NLCA: Towards an algorithmic implementation

F.A. Grootjen

Computing Science Institute/

CSI-R9811 April 1998

Computing Science Institute Nijmegen
Faculty of Mathematics and Informatics
Catholic University of Nijmegen
Toernooiveld 1
6525 ED Nijmegen
The Netherlands

NLCA: Towards an algorithmic implementation

Franc Grootjen
University of Nijmegen, The Netherlands

Abstract

In most mainstream approaches to natural language modelling, some form of hierarchical structure (e.g. phrase structure) plays a central role. However, practical application of phrase structure-based parsers in natural language processing has enjoyed only limited success. One reason for this lies in the rigidity of hierarchical structure on the one hand, as opposed to the high flexibility of language use on the other. The relative lack of success of rule-based parsers has inspired a search for alternative methods, such as statistically based or lexicon-driven parsing.

In search for a solution the NLCA project[1] took one step back, and examined the nature of hierarchical structure in general, and phrase structure in particular. It looked for ways to derive hierarchical structure from input, and to incorporate it in a mathematically well-founded theory of knowledge representation ([Sar96]). The result ([KS98]) is an approach in which hierarchical structure is found as the yield of the interaction between different, inherent combinatorial properties of linguistic units. The model identifies three different basic relations that underlie these combinatorial properties, at a level of abstraction that, in principle, allows language-independent modelling and analysis. The structural analysis of input is mapped onto formal concepts in the sense of lattice theory [Wil82], and in this way creates a suitable environment for information retrieval.

This paper summarises the basic ideas of NCLA and presents a sketch of an algorithm that implements NLCA for the English language.

1 Introduction

Nowadays, most natural language modelling techniques are based on the part-whole paradigm: using a rule scheme they try to describe the whole as the sum of its parts. The focus in these models is on *construction*: they describe how (by some exhaustive description method[2]) bigger units are constructed in terms of sequences of other (smaller) constituents. The relations between constituents in a part-whole construction are of less importance: even when they are recognised they merely serve as a construction condition (for example: the constituent agreement relationship).

In general these methods seems to be successful. In particular for clear cut sentences and languages that support analysis in terms of hierarchical levels. But even in those languages linguistic phenomena can arise that are inherently difficult to describe, for example discontinuous structures and structural variation. In these cases there is another concept, called relation, that clearly overshadows (and even interferes with) the part-whole principle.

[1] The acronym NLCA stands for Natural Language Concept Analysis.

[2] Probably the most representative members of these class of models are those that use phrase structure as their key concept. A formal grammar (or a resembling description method) functions as rule base.

To solve these kind of problems one could either (1) cling to the part-whole principle and try to find a way to capture the phenomenon's essentials *within* the model, (2) recognise the model's deficiency and extend the model so that it is capable of handling the particular constructions, or (3) take a step back and look for a new paradigm, more basic than the part-whole principle, that can deal with these kind of problems.

Despite the fact that the first alternative formally can solve any description problem (because of the theoretical description power of the model), experience shows that this strategy eventually leads to very complex descriptions that grow beyond an effectively maintainable limit. The second alternative, extending the formalism to solve a particular problem, meddles with the foundation of the model, risking the loss of theoretical modelling power by making it too ornate. Finally, the third alternative suggests a search for a new modelling technique. A search that reveals the underlying nature of hierarchical structure in language.

This paper summarises the basic ideas of NLCA and explores its algorithmic aspects. It results in a stepwise development of an algorithm for the derivation of hierarchical structure.

2 NLCA, a relational model

As opposed to phrase structure (grammar) based description methods, NLCA is based on *relations.* These relations combine linguistic units and - in doing this - form the structure of the sentence. A relation is an instance of one of the three relation schemes: *major predication*, *minor predication* and *qualification* [KS98]. The relation schemes, each with their own characteristics, are embodiments of linguistic concepts and have a philosophical foundation [DS98].

- Major predication (MP) - the symmetric relation between a predicate and its argument(s). The predicate introduces an argument structure and incorporates its arguments into a single relation (e.g. the verb-argument(s) relation).
- Minor predication (mp) - the relation between a (minor) predicate and its argument. Unlike major predication, this relation is asymmetric: the predicate needs its argument, but not the other way round. In English this property coincides with the optionality of modification (e.g. the adjective-noun relation).
- Qualification (Q) - the relation between a qualifier and a core. The qualifier has no information content of its own: its purpose is to make the core more specific (e.g. the article-noun relation).

