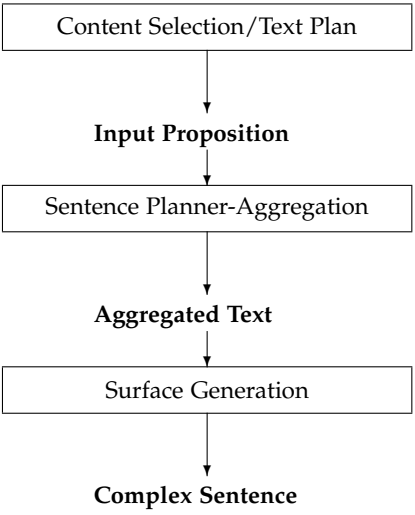

Fig. 1. NLG System with Aggregation.

If we apply our framework to the scheme of natural language generation steps in Figure 1, we obtain a completely equivalent NLG system that consists of the same steps specified with a recombination terminology as shown in Figure 2.

So, in our model we can say that aggregation is performed in the *pattern recombination* phase, whereas surface generation is equivalent to the *grammatical adaptation* phase in recombination patterns. Taking into account those equivalences, we could define an NLG system in which once the content of the text is determined we receive a text plan where:

1. we can recognize some patterns in the so-called *pattern recognition* phase;
2. we can apply *pattern recombination* by performing a task analogous to the one performed by the so-called *aggregation* technique –as a matter of fact, what we do in such a phase is to DELETE redundant elements and, therefore, to provide a more readable text (the same task performed by aggregation);
3. in the *grammatical adaptation* phase we perform a task analogous to *surface generation* in natural language generation mechanisms, since what

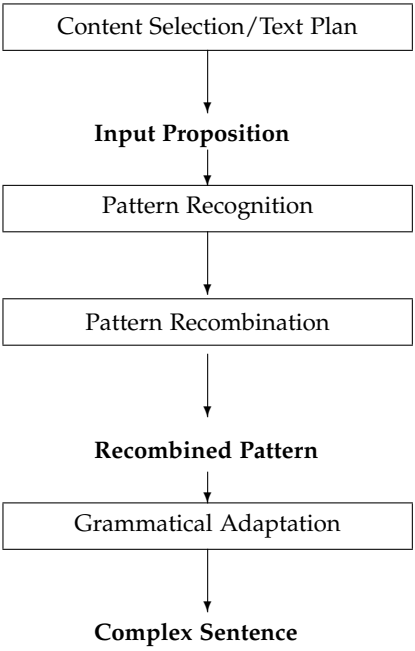

Fig. 2. NLG System with Recombination Patterns.

we do here is to make the necessary modifications in order to provide a grammatical sentence (agreement, tense modification, and so on).

6 Final remarks

In this paper we have made a preliminary introduction of a new framework for linguistics based on the behaviour of DNA molecules. Recombination patterns seem to be quite suitable for explaining in a completely new way some syntactic phenomena. We claim that they are a powerful and simple model that can be very useful to:

1. *Reconstruct syntax* with molecular methods.
2. *Formulate* some *systems* capable of generating the larger part of syntactic structures of language.

3. *Define a formalization* that can be implemented and may be able to describe and predict the behaviour of syntactic structures.

The considerable number of similarities between aggregation and recombination patterns has led us to suggest the possibility of carrying out aggregation by means of recombination. To the question 'why recombination patterns for aggregation processes?' we can answer by referring to the main features of the model introduced here:

- *Simplicity*:
  - A limited number of patterns are sufficient to explain every syntactic phenomenon.
  - It is possible to model syntax with just five operations.
  - It is based on cutting and pasting mechanisms, that are simpler and more efficient than rewriting.
- *Naturalness*: They reconstruct syntax with natural methods and therefore provide a natural model for natural languages.
- *Linguistic base*: They formulate systems to generate complex syntactic structures of natural language.
- *Computational Suitability*: They define a formalization of language that can be implemented.

References

1. G. Bel Enguix, *Molecular Computing Methods for Natural Language Syntax*. GRLMC Report, 18/01, Tarragona (2001).
2. H. Cheng & Ch. Mellish, Capturing the Interaction between Aggregation and Text Planning in Two Generation Systems. In *INLG'2000. Proceedings of the First International Conference on Natural Language Generation*. (2000).
3. H. Cheng, Ch. Mellish & M. O'Donnell, Aggregation Based on Text Structure for Descriptive Text Generation. In *Proceedings of the PhD Workshop on Natural Language Generation, 9th European Summer School in Logic, Language and Information (ESSLLI97)*. Aix-en-Provence (1997).
4. C.K. Chuah, Aggregation by Conflation of Quasi-Synonymous Units in Author Abstracting. In *TALN. 8e Conférence sur le Traitament Automatique des Langues Naturelles*. Tours (2001): 143-152.
5. H. Dalianis, Aggregation as a Subtask of Text and Sentence Planning. In J.H.Stewman (ed.): *Proceedings of Florida AI Research Symposium, FLAIRS-96*. Key West, Florida (1996): 1-5.

6. H. Dalianis, Aggregation, Formal Specification and Natural Language Generation. In *Proceedings of the NLDB'95, First International Workshop on the Application of Natural Language to Data Bases*. Versailles (1995): 135-149.
7. H. Dalianis, Aggregation in Natural Language Generation. *Journal of Computational Intelligence*, **15/4** (1999): 384-414.
8. H. Dalianis & E. Hovy, Aggregation in Natural Language Generation. In Adorni, G. and Zock, M. (eds.): *Trends in Natural Language Generation: an Artificial Intelligence Perspective, EWNLG'93, Fourth European Workshop*. LNAI 1036, Springer, Berlin (1996): 88-105.
9. M.D. Jiménez-López & V. Manca, Recombination Patterns for Natural Syntax. submitted.
10. E.J.S. Joanis, *Review of the Literature on Aggregation in Natural Language Generation*. Technical Report CSRG-398, Department of Computer Science, University of Toronto (1999).
11. M. Reape & Ch. Mellish, Just what is aggregation anyway? In *Proceedings of the 7th European Workshop on Natural Language Generation*. Toulouse (1999): 20–29.
12. J. Shaw, Clause Aggregation Using Linguistic Knowledge. In *Proceedings of the 9th International Workshop on Natural Language Generation*. Niagara-on-the-Lake, Canada (1998): 138-147.
13. J. Shaw, Segregatory Coordination and Ellipsis in Text Generation. In *Proceedings of the 36th Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*. Montreal (1998): 1220-1226.
14. J. Shaw & K. McKeown, An Architecture for Aggregation in Text Generation. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Nagoya (1997).
15. D. Scott & C. de Souza, Getting the Message Across in RST-based Text Generation. In *Current Research in Natural Language Generation*. Academic Press, London (1990).
16. J. Wilkinson, *Aggregation in Natural Language Generation: Another Look*. Technical Report, Computer Science Department, University of Waterloo (1995).