PersNLG: Natural Language Generation from Visual Input

Mojtaba Rohanian, Bahram Vazirnezhad, Kasra Ahmadvand, and Mohammad Bahrani

***Abstract*—Natural language generation (NLG) is one of the burgeoning areas of Computational Linguistics in which the primary concern is to automatically generate sentences in human languages. Based on the type of the input they receive, NLG systems can be divided into two categories: text-to-text and data-to-text. Systems of the first type are usually a part of machine translation systems. The latter type, which is the subject of the current research, deals with inputs other than raw text, such as databases, images, videos and so on. This project is an attempt to create an image-to-text system. This is the first research on data-to-text language generation for the Persian language.**

In the beginning, an extensive literature review was conducted and in the end of the first phase of the research, an overall NLG architecture was chosen to work with. The software was designed to be highly modular and follows a so-called pipeline architecture. As a higher priority was placed for the NLP part, in the prototyping phase the image processing aspect was chosen to be as simple as possible. Images were selected to be of simple geometric shapes and only a handful of colors were supported. However using the KNN clustering algorithm, the system was able to detect a wide range of color shades. The images were drawn in a vector graphics editor and saved in the SVG format. A parser was written to extract information from the SVG files and conduct further computation to get all the information necessary and pass it on to the relevant modules.

To convert images to text, first the information from the image recognition section combined with the binary relations between objects or groups of objects were recorded in a data structure. This formed the intermediate representation. In the next stage, for different parts of the planned text, corresponding text generators were designed. Generators receive relevant sections from the intermediate representation and each create a section of the output. Later, these outputs from different generators are put together to form a cohesive whole. These generators are based on context-free grammars and flexibly construct their grammar and lexicon every time they receive new input.

In the end, a collection of descriptions were created from a group of images and then the quality of the output was evaluated.

***Index Terms*— Natural Language Generation, SVG parser, data-to-text generation, context-free grammars, pipeline architecture**

I. Introduction

NATURAL language is the system of communication among human beings and the primary tool for expression of rational thought. One of the specific applications of this tool is description of visual phenomena. To describe any concept including the content of an image, human beings initially construct their thoughts in an intermediate stage and what comes to surface in the form of speech or written text is constructed from this deep structure. The theoretical basis for language production and the cognitive processes involved are well researched within Psycholinguistics (Carroll, 2008) and transformational grammar (Chomsky, 1965). In Computational Linguistics however, the focus is mostly on computational models that can generate natural language as human-like as possible. These models don't necessarily try to emulate cognitive processes. Programs that employ such models are called natural language generation (NLG) systems.

A. NLG systems

An NLG system is a program that outputs natural language, in the form of text or audio. Depending on the type of the inputs they receive, NLG systems could be categorized into text-to-text and data-to-text. The former is usually a part of machine translation systems. The latter, which is the subject of the present research, receives input in a non-textual format like images, tables, figures and so on and turns them into a description in natural language. To the best of our knowledge, there has been no attempt to create a data-to-text NLG system for the Persian language and our research is the first step in this direction.

B. Applications of NLG

The most commercially successful NLG systems are usually front-ends to data analytic systems that describe large data for corporations. From 2009 onward, a couple of startups have appeared that use NLG technology in this way. Two of the more famous ones are Arria NLG and Yseop (Woods, 2015).
Nevertheless there are many different areas where NLG systems have been employed. Among these are systems that automatically generate weather reports (Goldberg, Driedger, &

Mojtaba Rohanian is with Languages and Linguistics Center at Sharif University of Technology, Tehran, Iran (e-mail: rohanian@mehr.sharif.edu).

Bahram Vazirnezhad is with Languages and Linguistics Center at Sharif University of Technology, Tehran, Iran. (e-mail: bahram@sharif.edu).

Kasra Ahmadvand is a senior developer with Parspooyesh Fanavar Co., Tehran, Iran. (email: K.ahmadvand@parspooyesh.com)

Mohammad Bahrani is with Languages and Linguistics Center at Sharif University of Technology (email: bahrani@sharif.edu)

Kittredge, 1994), assist human writers (Sauper & Barzilay, 2009), help people suffering from speech disorders (Reiter & others, 2009) and visual impairment (Ferres et al, 2006) and many more.
Many of the algorithms and methods used in NLG have been applied to closely related fields of machine translation, summarization and dialogue systems.

C. *Statistical multi-modal systems*

Another application of NLG relevant to our current research is automatic labeling and description of images. In these systems, images are detected using statistical models that use tagged corpora. After the content of an image is ascertained, the system ranks a set of possible words or sentences that best match the image. As the output is chosen from a database, these kinds of systems could be categorized as applications of information retrieval (Aker & Gaizauskas, 2010). The generation process is normally very limited in these systems and usually only consists of templates or n-grams (Brants et al, 2007). As in this project we wanted to go beyond templates and at the same time avoid excessive data-driven approaches in the image detection phase, we stopped short of incorporating techniques from these projects and postpone it to later phases of the research when the NLG aspect has properly matured.

II. MOTIVATION AND RELEVANT WORK

Based on the type of input they receive, NLG systems could be categorized into text-to-text and data-to-text systems. The former is usually a part of a machine translation system (Dale & Reiter, 2000). The latter however, receives input in a form other than raw text and can act autonomously or as part of a bigger system.
As of yet, there has been no attempt at creating a data-to-text NLG system for the Persian language. The current research is the first step in this direction. Here we are concerned with the question of how to convert non-linguistic data to a linguistic representation. More specifically this project is an exercise in converting visual input to descriptions in the Persian language. The system created here could later be used as a baseline for other researchers trying to develop NLG systems for Persian.