The three relation schemes may be applied recursively. Their sum uniquely characterises the input.

3 Towards an implementation for NLCA

We now elaborate on the model's basic principles. The underlying idea for this way of software development is twofold: first of all it minimises the chance of drastic adaptations (since basic principles are unlikely to change). Secondly, it maximises the algorithm's transparency.[3]

[3] The user should only be aware of the basic principles of the model; there is no need to know any implementation detail.

3.1 Principles

NLCA's algorithm analyses the input from left to right trying to find relations between morphemes, words and larger units. Morphological units stored in the lexicon are called *lexical items.* A *lexical unit* is defined as a lexical item, or a combination of related lexical items.

Greediness

Guided by its *linguistic combinatorial properties*, each lexical unit attempts to relate with the 'nearest' surrounding lexical unit(s) and in doing this, may create a new lexical unit.[4]

Note that the term 'nearest' is used to stress the assumed innate efficiency of language: if there is more than one candidate available, the closest will be used. Furthermore, the relating process is directed by the combinatorial properties of all lexical units involved in it. These properties determine (1) under what conditions a lexical unit forms a (2) specific relation with (3) a other lexical units. Note that the combinatorial properties of lexical *items* are given, whereas those of the lexical *units* are derived from their constituent relations. The classification of lexical units of similar combinatorial properties yields a number of so-called *lexical classes.* Since hierarchical structure depends solely on the 'behaviour' (combinatorial properties) of its lexical units, and since these properties are similar for all lexical units in the same class, we can define our abstract algorithm in terms of these classes.

Each relation scheme can be defined in terms of *basic relations.* A basic relation (or *link*) between two lexical units indicates

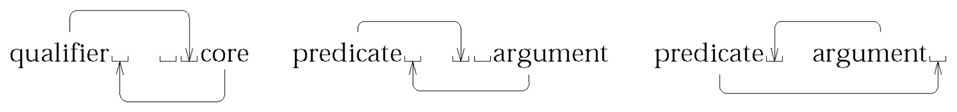
- the modification or qualification of a lexical unit by another lexical unit, or
- a lexical unit's need for an argument, as being fulfilled by some other lexical unit.

Since the first type of basic relations concerns internal properties of the modified (or qualified) lexical unit, we call them *internal links.* Likewise, relations of the second type are called *external links.* Because a lexical unit can be involved in several basic relations, we enhance our lexical units with internal and external argument positions to keep track of the relations. Formally, a lexical unit has two internal argument positions (indicating that the unit is subject to modification or qualification) and a number of external argument positions (expressing its combinatorial need).[5] To depict argument positions, we will use buckets like '␣' and arrows that point to these buckets to represent basic relations. The two buckets inserted *in front of* a lexical unit symbolise the modifying and qualifying internal argument positions (in that order), while the bucket(s) *behind* a lexical unit stand for the external argument position(s).

[4] However, this does not imply that the model is deterministic.

[5] Formally this means that we extend the set of lexical units L to $L' = L \times (\{q, m\} \bigcup S)$ where S is a finite subset of $\mathbf{N}^*$, denoting the numbered external arguments. In this way each instance of a basic relation is an element of the set $L \times L'$.

Using this notation we can represent the three relation schemes in terms of basic relations:



The use of argument positions is subject to certain restrictions. Modifying internal argument positions can be used by any number of minor predicates. Conversely, qualifying internal argument positions can only be filled a finite number of times, whereas external argument positions can (and should) only be filled once. This leads to the next principle (calling all the input a sentence):

Relatedness
In a syntactically correct sentence, each lexical unit is related to at least one other lexical unit and has its external argument positions filled.

According to this principle, a sentence with an unrelated lexical unit or with an unfilled external argument position is syntactically incorrect and can be rejected by the algorithm.

3.2 Adding more detail

The greediness principle claims that lexical units relate with their ‘nearest’ surrounding neighbours. We now make this ‘positive’ formulation more concise by introducing the term *invisibility*, and explain when lexical units *cannot* relate with others.

Sentence
A *sentence* is a finite sequence of lexical units. These units are numbered from 1 (the first unit) to n (the last unit). Formally one might see a sentence as a mapping s from $\{1,\ldots,n\}$ to L, the set of lexical units.

