Mathematical Problems of Computer Science 41, 114---121, 2014.

An Automated System for Correction of Rules of Transformation from Universal Networking Language into Natural Language[1]

Aram A. Avetisyan and Levon R. Hakobyan

Institute for Informatics and Automation Problems of NAS RA
e-mail: a.avetisyan@undlfoundation.org, levon.r.hakobyan@gmail.com

Abstract

An approach to automated correction of UNL grammar rules is implemented. The development process of the rule sets could be made in a more effective way if some tools are used which can find the rules responsible for possible errors and make suggestions about its correction. Several definitions for the formalization of the transformation process and the algorithm for such kind of functionality are created. The implementation of the algorithm in the UNDL Platform and some examples are described.

Keywords: UNL, Machine translations, Transformation rules, Grammar.

1. Introduction

UNL (Universal Networking Language) is a meta-language representing semantic information. Its main purpose is to store "the meaning" of natural language texts in a language-independent format. Each sentence in UNL is a oriented linked graph [1,2,3].

Any sentence in UNL is a semantic network believed to be logically precise, humanly readable and computationally tractable. In the UNL approach, information conveyed by natural language is represented sentence by sentence, as a graph composed of a set of oriented binary labeled links (referred to as "relations") between nodes (the "Universal Words", or simply "UW"), which stand for concepts. UWs can also be annotated with "attributes" representing modality [1,2].

Today one of the main systems for the UNL processing is UNDL Platform. UNDL Platform is a web-based application which was developed by UNDL Foundation [3].

Transformation process from natural language to UNL is called analysis and from UNL to natural language - generation. In UNDL Platform the analyzer (i.e. algorithm which implements the analysis) was developed under the IAN project, and the generator (i.e. algorithm which implements the generation) - under the EUGENE project [3].

[1] This work was supported by State Committee of Science, MES RA, in frame of the research project SCS 13-B321

Transformation applications use dictionaries and grammar rule sets for the corresponding natural language.

Therefore, one of the main goals of the UNL project is the development of grammar rule sets. But the process of creating and editing such rules can be a big challenge for users.

In this article we introduce an approach to the automation of the rules correction process using real examples.

We introduce some new definitions, describe the corresponding algorithm for responsible rules searching and give basic information about its implementation in the UNDL Platform system.

2. The Problem Setting

Transformations in the UNDL Platform system are implemented by consequently applying transformation rules on the UNL sentence. In case of analyzer, the result of this process is transformation from the linear structure to the semantic graph. In case of generator, it is transformation from the semantic graph to the linear structure.

Transformation from UNL to natural language has the following input parameters: dictionary, UNL sentence and grammar rule set. There are two types of grammar rules: transformation rules (t-rules) and disambiguation rules (d-rules) [4].

Transformation rule is a pair (α, β), where the left side α is a condition statement, and the right side β is an action to be performed over α.

Disambiguation rule is a pair (α, P), where the left side α is a statement and the right side P is an integer from 0 to 255 that indicates the priority coefficient which is taken into account for the implementation of α.

Without loss of generality, let us assume that there are no d-rules, and attach priorities to the corresponding t-rules.

The transformation process consists of several iterations (steps). On each step one t-rule is applied. Rule applying process continues until there are no rules which can be applied to the sentence.

On each step, transformation algorithm loops through all t-rules. If a t-rule with a satisfying condition is found, that rule will be applied to the current sentence, i.e. an action of current t-rule will be applied to the current sentence.

When the algorithm finishes, the user gets a transformation result and trace-data. Trace-data contains information about all steps of the transformation process applied to the sentence. In fact, this is the description of the transformation process. But because of a huge amount of information, this description is not easy for the users to deal with.

If the user finds mistakes in the result of transformation, he has to look for a t-rule that caused the current mistake to appear in the result. This process is not really effective for the user. This approach can become even more ineffective if the user also wants to find dependencies between rules and to make a decision on how applying some rules affect on choosing of the rules on the next steps.

Therefore, one of the main goals of the UNL project is creating additional tools for users to help them for improving the grammar rule sets.

As we mentioned above, the result of the transformation from UNL to the natural language is a linear structure with nodes. Therefore, possible mistakes are contained in one or more nodes. And the user can mark the corresponding nodes whose transformation should be corrected.

Proceeding from the above, we can define our goal as the development of the corresponding functionality, to help us find the rules that caused the specified node's state.

To find an appropriate solution we should, at first, introduce some definitions about the rule's responsibility for the corresponding structure of the transformation result.