Recently there has been considerable interest in automatic description of videos and images. For example, systems have been recently designed to detect moving targets (Baltaretu et al, 2015), describe pictographs (Sevens et al, 2015) and regular images (Farhadi, et al., 2010). These projects bend towards statistical analysis and place more importance on the detection phase.
In this project however, we are first and foremost interested in developing an NLG system heavy on the linguistic side. So our attention was diverted to the literature on language generation techniques and the general architecture of an NLG system.

A. *Architecture of an NLG system*

Although there are several different architectures proposed for an NLG system, the one that has been consistently used in recent publications and is specially suitable for a modular design is Ehud Reiter's pipeline architecture in which the generation process is broken down into three separate levels as is shown in Figure II-1.

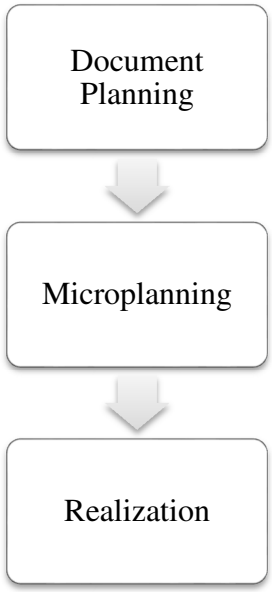


Figure II-1 Pipeline Architecture for an NLG system

The architecture is hierarchical, with the higher stage feeding the lower one its output. In other words, an output from the upper stage is the direct input for the lower (Dale & Reiter, 2000). This linear sequence of specialized autonomous modules form the basis of this architecture. It is important to note that this is just an abstract idea of how an NLG system should like, however, in reality often times the lines separating these stages are somewhat blurry and the architecture should be simply regarded as an ideal to look up to.
Document planning is the stage where we decide what exactly is going to be generated and in what format. This is arguably the most important stage because by the end of this process we should have already reached an abstract representation of what we want to express in natural language. This representation partitions the text in terms of its semantic content and shows the relationships among different portions of the text. The finer details such as the choice of words or length of sentences are dealt with in the microplanning phase.
As can be seen in Figure II-2 microplanning is comprised of several possible phases (Dale & Reiter, 2000):

- Aggregation controls the length of sentences. It combines similar sentences and prevents repetition of information.
- Lexicalization is the stage where we decide which words and syntactic structures should be uses. This could be dependent on the intended audience and variable like age.
- Referring expression generation controls the way entities in the text are referred to.

After microplanning comes the last stage where it all comes together in the form of natural language. The system decodes

all the information using a surface realizer. This could be done using templates or in a more sophisticated manner by incorporation of grammar-based methods.

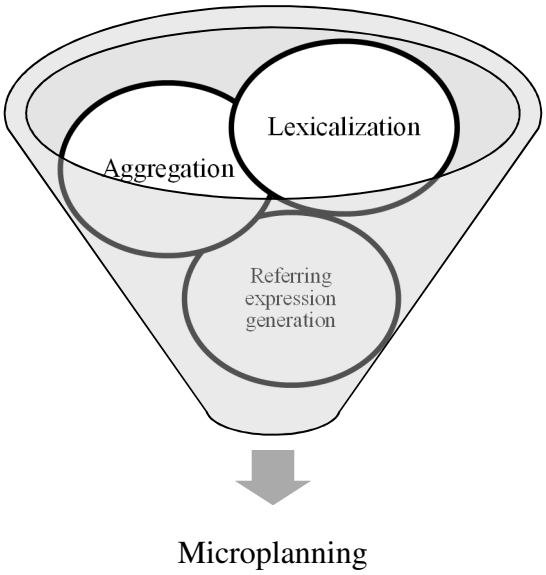


Figure II-2 Microplanning could be broken down to several sub-tasks

There are also statistical and hybrid methods which we will briefly explore. In the following section we take a closer look at each of the 3 modules in the pipeline architecture and describe some of the design choices with regards to creation of PersNLG.

III. Design choices for PersNLG

In section II.A we saw an overview of a typical NLG system that follows the pipeline architecture. In this section we explore these themes in more depth and explain some of the specific choices we made during the development of PersNLG.

A. *Document Planning*

Document planning is the stage where we design the overall shape of the text. Here we should know exactly what information is presented in what part of the text. Furthermore, the order in which the output sentences are presented can potentially alter the meaning of the text. That is why this stage is a crucial stage in the system. Traditionally there have been two different methods of conceiving discourse relations in NLG systems.

One sophisticated method is the use of Rhetorical Structure Theory (RST) in which the structure of a text is outlined using a hierarchical tree, in which each unit of the text is either a nucleus or a satellite (Mann & Thompson, 1988). The nuclei contain the most important information in the text and their omission would lead to the loss of coherence in the text. Satellites are parts of the text that are dependent on a nucleus and contain additional information about them. Deletion of satellites would not hurt the intended message of the text.

RST is a powerful tool for designing the structure of a text dynamically. In interactive NLG systems where the output could change depending on the input from the user, RST lets us plan the text differently if need be (Hovy, 1993). It gives us information about the role of each portion of the text and their interrelations so when and if the system fails to produce a section of the text, the system would be potentially aware of what to replace the failed output with because the existential reason of each part is known to the system.