Invisibility
Let $s(i)$, $s(j)$ and $s(k)$ be three lexical units, with $i < j < k$ (or $k < j < i$). Let $s(j)$ and $s(k)$ be involved in one major predication or qualification.[6] All lexical units following (preceding) and including $s(j)$ are *invisible* to $s(i)$. The set of units visible to $s(i)$ is called its *visibility range.*

As suggested by the greediness principle, there are certain circumstances that cause the creation of a new lexical unit. In order to define the exact conditions for this to happen, we introduce some new terminology.

Linking
Two lexical units a and b are *directly linked* (notation $a \sim b$) iff there is a basic relation between a and b. Let $\overset{*}{\sim}$ be the transitive reflexive closure of $\sim$. The lexical units a and b are said to be *linked* iff $a \overset{*}{\sim} b$. Finally, the set

[6] This applies to English. In the NCLA model minor predication does not introduce a range for itself.

of all lexical units linked with a lexical unit x is denoted as $\langle x \rangle$.

Let a and b be two lexical units involved in a major predication or qualification relation. The process that creates a new lexical unit (and derives its linguistic combinatorial properties) from the set $\langle a \rangle$ $(= \langle b \rangle)$ is called *tentative concept formation* (TCF). As a result of this formation, the lexical units involved in $\langle a \rangle$ are replaced by the newly created unit. Note that this replacement may affect visibility.

Hypothesis: when the combinatorial need of a linguistic unit is fulfilled, TCF is triggered. However, the exact nature of concept formation, especially the question of what properties the newly formed unit does have, is a subject for further research.

4 Lexicon

Lexical units (and their linguistic properties) play a central role in NLCA. For each lexical unit the lexicon contains information. Since there is - at a certain level of accuracy - no need to (syntactically) discriminate between lexical units with the same combinatorial properties, the lexicon is ordered in classes.

It is important to distinguish between a morphological unit and a lexical unit: the first may be ambiguous, the latter (by definition) cannot. For example, the morphological unit 'man' may be mapped to a lexical unit denoting a human being, or to a lexical unit which will act as transitive verb. Note that this form of ambiguity only arises when two (semantically) different meanings of a morphological unit differ in their combinatorial properties. Obviously, this depends on the complexity (level of detail) of the lexicon.

Dictionary
The *dictionary* is a set of morphological units (D) and a function d from D to L that maps morphological units to lexical units stored in the lexicon. As we have seen, this mapping is not injective (due to the potential ambiguity of morphological units).

For each lexical unit, the lexicon contains information stating under what conditions the unit can create a specific relation and with which class of lexical units. If the lexical unit plays a role in major predication or qualification, the lexicon describes to which class the new lexical unit belongs to.

4.1 Building a lexicon

Since early childhood, every speaker of a natural language has acquired an enormous amount of (partly unconscious) linguistic knowledge. It is the task of the NLCA lexicon builder to capture and model this knowledge using a sufficiently powerful lexicon description language. At present, the project investigates possible description languages and tools, which should be (machine)translated to the so-called *basic lexicon description language* defined in the next section. The reason for defining such a basic description language is to obtain a minimal (but sufficient) implementable lexical base. Although the basic description language is not designed for building lexica 'by hand' we will use it to clarify the working of the algorithm.

4.2 Basic lexicon description language

A lexicon entry consist of a set of lines. Each line contains one or more rules of the form $r(c, d)$ and may be terminated by a single parameter n. The rule $r(c, d)$ states that the lexical unit is willing to create a type r relation (MP for major predication, mp for minor predication or Q for qualification) with another lexical unit of class c which is located to the d (`left` or `right`) side of it. Multiple rules on a single line should all be satisfied (in the given order). The parameter n (only present in the case of a major predication or qualification relation) refers to the class of the new tentative concept formed.

As mentioned in section 2, the qualifier of a qualification relation has no meaning independent of its core. The expectation of the core (in the case that the qualifier precedes the core) is modelled in NLCA by introducing a *proto item*, a lexical unit which is a placeholder for the core. As soon as the core is realised, it replaces the proto item.

In English we distinguish between *asymmetrical* qualifiers (e.g. 'the'), yielding a proto item that must be filled by its core, and *symmetrical* qualifiers (e.g. 'some') yielding a proto item that may function as an implicit core and need not be realised. In the lexicon this is indicated by '`symproto`' or '`asymproto`' for the d direction parameter. Note that a proto item belongs to a specific lexical class (the one supplied by the lexicon entry) and can only be filled by a lexical unit of the same class.