In this paper, we consider the method for a development of the corresponding functionality. We also introduce an algorithm for responsible rules searching and give basic information about its implementation in the UNDL Platform system.

Our approach is developed for transformations from UNL to natural language. But with some modifications it could be used also for transformations from natural language to UNL.

3. One Possible Solution of the Problem

As we mentioned above, during the transformation process from UNL to natural language, a transformation from the semantic graph to a linear graph takes place. As a result the user gets a linear graph which contains one node for each word in the sentence in the natural language. So every mistake is contained in nodes from the UNL sentence obtained during the transformation. This makes possible to mark nodes which contain mistakes.

We assume that the source UNL sentence and dictionaries are correct. Considering this we claim that the correctness of the transformation result totally depends on transformation rules.

3.1. Basic Definitions

At first, we should give some basic definitions of the roles of rules in the transformation process. As we have mentioned, the UNL sentence is a graph, so sub-graphs may be considered.

Rule is called *affected on current node or relation* if the current node or relation has been created or modified by applying the rule, or this rule’s application has triggered the creation or modification of the mentioned node or relation in future steps.

Rule is called *responsible for some sub-graph* if its application has resulted from the corresponding sub-graph appearance in the transformation process.

Rule is called *indirectly responsible for some sub-graph* if its application has triggered a rule responsible for this sub-graph in future steps.

Of course, several rules can be responsible for a single sub-graph.

Taking these definitions into consideration, we can rephrase our goal as: functionality for searching all rules responsible for current nodes.

3.2. Approach

Our approach consists of two parts: basic part and modules. Basic part is an algorithm that reviews the steps of the transformation process and marks affected steps. The modules part consists of semi-independent logical modules. In the current implementation, the module is a function. Each of these modules should be used for a specific type of errors. For example, there is a module for syntactic errors.

Basic algorithm reviews all transformation steps, marks some of them which have affected on current nodes, decides which module should be called and prepares data to be passed to that module depending on user input.

On the second stage the module starts processing. This module analyzes all rules marked as affected on current node by basic algorithm and marks the rules responsible for current nodes.

3.3. Implementation

In our implementation, during the transformation process the transformation function already saves some critical information about transformation steps. This feature lets us start analyzing steps immediately after the transformation process.

In UNDL Platform, the steps of the implemented transformation process are as follows: Function *match* is used to check, if the sentence satisfies the rule condition. It returns the object of the *MatchSet* class. This object is empty if there is no sub-graph which satisfies the current condition. If *match* function finds a sub-graph which satisfies the condition showing that this rule is applicable to the current sentence, it returns the object of the *MatchSet* class which contains a copy of the current sub-graph.

If there is a corresponding sub-graph, the rule applying takes place. Rule applying is implemented by the *execute* function. This function returns the object of the OutputSet class, which contains a copy of the sub-graph, obtained by rule applying.

In fact, the object of the *MatchSet* class contains data about nodes and relations of the sub-graph which satisfies the condition. And the object of the *OutputSet* class contains data about nodes and relations, which were obtained by rule applying.

We have developed the *Tracer* class, which stores all critical information for searching responsible rules.

On every step our algorithm saves *MatchSet* and *OutputSet* objects, the applied rule, and the current sentence obtained by rule applying, in the object of the *TraceNode* class.

The *Tracer* class contains a *Path* object which is implemented by the hash-table. In this hash-table the key is a number of applied rule, and its corresponding value is an object of the *TraceNode*. Step number and *TraceNode* object created on that step are added to the Path object on every step of the transformation process.

So, after transformation the *Tracer* object stores all information about transformation steps we need.

In order to find the affected rules, unique identifiers of the corresponding nodes are allocated. The *AnalyzePath* function is called with these identifiers as parameters.

The *AnalyzePath* function looks through all *TraceNode* objects that are stored in Path hash-table from the last to the first. *OutputSet* and *MatchSet* of the corresponding *TraceNode* are analyzed on every step.

In fact, a rule can be claimed as affected on node, if the identifier of that node is contained in *OutputSet* or *MatchSet*.

For every step with an affected rule, an object of *MarkedItem* class is being created. The *MarkedItem* stores the link to the current *TraceNode* and some additional information. The *MarkedItems* set is then passed to the corresponding module.

So, the basic algorithm finds all rules affected on current node. On the second stage, a module for deep error processing starts.

This module loops through all the steps marked by the basic algorithm and analyzes them. It also analyses *OutputSet* and *MatchSet*, but the processing logic depends on the error type.