While RST is an important theory and has found applications in other related fields such as dialogue systems and summarization, it's not always necessary to construct the output text every time, specially if we are dealing with a monologue rather than an interactive system (Hovy, 1993).

A simpler solution is the use of fixed schemas that act as the skeleton of the text and at the same time have enough flexibility to allow for recursive structures. This was first introduced in the TEXT system in the 1980's (McKeown, 1992). These schemas outlined a hierarchical structure of the text based on its semantic content. A constituent that could happen more than once ended with a + and optional parts were inside curly braces.

In PersNLG, we want the system to be able to describe images consisting of simple geometric shapes (square, rectangle, triangle, pentagon, and circle). We decided that the output text would have 3 different parts occurring in the following order:

(1) Identification (shape & number) +
(2) Absolute Positions +
(3) Relative Positions +

Figure II-3 Description schema for PersNLG

1. Identification of the objects and their respective colors
2. Absolute position of the objects on the plane
3. Relative positions of objects with regards to each other

If we are to write the general schema in the style of text, it would be the one depicted in Figure II-3.

In the following section we will discuss how each individual level was designed. The general outlines are taken from Dale and Reiter's classic book on NLG (Dale & Reiter, 2000).

B. *Microplanning*

Microplanning is an intermediate stage where we trim duplications and decide the fine details of the output text. Texts that have not gone through this stage would seem unnatural, vague, and/or of poor quality. The reason is that such a text would go against maxim of manner in the Grice cooperative principles.

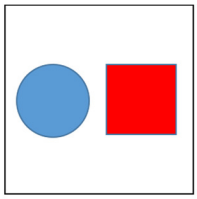
1) *Lexicalization*

This is where we decide about the exact words that should appear on the surface. For instance we know we know the concept that needs to be expressed is commanding the audience to sit down. This could be expressed in different ways. "Please be seated!" and "Sit down!" are both valid but have different connotations. The variables here are set

according to criteria such as age, sex and level of education of the audience.
In PersNLG this sub-module is kept as simple as possible for the moment to avoid excessive complexity. The audience is assumed to be an educated adult and except for a few places, the concepts are almost always mapped one-to-one to surface realizations.

2) Referring Expression Generation (REG)

This sub-module controls the way entities are referred to throughout the text. The proper way to address would be to first mention them using their full name and in later occurrences use a pronoun. This was noted as necessary for PersNLG. There are several instances where such a situation might occur. One situation is depicted at Figure III-1.



مربع قرمز و دایره آبی در وسط صفحه هستند. آن‌ها با هم هم‌ارتفاع هستند.

The red square and blue circle are in the middle of the page. They are of the same height.

Figure III-1 A possible occurrence of REG in PersNLG

REG was incorporated into PersNLG. When the relations between the two objects in the final module are the same as the ones immediately discussed in the absolute positions module, this triggers the activation of this process.

3) Aggregation

Aggregation is the most important stage of the microplanning module. During this process, parts of the text that could be directly or indirectly inferred from other parts are deleted. Figure III-2 shows different instances of aggregation in text. We will go over them one by one and see if they could show up in image descriptions of the type PersNLG is supposed to do.

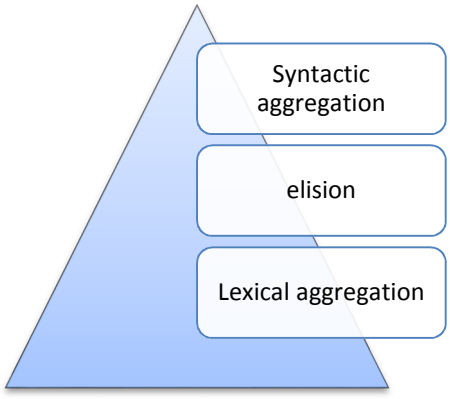


Figure III-2 Different kinds of Aggregation (Dalianis, 1999)

a) Syntactic Aggregation

This is the simplest type of aggregation to implement. It unifies different subjects/predicates of otherwise identical sentences. For instance sentences in (1) are aggregated to (2):

He is an athlete. He is a father. (1)
He is an athlete and a father. (2)

This can easily occur during the description of the objects in any parts of the general schema in Figure II-3, specially in the identification phase when different groups of objects are detected (Figure III-3).

سه دایره سبز در صفحه دیده می شود. دو مربع آبی در صفحه دیده می شود.

Two blue squares are visible in the picture. Three green circles are visible in the picture.

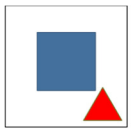
سه دایره سبز و دو مربع آبی در صفحه دیده می شود.

Two blue squares and three green circles are visible in the picture.

Figure III-3 Example of a possible syntactic aggregation in PersNLG

b) Elision

Elision is a type of aggregation in which certain parts of NPs and/or VPs in a sequence of sentences are deleted (Dalianis, 1999).



مربع آبی بزرگ در وسط تصویر است. مثلث قرمز در گوشه پایین سمت راست تصویر است.

A big blue square is in the middle of the page. A red triangle is at the bottom right corner of the page.

مربع آبی بزرگ در وسط تصویر و مثلث قرمز در گوشه پایین سمت راست تصویر است.

A big blue square is in the middle of the page and a red triangle at the bottom right corner of the page.

مربع آبی بزرگ در وسط و مثلث قرمز در گوشه پایین سمت راست تصویر است.

A big blue square is in the middle and a red triangle at the bottom right corner of the page.