4.3 Simple lexicon

The table below illustrates a simple lexicon which will be used later on in the demonstration of the algorithm. For each (numbered) lexical class the table contains a label, some representative class members and a set of rules.

class	label	example(s)	rules	
1	singular noun	`girl`, `flower`		
2	definite article	`the`	`Q(1,asymproto)`	3
3	definite singular noun	`the girl`		
4	verb stem	`buy`, `find`		
5	past tense affix	$\texttt{ed}_1$	`Q(4,left)`	6
6	past tense	`bought`, `found`	`MP(3,left) MP(9,right)`	7
7	clause			
8	plural affix	$\texttt{s}_1$	`Q(1,left)`	9
9	plural noun	`flowers`		
10	det/head quantifier	`some`	`Q(9,symproto)`	9
11	adjective	`happy`, `sad`	`mp(1,right)`	

Although the lexicon rule entry for singular nouns is empty, this does not necessarily mean that there are no rules for singular nouns. It is possible that rules in other lexical entries induce rules for singular nouns. For example, the rule `mp(1,right)` for adjectives (class 11) induces an *implicit rule* for singular nouns: `mp(11,left)`.

5 Algorithm

5.1 Ambiguity and non-determinism

In NLCA there can be two reasons for ambiguity.[7] The first one corresponds to the occurrence of an ambiguous morphological unit, the second one refers to the fact that a lexical unit can satisfy more than one rule at the same time. An ambiguity may lead to multiple readings. The algorithm described in this paper uses non-determinism to handle ambiguity. An implementation can make use of more elaborated techniques (e.g. tabulation) to minimise parsing time.

5.2 Evaluation order

The algorithm evaluates the input from left to right. Since all possible relations are evaluated, the order of evaluation does not affect the outcome (there is always a finite number of relations possible). For the evaluation speed however, the left to right order is a plausible choice: while reading the input, the algorithm will relate (according to the greediness principle) as many lexical units as possible. Note that upon reading a lexical item, all relations the algorithm creates involve that item.

5.3 Sketch of algorithm

We now present the outline of the algorithm written in a PASCAL-like language. Comments are enclosed in curly braces. The non-deterministic behaviour of the algorithm is reflected by calls to the procedures `process` and `try`.

```
PROGRAM nlca(output);
BEGIN
  WHILE read(word)
  DO
    process(word) { non-deterministic call }
  OD;
  check wellformedness
END.

PROCEDURE process(word)
BEGIN
  IF word can fill proto item
  THEN fill proto item(word)
  ELSE try(word.rules) { non-deterministic call }
  FI
END.

PROCEDURE fill proto item
  IF related qualifier has unfilled external argument position
```

[7] The potential ambiguity due to the formation of tentative concepts is not considered in this paper.

```
  THEN make external link(proto,qualifier)
  FI
END.

PROCEDURE try(rule)
BEGIN
  IF rule implies proto item
  THEN create proto item;
       make qualifying internal link(qualifier,core);
       IF symmetric proto
       THEN make external link(core,qualifier)
       FI
  ELIF rule conditions are satisfied
  THEN apply(rule)
  FI
END.

PROCEDURE apply(rule)
BEGIN
  SWITCH relation type
  CASE minor predication:
    make modifying internal link(predicate,argument);
    make external link(argument,predicate);
    inherit external links(argument)
  CASE major predication:
    make external link(argument,predicate);
    make external link(predicate,argument)
  CASE qualification:
    make qualifying internal link(qualifier,core);
    make external link(core,qualifier)
END.

PROCEDURE make external link(src,dst)
  create external link(src,dst);
  IF dst is qualifier or major predication AND
     all external positions of dst are filled
  THEN install trigger
  FI
END.

PROCEDURE inherit external links(argument)
  FORALL qualifiers and minor predicates related to(argument)
  DO
    make external link(argument,qualifier)
  OD
END.

PROCEDURE trigger(unit,rule)
BEGIN
  form tentative concept(<unit>,class);
  process(concept)
END.
```

```
PROCEDURE check wellformedness
BEGIN
  FORALL lexical units
  DO
    IF unreferenced(lexical unit)
    THEN fail
    FI
  OD
  succeed
END

BOOL PROCEDURE unreferenced(lexical unit)
BEGIN
  IF lexical unit part of tentative concept
  THEN unreferenced(tentative concept)
  ELSE all its external arguments positions are filled
  FI
END
```