In rules sets usually rules, which are used for some kind of sentence post processing, occur. They remove unused brackets and add whitespaces to sentences.

These rules applying usually does not have a semantic meaning and these rules applying does not have a significant impact on the NL sentence obtained by the transformation process. So, in our algorithm we have the ability to ignore such kind of post processing rules.

3.4. Computational Complexity

Computational complexity is very important for us, because UNDL Platform is a web-application and its computational time is bounded with the server response time. So we should show that the upper bound for computational time of our algorithm is appropriate for us.

If we take maximal count of all nodes and relations for all steps as M, and count of steps in transformation process as N, we claim that the estimated algorithm time is O(2(N * 2M)).

Indeed, the maximal count of steps analyzed on the current stage, for the first and second stages, is equal to N. And the maximal possible count of reviewed nodes and relations (taking into consideration that both OutputSet and MatchSet are reviewed) is equal to 2M. From here we get an upper bound for the current stage of the algorithm which equals to N * 2M. Therefore, the whole algorithm upper bound is equal to 2(N * 2M).

4. Examples

4.1 The Basic Algorithm

During our development and testing process we used the Test Drive corpora developed by UNDL Foundation [5]. We used the following sentence with the "PRE#1" identifier:

```
[S: PRE#1]
    {org}
       the book on the table
    {/org}
    {unl}
       plc(book:01.@def, table:02.@def.@on)
    {/unl}
[/S]
```

After transformation we get the "the book on the table" result (it is equal to the original). And it has the following linear representation:

```
<SHEAD>
  #L("the":06, " ":09)
  #L(" ":09, "book":01)
  #L("book":01, " ":0A)
  #L(" ":0A, "on":04)
  #L("on":04, " ":0B)
  #L(" ":0B, "the":08)
  #L("the":08, " ":0C)
  #L(" ":0C, "table":02.@on)
 <STAIL>
```

We mark the ":08" node to get all rules responsible for that node, and run our basic algorithm. As a result, we get this set of rules which are marked as responsible for the ":08" node.

When we get this information, the corresponding module can be called to deeply analyze the marked rules.

During this process two rules were ignored as post processing rules. They are shown in Table 2.

Table 1

Rule Id	Rule	Depth	MatchSet	OutputSet
781	NP(%x;%y):=(%y,+>BLK)(%x,+>BLK);	29	NP:03(table:02.@on, the:08)	#L:03(the:08, table:02.@on)
711	NS(%x;%y):=NP(%x;%y);	25	NS:03(table:02.@on, the:08)	NP:03(table:02.@on, the:08)
685	XP(%x,N;%y):=NS(%x;%y);	22	XP:03(table:02.@on, the:08)	NS:03(table:02.@on, the:08)
650	/[ACDIJNPV]S/(%x;%y,^spec):=XP(%x;%y,+spec);	16	NS:03(table:02.@on, the:08)	XP:03(table:02.@on, the:08)
625	(%x,N,@def):=(NS(%x,-@def;%y,[the],+LEX=D,+POS=ART),+LEX=N); book.@def > NS(book,the)	8	[table:02.@def.@on]	[sc:03(NS:03(table:02.@on, the:08))]

Table 2

Rule Id	Rule
816	((%x)(%y)):=(%x)(%y);
817	(%x,>BLK)(%y,^BLK,^PUT,^STAIL):=(%x,->BLK)(" ",+BLK)(%y);

4.2. An Error Specific Module

Here we consider an example of one simple module which could be a prototype for more complex modules creation.

We used the following sentence with the "VER#1" identifier from Test Drive corpora: "He arrived".

Let's assume we get the following transformation "He arrives". Here is an error in the last letter. User shows that error and inputs the right letter "d".

So the basic algorithm returns the following subset of steps to be analyzed by the module:

Table 3

Rule Id	Rule
419	(@past,^att=@past):=(-@past,+att=@past,+WHEN);
45	(%x,V,@past):=(%x,-@past,-VBL,+ATE=PAS);
166	(%x,^inflected,FLX):=(%x,!FLX,+inflected);

On the last step analyzing (the algorithm goes from the last to the first), the algorithm assumes that "s" was added on that step by comparing the MatchSet and the OutputSet. After that, the algorithm analyzes the rule and assumes that on the last step the "s" letter was added by applying "FLX" of the current node because of the "ATE=PAS" attribute. Therefore, this is the reason of current error. So this is the responsible rule.