Figure III-4 Two stage aggregation with elision of NP and VP

To give a relevant example, at Figure III-4 we see two descriptive sentences. In the first stage the duplicated VP is trimmed and using a conjunction the two sentences are unified. In the next, the duplicated NP is cut off. As you see, this type of aggregation reduces the number of sentences.
Incorporating this into PersNLG needed some care, because we had to know exactly what to delete and where to insert the conjunction. This needs some syntactic analysis.

مربع آبی بزرگ در وسط صفحه است. (3)

A big blue square is in the middle of the page.

For instance in (3) the sentence has the structure outlined in Figure III-5.

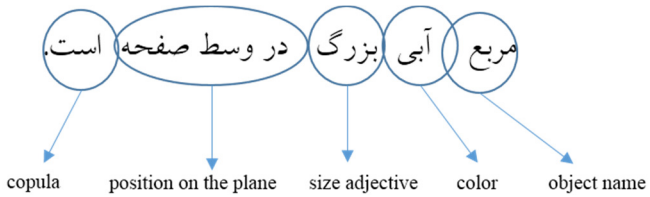


Figure III-5 structural analysis of the example (3)

Zooming in further, the structure of "position on the plane" would be:

Prep. + Position + "pagelimage" + copula (4)

So the elision process in Figure III-4 could be achieved by processing the first sentence, eliminating the last two units in (4) which consist of removing the copula and the fixed string that precedes it. The conjunction then is inserted in the exact same place connecting the sentences together.

c) Lexical Aggregation

In lexical aggregation, a set of items is replaced with a new item (Dalianis, 1999). To implement this, we need to have access to a knowledge-base beyond the text itself. For instance if we have sentence like (5), understanding that a week consists of 7 specific days we could aggregate the information, resulting in fewer words in (6).

It is going to rain on Saturday, Sunday, Monday, Tuesday, and Wednesday. (5)

It is going to rain on weekdays except Thursday and Friday. (6)

This is called *bounded lexical aggregation*. Here the information is intact and the original form is retrievable. In another type of lexical aggregation called *unbounded lexical aggregation* the information maybe reworded in a way that would lead to partial loss of data and therefore its use is only recommended when the information deleted is not critical. An example of this is seen at (7) and (8).

I had peanut butter sandwich for lunch. I drank a little soda too. (7)

I had something light for lunch. (8)

There are cases in PersNLG where lexical aggregation could theoretically be employed. For example suppose we have a situation where there are three objects in the picture and we have already constructed a set of NPs aggregated by conjunctions as in (9).

مربع سبز و مثلث آبی و مستطیل قرمز در وسط صفحه هستند. (9)

A green square, a blue triangle, and a red rectangle are in the middle of the page.

We know that these 3 objects include everything on the plane and that all of these are situated in a common area. So equipped with this information (9) can be shortened to (10).

همه اشکال در وسط صفحه هستند. (10)

All the shapes are in the middle of the page.

Incorporating this was rather easy in PersNLG because the information about the possible objects and their attributes are recorded in the program.

C. Surface Realization

This is the stage when it all comes together and natural language is finally produced. It is the final stage of the NLG system. There are several different methods for surface realization. We briefly introduce some of the more popular ones.

1) Template-based generation

This is the simplest form of surface realization. It is best suited for form filling and for generating generic sentences with little variability.

It is fast and creates predicable output. Different templates could be combined together to expand the output. However it is not as flexible as other methods and requires a large database of templates should we decide to cover more syntactic forms and support a larger vocabulary. In PersNLG we started with a very limited vocabulary (5 objects, around 10 colors, 5 possible areas on the plane and a few adjectives and adverbs) and there was no need to cover a lot of different syntactic forms. That is why template-based generation was the method we chose for initial prototyping.

2) Grammar-based generation

This is the most linguistically rich method for language generation. Here we need to write grammars and have access to a separate lexicon. Based on these two we generate sentences from an intermediate sentence specification.

Depending on the level of control we need on the details of the sentences, this process could be easy or difficult. There are tool-boxes like SimpleNLG that let the user specify English sentences in an easy straightforward manner. There are also more complex tools like FUF/Surge that use feature-based grammars and let the user control finer details of the output but are harder to use. We will briefly introduce these here as well. These are only designed for English and few other European languages and adapting them to Persian was beyond the scope of this research.

a) FUF/Surge

FUF/Surge is a surface realizer that uses a grammar-based approach for language generation. It has a database of hand-written unification-based grammar rules called *Surge* and a generator called *FUF*. The grammar and the lexicon are both stored as matrices called functional descriptions (FDs) that contain syntactic labels describing each constituent. The lexicon and intended grammar are combined in a process called unification and then a linearizer takes the unified FDs and converts them to sentences in natural language. FF/Surge is powerful but slow and requires a lot of human effort for grammar writing and maintenance (Zhong, 2009).

b) SimpleNLG

SimpleNLG is another grammar-based surface realizer for English (Gatt & Reiter, 2009). It has gained considerable attention since its introduction in 2009. It is faster and simpler than FUF/Surge but supports a more limited range of syntactic forms (Zhong, 2009). It is a Java library that lets the user define the structure of a sentence using objects and methods of the langue. For instance to create the sentence "Mary chases the monkey." we need to first create the phrase object and then specify the subject, verb and the object. SimpleNLG then constructs the correct sentences based on its set of grammatical and morphological rules.

```
SPhraseSpec p = nlgFactory.createClause();
p.setSubject("Mary");
p.setVerb("chase");
p.setObject("the monkey");
```

There are no libraries to be used for automating the surface realization phase for Persian. Also, for the reason that will be explored later in section V, creating a feature-based generator for Persian was met with practical difficulties and implementation of a feature-based realizer based on functional unification was postponed to later stages of the research.
Meanwhile an equally powerful (albeit more verbose) alternative was to use detailed context-free grammars and generators. With the in-built support for CFGs in NLTK, this turned out to be a very robust and effective strategy.