5.4 Example

Using the lexicon introduced in section 4.3, we exemplify the algorithm for the sentence:

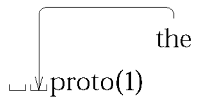
`the happy girl bought some flowers.`

We presume the existence of a morphological engine that rephrases this sentence into:

`the happy girl buy` $\texttt{ed}_1$ `some flower` $\texttt{s}_1$`.`

Traceback:

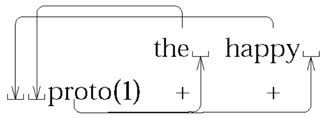
```
read word("the")
process("the") { lexical class=2 }
  IF word can fill proto item { no, there is no unfilled proto item }
  ELSE try(Q(1,proto) 3)
     IF rule implies proto item { yes }
     THEN create proto item { lexical class=1 }
          make qualifying internal link("the","proto")
          IF symmetric proto  { no }
```



After reading the first word of the sentence, the algorithm allocates a column for the qualifier '**the**' and a row for the generated proto item. Since '**the**' is a asymmetric qualifier, only one (internal) link is created.

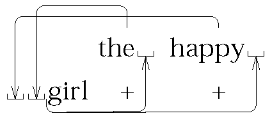
```
read word("happy")
process("happy") { lexical class=11 }
  IF word can fill proto item { no, lexical class 11 ≠ 1 }
  ELSE try(mp(1,right))
       IF rule implies proto item { no }
       ELIF rule conditions are satisfied { yes }
```

```
        THEN apply(mp(1,right))
               make modifying internal link("happy","proto");
               make external link("proto","happy");
                 IF dst is qualifier or major predication AND
                    all external positions are filled { no }
               inherit external links("proto")
                 make external link("proto","the");
                   IF dst is qualifier or major predication AND
                      all external positions are filled { yes }
                   THEN install trigger("the",Q(1,proto) 3)
```



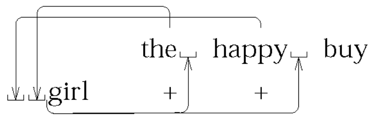
The occurrence of the adjective '**happy**' causes the algorithm to allocate a column for it. A minor predication is found between '**happy**' and the (still unfilled) proto item. Since the external argument position of '**the**' is filled (caused by inheritance from '**happy**') a trigger is installed that (when activated) forms the nominal adjective phrase '**the happy**'.

```
read word("girl")
process("girl") { lexical class=1 }
  IF word can fill proto item { yes, lexical class=1 }
  THEN fill proto item("girl")
         IF related qualifier has unfilled external argument position { no }
```



The proto item is replaced by '**girl**'.

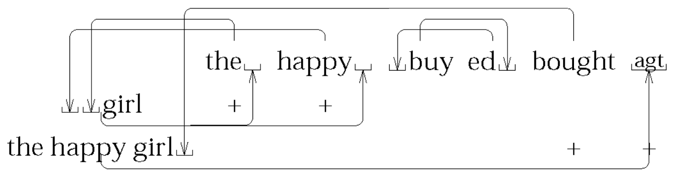
```
read word("buy")
process("buy") { lexical class=4 }
  IF word can fill proto item { no, there is no unfilled proto item }
  ELSE try(Q(5,right) 6) { implicit rule }
       IF rule implies proto item { no }
       ELIF rule conditions are satisfied { no }

-> trigger("the",Q(1,proto) 3)
     form tentative concept("the happy girl", 3)
     process("the happy girl")
     IF word can fill proto item { no }
     ELSE try(MP(6,right)) { implicit rule }
          IF rule implies proto item { no }
          ELIF rule conditions are satisfied { no }
```