After that, the algorithm starts to analyze the other rules from the subset in order to find indirectly responsible rules. Now the algorithm looks for the steps which are responsible for the "ATE=PAS" attribute of the current node and/or the MatchSet of the last overviewed step.

The algorithm assumes that the second step is indirectly responsible, because the "ATE=PAS" attribute was added on that step and the MatchSet of the last overviewed step is equal to the OuputSet of the current step.

The last step is analyzed by the same way. On the previous overviewed step the "ATE=PAS" attribute was added because of the "att=@past" attribute. So, on this step the algorithm looks for the reason of the "@past" attribute and the corresponding equality between the OutputSet of the current overviewed step and the MatchSet of the previous overviewed step.

After that, because there are no other steps, the algorithm assumes that there are two indirectly and one directly responsible rule as it is shown on Table 4.

Table 4

Rule Id	**Rule**	**Type of Responsibility**
419	(@past,^att=@past):=(-@past,+att=@past,+WHEN);	Indirectly
45	(%x,V,@past):=(%x,-@past,-VBL,+ATE=PAS);	Indirectly
166	(%x,^inflected,FLX):=(%x,!FLX,+inflected);	Directly

And finally we get recommendation to replace "PAS:=0>"s";" part of the corresponding dictionary entry attribute by "PAS:=0>"d";" string.

As we see, the modules for error processing entirely depend on the type of the error. In general, on every step a module should analyze the MatchSet and the OutputSet of the current step, the MatchSet of the previous step, the condition and the action of the rule and the attributes of the corresponding nodes.

The modules should be intelligent enough to be able to algorithmically analyze the structures and the goals of the rules.

5. Conclusion

Today, a development of the methods for generating of transformation rules is one of the important parts of the UNL project. The main goal of such methods is a creation of tools, which would be able to increase efficiency of the rules development process.

Several definitions for the formalization of some aspects of the transformation process have been introduced above. The algorithm for responsible rules searching and the structures, used in the current implementation under the UNDL Platform project, are described. Also the examples of an error processing module and the basic algorithm were given.

We propose here the first implementation of the mentioned method. It should be improved during testing and the analysis of new cases of errors processing.

Researches will be done in order to create new algorithms and improve the existing ones for more precise automation of the error analyzing process.

References

[1] H. Uchida, M. Zhu, and T. Della Senta, *A Gift for a Millennium*, IAS/UNU, 1999.
[2] R. Martins, *UNL Wordnet*, Mackenzie University, 2008.

[3] UNDL Foundation web site, [Online]. Available: http://www.undlfoundation.org/
[4] DeConverter Specification Version 2.6, UNL Center /UNDL Foundation 2002.
UNL Center, Enconverter Specifications, Version 3.3 Tokyo, 2001.
[5] Test Drive Corpus, UNDL Foundation, [Online]. Available:
http://www.unlweb.net/wiki/EUGENE#Test_drive

Submitted 14.12.2013, accepted 22. 02. 2014.

UNL լեզվից դեպի բնական լեզու տրանսֆորմացման կանոնների ուղղման ավտոմատացված համակարգ

Ա. Ավետիսյան և Լ. Հակոբյան

Ամփոփում

Հոդվածում նկարագրված է UNL լեզվի քերականության կանոնների ուղղման ավտոմատացված մեթոդը: Կանոնների մշակման գործընթացը կարող է ավելի արդյունավետ լինել, եթե օգտագործվեն համապատասխան գործիքներ, որոնք թույլ կտան գտնել կանոններ՝ պատասխանատու հնարավոր սխալների համար և ստանալ հանձնարարական դրանց շտկման հնարավոր մեթոդի վերաբերյալ: Հոդվածում բերված են որոշ սահմանումներ տրանսֆորմացման ընթացքը ֆորմալիզացնելու համար և նկարագրված է դրան համապատասխան ալգորիթմը: Հոդվածում նկարագրված է նաև համապատասխան ալգորիթմի իրականցումը UNDL Platform համակարգում և բերված են դրա կիրառման առանձին օրինակներ:

Автоматизированная система для корректирования правил трансформации с языка UNL на естественный язык

А. Аветисян и Л. Акопян

Аннотация

В статье описан метод автоматизированного корректирования правил UNL грамматики. Процесс разработки правил может быть более эффективным с использованием инструментов позволяющих найти правила, ответственных за возможные ошибки, и получить рекомендации о методе их исправления. В статье даны некоторые определения с целью формализации процесса трансформации и приведен соответствующий алгоритм. В статье также описаны реализация алгоритма в системе UNDL Platform и некоторые примеры его использования.