3) Two-stage and classification-based methods

These realization methods are based on statistical reasoning and require corpora. In the two-stage method, several candidate sentences are created using a hand-written or statistically derived grammar and then using a language model, the results are ranked. In other words, the system first generates a set of possible sentences (i.e. overgenerates) and then undesirable results are weeded out with a statistical filter.
Classification-based methods split the generation process into separate tasks and treats each one like a classification problem. The classification tasks could be as detailed as "negating the main verb" or "setting the tense of the verb to PAST". Feature selection is difficult here and putting the outputs of the classifiers in correct order is also challenging. However very high speeds can be achieved using this method (Zhong, 2009).
At this stage of the research we are less enthusiastic about statistical methods and are more interested in the problem of correct rule-based conversion of logical representations to natural language. That is why we stopped short of experimenting with these methods here.

Table III-1 summarizes the main points about different approaches in surface realization.

Table III-1 Comparison of four different approaches to surface realization (Zhong, 2009)

Approach	Advantages	Disadvantages
Grammar-based	High quality generation, Domain independent	Difficult to maintain, Grammar writing requires expertise
Template-based	Fast, Predictable output Ease of generation	Difficult to adapt to new domains, Needs large number of templates
Two-stage	Less human effort needed, Simple grammar, Adaptable to new domains	Over generation, Additional step needed
Classification-based	Little human effort needed, Adaptable to new domains	Annotated corpus needed

D. Representing semantic relations

When we are reasoning about object relations and controlling the content of what is going to be produced, we need to represent all the information including object features, their relative and absolute positions using an abstract representation. In other words, all our primary or derived knowledge about the image should be encoded using an abstract specification. This intermediate representation gets converted to natural language in the final stage. Figure III-6 shows a sample image with its possible intermediate semantic representation.
There are several possible ways to create a semantic network that would accommodate all object relations.

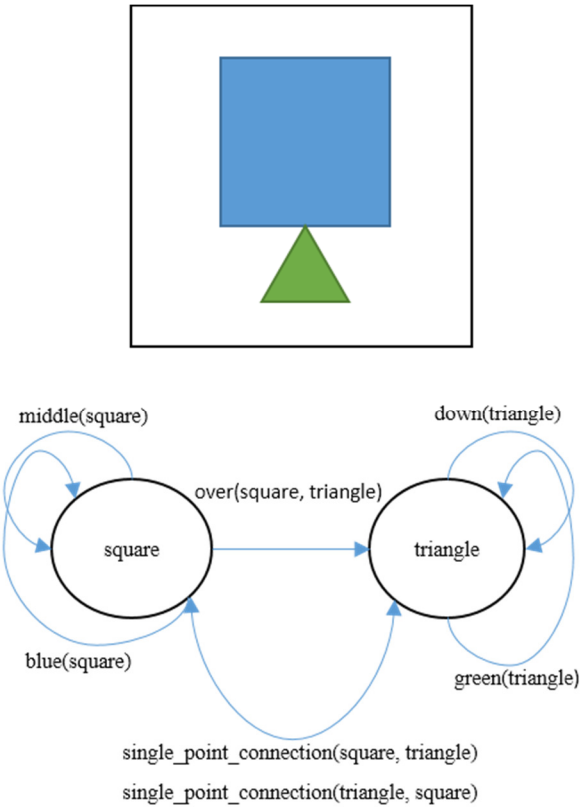


Figure III-6 An example of a possible semantic network for a visual input in PersNLG

One way to do this is to use the XML-based RDF that uses a graph-based data model to represent relationships between concepts. It consists of triples of (subject, predicate, object) as is shown in Figure III-7. RDF uses a query language called SPARQL.

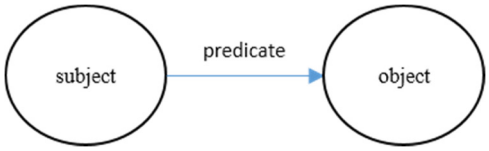


Figure III-7triples in RDF

There are other tools from the field of knowledge representation and ontology that could be used. One famous example is protégé.
Although these methods seem attractive, but for our purposes here, we decided not to use them. The reason is, all the possible object relations and features can be easily stored in the Python

data structure *dict* using key-value pairs and when the number of relations increase, we can save them as json files and load them very fast.
For example suppose we want to express the relation of *inclusion* between two objects. we can easily do that with (11). Now this is a one-directional binary rule so we have to use a tuple as the value. If the relation would go both ways, we could use a set as the value of the dictionary.

```
relation = {'inclusion':('circle', 'triangle')}
```
(11)

IV. Prototyping Based on Templates

With the design choices explained so far, we set out to build a prototype of our NLG system using templates in the realization phase. As we will explain later in section V, in the revision phase we enhanced our realization by utilizing context-free grammars.
The first step in the prototyping phase was the problem of image processing and object detection. This stage was dealt with in a way that would let us quickly get to the NLG parts instead of getting bogged down in the details of image processing. The solution we came up with was to use vector graphics and the open source software InkScape.