the happy girl

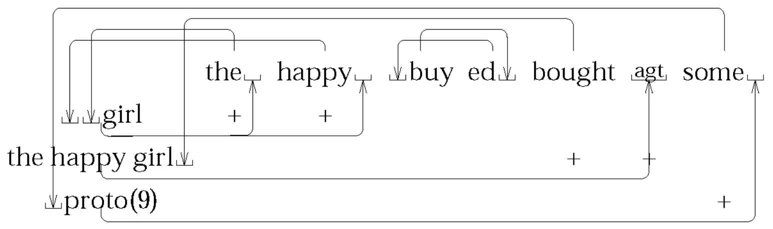
The algorithm reads '**buy**' and tries to relate it (without success). The previously installed trigger is activated and the tentative concept '**the happy girl**' is formed.

```
read word("ed$_1$")
process("ed$_1$") { lexical class=5 }
  IF word can fill proto item { no, there is no unfilled proto item }
  ELSE try(Q(4,left) 6)
       IF rule implies proto item { no }
       ELIF rule conditions are satisfied { yes }
       THEN apply(Q(4,left) 6)
            make qualifying internal link("ed$_1$", "buy");
            make external link("buy", "ed$_1$");
              IF dst is qualifier or major predication AND
                 all external positions are filled { yes }
              THEN install trigger("ed$_1$",Q(4,left) 6)

-> trigger("ed$_1$",Q(4,left) 6)
     form tentative concept("bought", 6)
     process("bought")
     IF word can fill proto item { no }
     ELSE try(MP(3,left))
          IF rule implies proto item { no }
          ELIF rule conditions are satisfied { yes }
          THEN apply(MP(3,left))
               make external link("the happy girl","bought (agent)");
                 IF dst is qualifier or major predication AND
                   all external positions are filled { no }
               make external link("bought","the happy girl");
                 IF dst is qualifier or major predication AND
                   all external positions are filled { no }
```



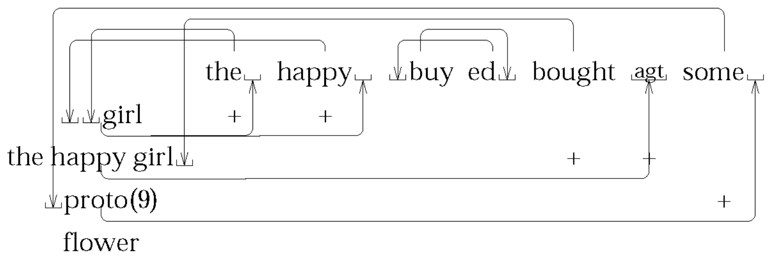
Upon reading the past tense suffix **ed**$_1$, a series of events happen. First the algorithm creates a column for it, finds the qualification with '**buy**' resulting (after triggering) in a new lexical unit '**bought**' which, in turn, gets its first external position (agent) filled by a major predication with '**the happy girl**'.

```
read word("some")
process("some") { lexical class=10 }
  IF word can fill proto item { no, there is no unfilled proto item }
  ELSE try(Q(9,proto) 9)
       IF rule implies proto item { yes }
       THEN create proto item { lexical class=9 }
            make qualifying internal link("some","proto");
            IF symmetric proto { yes }
            THEN make external link("proto","some");
                 IF dst is qualifier or major predication AND
                    all external positions of dst are filled { yes }
                 THEN install trigger("some",Q(9,proto) 9)
```



The lexical unit '`some`' creates a new column and a proto item. Since '`some`' is a symmetric qualifier, both the internal and the external links are set up, causing the installation of a trigger.

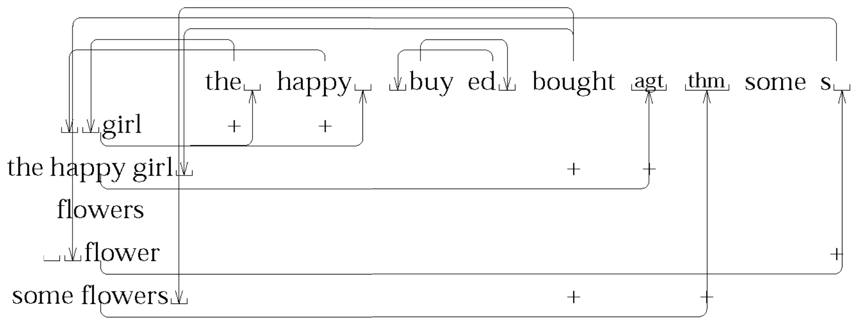
```
read word("flower")
process("flower") { *lexical class=1* }
  IF word can fill proto item { *no, lexical class 1 ≠ 9* }
  ELSE try(mp(11,left)) { *implicit rule* }
       IF rule implies proto item { *no* }
       ELIF rule conditions are satisfied { *no* }
  ELSE try(Q(8,right) 9) { *implicit rule* }
       IF rule implies proto item { *no* }
       ELIF rule conditions are satisfied { *no* }
  ELSE try(Q(2,left) 3) { *implicit rule* }
       IF rule implies proto item { *no* }
       ELIF rule conditions are satisfied { *no* }
```