A. Scalable Vector Graphics (SVG) and Inkscape

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics with support for interactivity and animation. SVG is developed by W3C and is supported by major web browsers. It is possible to draw fairly complex images with SVG. We can either specify images by writing down all the details similar to CSS, or we can simply draw the images in an editor like Inkscape and save the file in the SVG format.
XML is a complex standard and it gives little direct information about its depicted objects. For simple images like rectangles and circles though, it has separate labels and coordinates and color codes are easily accessible. For instance (12) shows a portion of an SVG file describing a square. Note that there is only one label for square and rectangle in SVG and we understand the difference by looking at the difference between the numbers for height and width. The colors are specified using hexadecimal values.

```
<rect
   y="672.36218"
   x="239.99998"
   height="231.42857"
   width="231.42857"
   id="rect4140"
```
(12)

We decided we only wanted to have five types of shapes: rectangle, square, circle, triangle and pentagon.
The first three objects are easy to read in an SVG file but things tend to get messier when it comes to any other polygon. Triangles and polygons are both labelled as *path*s. For instance, this is how a triangle can be represented in SVG:

```
<path
    d="M 562.85716,900.93363
212.65938,828.83469 450.19778,561.60399 Z"
```
(13)

To understand this we also need to know a little bit about the coordinate system in SVG. The upper left corner of the frame is the starting point namely (0, 0). Moving to the right increases the values on the x-axis and moving down, counter-intuitively increases the values on the y-axis. In (13) the letter M which stands for Moveto specifies the start of the path which is the coordinates (`562.85716,900.93363`). The next pair of numbers specify the changes in the x and y axis till we reach the next corner of the triangle. So in order to compute the coordinates for the second corner we need to add the new values to the coordinates of the starting point. The same thing gets repeated to get the third corner. Z triggers the end of the path.
A similar method is used to represent pentagons.

B. Color detection in SVG

As we noted in section IV.A, colors of the individual objects are shown using hexadecimal values. We only wanted to support around ten of the more basic colors, so we could have simply written a hash table to map color codes to their respective names. The problem with this approach is that, a single color like *green* has a long continuum of possible shades and our program is only able to detect one or a small number of them.
To circumvent this problem, we wrote a function to convert all hexadecimal values to RGB color codes. Then we assembled together around 100 RGB codes (several shades from each color) and constructed a hash table with their names. Then whenever we receive a new hexadecimal color code from Inkscape, it first gets converted to RGB and if the code is not found in the keys of the table, the RGB triple along with the rest of the color codes in the table are mapped on a 3D space and using KNN clustering algorithm (with K = 1), the nearest neighbor to the new RGB code is chosen and its value in the hash table is given as a color name to the new code. In this way we succeeded in covering a very large range of possibilities.

C. Implementation of an SVG parser

Based on the information about SVG's coordinate system, labels and general structure, a parser was written to extract necessary information. The parser does further computation to derive object coordinates and color names. The output for the parser would be a list of dictionaries. These dictionaries contain keys like object name, color and coordinates.

D. Extracting relations for each portion of the output text

As was discussed in section III.A and demonstrated schematically in Figure II-3, the output text is intended to consist of three separate parts. In the first part the objects and their colors are mentioned. Next comes the positions of the shapes on the plane. In the last section, spatial relations of the objects are discussed. For this, we need three separate processes that act semi-autonomously. We call them modules. Each module cares about a portion of the semantic network and constructs its own textual output based on that. In the end, all of these pieces are put together to form a cohesive whole.
In the following sections we describe how each of these modules were conceived.

1) Identification

Supposing that in a sample image, there are n different objects that belong to k clusters, we can construct several templates for this section based on the information from the parser. Three types of templates were implemented. One of them is presented here in the Figure IV-1 and the other is very similar to this. An example that such a template could produce is given at (14).

$<num_1><object_1><color_1>$ *conj* $<num_2><object_2><color_2>$ *conj* ... *conj* $<num_k><object_k><color_k>$*<canned-text>*

Figure IV-1A template for the identification part

دو مربع قرمز و سه دایره سبز و یک مستطیل سفید در تصویر دیده می‌شود. (14)

2 red squares and 3 green circles and a white rectangle are seen in the picture.

Each module including the identification has a function called combiner. The combiner function creates the object clusters based on commonalities in color, shape or a combination of both.

Post-processing using regular expressions is needed to edit the conjunctions and insert commas before the last cluster.

In another type of template for identification phase, the text contains the relation of contrast. For instance, when there are 5 instances of one cluster and *only* 1 instance of the other, we can have a template in which the adjective تنها (only) is inserted after the conjunction.

2) Absolute Positions

Here, we need to create several sentences where objects are mapped to an area on the plane. Before we proceed to the demonstration of the template, it is necessary to see how the plane is partitioned and what algorithms were used to assign a partition to an object.

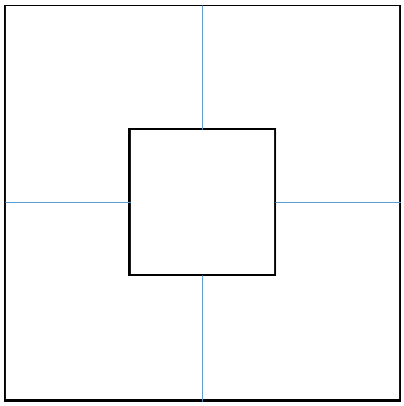


Figure IV-3 The plane is partitioned into 5 areas

As is seen in Figure IV-3, we have partitioned the plane into 5 areas. The task here is to assign a shape to one of these areas. This is a challenge because the object could be so big that could stretch across several areas. The solution was to compute the centroid for the shape and then checked to see if the centroid is situated inside any of the areas. Finding the centroid for the simple shapes we have is trivial and even in the case of pentagon it is computable by simple formulas averaging the coordinates. Determining whether the centroid is inside an area however is a little more challenging. Here we used the ray casting algorithm. We simply count how many times a ray, starting from a point and going in any fixed direction, intersects the edges of the polygon. If the number is even, the point lies outside the polygon and if the number is odd, it lies inside. Because we have all the coordinates for each shape, we can write equations for all of the lines of a shape. Supposing that the centroid is shown as C and its height is Y_C the line y = Y_C is the equation of the ray passing through the area, if this line intersects the lines of the shapes in an odd number of times, we realize that the centroid is situated inside an area. Otherwise we repeat the process with another area until we find where the centroid lies in.

After discovering the absolute position of the shapes, it is easy to construct sentences using templates if the kind shown in Figure IV-2.

$<object_i><color_i>$ *<prep.>* $<position_i>$ *<copula>*. +

Figure IV-2 A template for absolute positions

Here we also employ a combiner function to group shapes into clusters based on their common positions. This way we can aggregate sentences according to what was discussed in section III.B.3)a)3.

3) Relative Positions

This is the module where we try to find spatial relations among objects and mention them in the output. Several different relations were conceived.

a) Connection in a single point

In this relation, the two objects share one pair of coordinates.

b) Same height

The centroids of the two shapes have a common y coordinate.

c) Contrast in size and in color

We have defined certain pairs of colors to be contrasts of one another. Also, if the area of one object is more than 2 times of another, they contrast in size. One thing to note here is that, contrast is an element that is dependent on the perception of the viewer as well. For instance, two objects might indeed be in such a relation, but the viewer wouldn't perceive them as such because they are either too small or too big and the relation is not obvious. That is why, the relation is overlooked in such borderline cases.

d) Encirclement

When an object is entirely within the confines of another, this relation holds. They way to ascertain this is to check and see if

the points of the smaller shape are all indeed situated inside the bigger one. Here we used the ray casting algorithm again.

e) Relative distances

The distance between two objects is determined by the minimum distance among all their points. A distance between two objects might be longer or shorter than the distance between two other shapes. This is shown in an example in Figure IV-4.

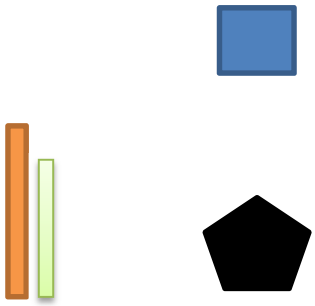


Figure IV-4 The distance between the pentagon and the square is longer than the distance between the rectangles

Similar to the two previous modules, here too we constructed relevant templates to account for different possible relations. It is important to note that in each of these modules, we first find new relations and add them to our general knowledge of the picture (according to the format specified in III.D). The combiner function in each module is tasked with selecting the necessary information from the semantic network. Only then we proceed to surface realization with templates.

V. Revision of the Surface Realization Stage and Incorporation of CFGs

In the revision stage, we replaced template-based realization with context-free grammars. The procedure is as follows. All the constituents including the grammar rules and the lexicon are demonstrated in CFG rules in terms of terminals and nonterminals. Every time we receive new input, we flexibly construct the CFG grammar rules on the fly and proceed to create the sentences based on NLTK's generation methods.

(15)

```
% start S
S -> FIXED S_REST
FIXED -> 'در این صفحه'
S_REST -> NP VP
NP -> OBJ1 CONJ OBJ2
OBJ1 -> 'مربع قرمز 5'
OBJ2 -> 'مثلث آبی 2'
CONJ -> 'و'
VP -> 'دیده می شود'
```

The example in (13) shows a sample CFG grammar for the identification module. Now if we want to include the relation of contrast between OBJ1 and OBJ2, we know where to systematically insert the adverb that refers to this relation. This gives us a greater amount of freedom to augment and change the structure of texts.

The problem with using raw CFGs is that, to avoid overgeneration and account for syntactical variations and dependencies, we need to build detailed and fully specified grammars. This gets out of hand soon if our lexicon grows bigger and we decide to cover more variation in our textual output. A better solution would be to use feature-based CFGs similar to HPSG grammars. However, implementing a feature-based realizer like FUF/Surge is beyond the scope of this research. One easy solution would be to first generate all the possible sentences using a raw CFG grammar and then weed out the incorrect ones using a chart-parser that accepts feature-based grammars. NLTK happens to support such a parser. In practice though, this takes a lot of time and as the grammars get more complex, it becomes less practical.

So, for the time being vanilla CFG is good enough for our purposes but incorporating feature-based methods is the future direction future research will take.

VI. Evaluation

There are several different methods to evaluate NLG systems. If the evaluation directly measures the quality of the text, then it is called intrinsic and if it concerns a task related to the text, it is called extrinsic (Reiter & Belz, 2009). Examples of extrinsic evaluations would be measuring the amount of post-edits done by the domain experts or finding out how fast people read the texts on average.

Traditionally NLG systems are evaluated by human judges (Mellish & Dale, 1998), but there exists a number of automatic tests that rely on a corpus of human generated text. One of the most famous of such metrics is BLEU that finds the number of common n-grams between the corpus and the generated text (Reiter & Belz, 2009). It outputs a number between 0 and 1. 1 is the state when all of the n-grams in the generated text exist in the corpus as well. Another similar metric is NIST which assigns more weight to rarer n-grams. These metrics seem to produce results that are consistently in accord with real human evaluation. However, using n-grams is fundamentally flawed because they cannot take into account long-distance dependencies and other fine details of the language (Belz & Reiter, 2006).