A new row is created for '`flower`'. As yet, it cannot relate with other lexical units.

```
read word("s_1")
process("s_1") { *lexical class=8* }
  IF word can fill proto item { *no, lexical class 8 ≠ 9* }
  ELSE try(Q(1,left) 9)
       IF rule implies proto item { *no* }
       ELIF rule conditions are satisfied { *yes* }
       THEN apply(Q(1,left) 9)
            make qualifying internal link("s_1", "flower");
            make external link("flower", "s_1");
              IF dst is qualifier or major predication AND
                 all external positions are filled { *yes* }
              THEN install trigger("s_1",Q(1,left) 9)

-> trigger("s_1",Q(1,left) 9)
     form tentative concept("flowers", 9)
     process("flowers") { *lexical class=9* }
       IF word can fill proto item { *yes* }
```

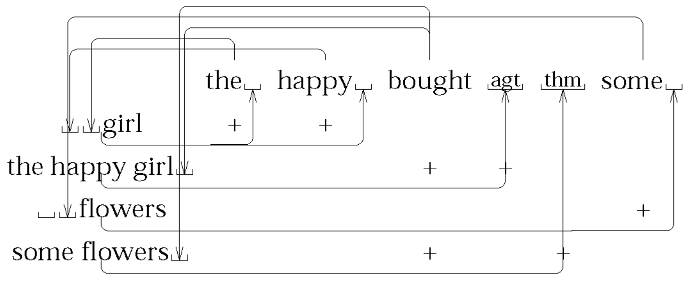
```
        THEN fill proto item("flowers")
             IF related qualifier has unfilled external argument position { no }

-> trigger("some",Q(9,proto) 9)
     form tentative concept("some flowers", 9)
     process("some flowers") { lexical class=9 }
        IF word can fill proto item { no }
        ELSE try(MP(6,left) 7) { implicit rule }
             IF rule implies proto item { no }
             ELIF rule conditions are satisfied { yes }
             THEN apply(MP(6,left) 7)
                  make external link("some flowers","bought (patient)");
                    IF dst is qualifier or major predication AND
                      all external positions are filled { yes }
                    THEN install trigger("MP(3,left) MP(9,right) 7")
                  make external link("bought","some flowers");
                    IF dst is qualifier or major predication AND
                      all external positions are filled { no }
```



After reading 's_1' a column is allocated. Subsequently, the qualification with `flower` is identified and the lexical unit '`flowers`' is formed, for which a row is allocated. This lexical unit fills the proto item which leads to the lexical unit '`some flowers`'. Now the second (theme) external argument position of '`bought`' is filled completing the major predication.

```
read word(".")
-> trigger("bought",MP(3,left) MP(9,right) 7)
     form tentative concept("the happy girl bought some flowers", 7)
```

Reading the end of sentence symbol will activate the last trigger and the complete clause is formed. In the last part, the algorithm checks wellformedness. Since all lexical units are part of the tentative concept forming the complete clause, the wellformedness condition holds trivially. Finally, the resulting analysis is as follows (morphologically realised relations are left out):



6 Summary and further research

We presented a non-deterministic algorithm for NLCA and a proposed data structure for the lexicon. A better support for the development of lexica and tools that provide a more powerful and user friendly description mechanism are the subjects of further research.

References

[DS98] G.J.Y. Debrock and J.J. Sarbo, "Towards a Peircean model of language", Technical Report CSI-R9802, University of Nijmegen, 1998.

[KS98] V. Kamphuis and J.J. Sarbo, "Natural Language Concept Analysis", in *Proc. of NeMLaP3/CoNLL98: International Conference on New Methods in Language Processing and Computational Natural Language Learning, ACL*, ed. by D.M.W. Powers, pp. 205-214, 1998. Sydney, Australia

[Sar96] J.J. Sarbo, "Lattice embedding", in *Conceptual Structures: Knowledge Representation as Interlingua (ICCS'96)*, ed. by P.W. Eklund and Gerard Ellis and Graham Mann, vol. 1115, pp. 293-307, 1996.

[Wil82] R. Wille, "Restructuring lattice theory: An approach based on hierarchies of concepts", in *Ordered sets*, ed. by I. Rival, pp. 445-470, D. Reidel Publishing Company, Dordrecht-Boston, 1982.