Given the exploratory nature of this research, we decided to evaluate the output using human judges. 10 descriptions were shown to 20 human judges. The judges were educated adults. They were asked to rate the system based on clarity, consistency and general quality. The scale was 0 to 5.

Table VI-1 Evaluation of PersNLG by human judges

	clarity	*consistency*	*General quality*
Easy pictures (max. 3 objects)	4.7	4.4	4.5
Difficult pictures	4.1	4.0	3.8

2.5 could be regarded as average and the figures obtained clearly show that the system has been successful in its purpose. There are several other research projects in NLG that have used human judges and similar criteria, and they normalize their

evaluations to 0-1 scale (Reiter & Belz, 2009). The numbers attained here are well above the baseline, but we don't cite the numbers for the baselines here because their domains are different (usually about weather reports) and a direct comparison would be problematic.

VII. Conclusion

We created a simple NLG system for Persian that can be considered the first data-to-text NLG system for the language. We did the image processing part using the SVG format and prototyped the first version of the system using templates. In the revision phase we started using the more linguistically rich context-free grammars that significantly improved our freedom over the shape of the final text and allowed us more control in incorporating more from the semantic network into the realization part.

In the end the system was evaluated using human judges. The results seem to indicate that the system has been successful in its narrow purpose of describing simple pictures of geometric shapes. For future projects, we will try to incorporate feature-based grammars into surface realization and also improve the image processing aspect by trying to detect real objects.

VIII. References

Aker, A., & Gaizauskas, R. (2010). Generating image descriptions using dependency relational patterns. *In Pr. ACL*, (pp. 1250–1258).

Baltaretu, A., Krahmer, E., & Maes, A. (2015). Moving Targets: Human References to Unstable Landmarks. *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)* (pp. 48-51). Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/W15-4706

Belz, A., & Reiter, E. (2006). Comparing Automatic and Human Evaluation of NLG Systems. *Annual Meeting of The European Chapter of The Association of Computational Linguistics* .

Brants, T., Popat, A. C., Xu, P., Och, F. J., & Dean, J. (2007). Large language models in machine translation. *In EMNLP-CoNLL.*

Carroll, D. W. (2008). *Psychology of language.* Australia ; Belmont, CA: Thomson/Wadsworth.

Chomsky, N. (1965). *Aspects of the Theory of Syntax.* MIT Press.

Dale, R., & Reiter, E. (2000). *Building natural language generation systems.* Cambridge, UK: Cambridge University Press. doi: ISBN 0-521-02451-X

Dalianis, H. (1999). Aggregation in Natural Language Generation. *Computational Intelligence, 15*.

Farhadi, A., Hejrati, M., Sadeghi, A., Young, P., Rashtchian, C., Hockenmaier, J., & Forsyth, D. A. (2010). Every picture tells a story: generating sentences for images. *ECCV.*

Ferres, L., Parush, A., Roberts, S., & Lindgaard, G. (2006). Helping People with Visual Impairments Gain Access to Graphical Information Through Natural Language: The iGraph System. *Lecture Notes in Computer Science*, 1122-1130. doi:10.1007/11788713_163

Gatt, A., & Reiter, E. (2009). SimpleNLG: A realisation engine for practical applications. *Proceedings of ENLG09*. Retrieved from http://www.aclweb.org/anthology/W09-0613.pdf

Goldberg, E., Driedger, N., & Kittredge, R. (1994). Using Natural-Language Processing to Produce Weather Forecasts. *IEEE Expert, 9*(2), 45–53. doi:10.1109/64.294135

Hovy, E. (1993). Automated discourse generation using discourse structure relations. *Artificial Intelligence*(63), 341-386.

Kulkarni, G., et al. (2011). Baby talk: Understanding and generating image descriptions. . *CVPR.*

Mann, W. C., & Thompson, S. A. (1988). Rhetorical Structure Theory: Toward a functional theory of text organization . *Text , 8*(3), 243-281.

McKeown, K. (1992). *Text Generation (Studies in Natural Language Processing).* Cambridge University Press.

Mellish, C., & Dale, R. (1998). Evaluation in the context of natural language generation. *Computer Speech and Language, 13*, 349–373.

Reiter, E., & Belz, A. (2009). An Investigation into the Validity of Some Metrics for Automatically Evaluating Natural Language Generation Systems. *Computational Linguistics, 35*(4), 529-558 .

Reiter, E., et al. (2009). Using NLG to Help Language-Impaired Users Tell Stories and Participate in Social Dialogues. *ENLG2009.* Athens, Greece: Association for Computational Linguistics .

Sauper, C., & Barzilay, R. (2009). Automatically Generating Wikipedia Articles: A Structure-Aware Approach. *Proceedings of ACL.*

Sevens, L., Vandeghinste, V., Schuurman, I., & Van Eynde, F. (2015). Natural Language Generation from Pictographs. *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)* (pp. 71-75). Brighton, UK: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/W15-4711

Woods, D. (2015, July 9). *Forbes*. (Forbes, Inc.) Retrieved from http://www.forbes.com/sites/danwoods/2015/07/09/why-big-data-needs-natural-language-generation-to-work/

Zhong, H. (2009). *Trainable English generation with modifier and adjunct ordering.* Stony Brook: State University of New York .