User-Controlled, Robust Natural Language Generation from an Evolving Knowledge Base

Eva Banik
Computational Linguistics Ltd
London, UK
ebanik@comp-ling.com

Eric Kow
Computational Linguistics Ltd
London, UK
kowey@comp-ling.com

Vinay Chaudhri*
SRI International
Menlo Park, CA
chaudhri@ai.sri.com

Abstract

In this paper we describe a natural language generation system which produces complex sentences from a biology knowledge base. The NLG system allows domain experts to discover errors in the knowledge base and generates certain parts of answers in response to users' questions in an e-textbook application. The system allows domain experts to customise its lexical resources and to set parameters which influence syntactic constructions in generated sentences. The system is capable of dealing with certain types of incomplete inputs arising from a knowledge base which is constantly edited and includes a referring expression generation module which keeps track of discourse history. Our referring expression module is available for download as the open source Antfarm tool[1].

1 Introduction

In this paper we describe a natural language generation system we have developed to interface with a biology knowledge base. The knowledge base (KB) encodes sentences from a biology textbook, and the ultimate goal of our project is to develop an intelligent textbook application which can eventually answer students' questions about biology[2] (Spaulding et al., 2011).

The natural language generation module is part of a larger system, which includes a question understanding module, question answering and reasoning algorithms, as well as an answer presentation module which produces pages with information from the KB. We measure the progress and consistency of encoding by asking "what is an X?" type questions of the application and evaluate the quality of answers. In response to these questions, the system generates "glossary pages" of concepts, which display all information about concept X in the KB that are deemed relevant. The NLG module is used for two purposes in our system: to check the completeness and consistency of the KB (instead of looking at complex graphs of the encoded knowledge, it is easier to detect errors in natural language sentences), and to present parts of answers in response to questions.

One goal of our project was to develop a tool which empowers biology teachers to encode domain knowledge with little training in formal knowledge representation. In the same spirit, we aimed to develop an NLG system which allowed domain experts to easily and intuitively customize the generated sentences as much as possible, without any training on the grammar or internal workings of the system. This was necessary because many domain-specific concepts in the KB are best expressed by biology terminology and linguistic constructions specific to the domain. We developed a utility which allows encoders to not only associate lexical items with concepts in the KB but also customise certain lexical parameters which influence the structure of sentences generated to describe events.

Another requirement was robustness: since the knowledge base is constantly edited, the NLG system had to be able to deal with missing lexical information, incomplete inputs, changing encoding guidelines, and bugs in the KB as much as possible. The system also had to be flexible in the sense

*The work reported in this paper was supported by funding from Vulcan, Inc. We would also like to thank the members of the Inquire Biology development team: Roger Corman, Nikhil Dinesh, Debbie Frazier, Stijn Heymans, Sue Hinojoza, David Margolies, Adam Overholtzer, Aaron Spaulding, Ethan Stone, William Webb, Michael Wessel and Neil Yorke-Smith.

[1] https://github.com/kowey/antfarm

[2] http://www.aaaivideos.org/2012/inquire_intelligent_textbook/

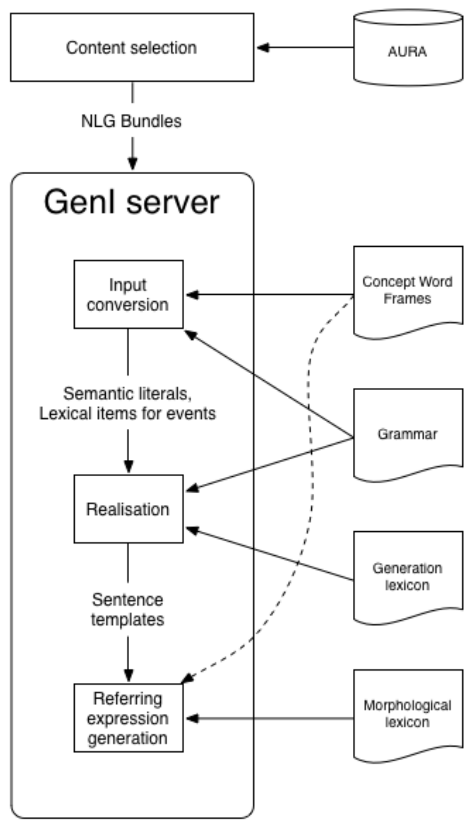


Figure 1: Architecture of the AURA NLG system

that it had to be able to generate different versions of the same output to suit specific contexts or types of concepts in its input. Our system therefore generates all possible realizations for a given input, and allows the answer presentation module to send parameters to determine which output is returned in a specific context.

After describing the architecture of the NLG module in detail we explain how the system is able to deal with unseen combination of event-to-entity relations when describing events. We illustrate the utility we developed to allow domain experts to customize the system's output by adding parameters to lexical entries associated with concepts.

2 Related Work

Work on natural language generation from ontologies and knowledge bases tends to fall into two groups. On the one hand, there are tools for ontology verbalization which tend to handle a limited number of relations, and where the goal of the system is to help the work of knowledge engineers. These systems produce template based outputs, and the texts closely follow the structure of the ontology (Wilcock, 2003; Galanis and Androutsopoulos, 2007). Some of these systems attempt to minimize reliance on domain-specific linguistic resources and attempt to detect words in the labels of the ontology to use as lexical items (Mellish and Sun, 2005). On the other hand there are NLG systems which take their input from complex knowledge bases (Reiter et al., 2003; Paris, 1988) and produce fluent texts geared towards users other than knowledge engineers. These systems produce outputs tailored to the user or the context and they are difficult for non-NLG-experts to customize or port to a different domain. Our system falls halfway between these two groups: like ontology verbalizers, we wanted to produce output for all inputs, using ontology labels if necessary in the absence of lexical entries. However, like sophisticated NLG systems, we also wanted to generate good quality output for inputs for which the system had lexical resources, and we also wanted to be able to tailor the generated output to the context in which it is displayed. Our input was also more expressive than the input of ontology verbalizers, because of the presence of cardinality constraints and co-references in our KB. Our work is perhaps most closely related to the MIAKT system which also allows domain experts to edit lexical knowledge and schemas (Bontcheva, 2004; Bontcheva and Wilks, 2004). Like MIAKT, we also aimed to develop an NLG system which can be easily maintained as the KB changes.

3 Architecture of the AURA NLG system

Our NLG system generates complex sentences from the AURA knowledge base (Gunning et al., 2010), which contains information from a college-level biology textbook. AURA is a frame-based KB which encodes events, the entities that participate in events, properties, and roles that the entities play in an event (e.g., catalyst, reactant, messenger, parent). The KB specifies relations between these types, including event-to-entity, event-to-event, event-to-property, entity-to-property. The AURA KB is built on top of the CLIB ontology of general concepts (Barker et al., 2001), which was extended with biology-specific information. The KB consists of a set of concept maps, which describe all the statements that are true about a concept in our KB. The input to our NLG system is a set of relations extracted from the KB either in response to users' questions or when generating glossary pages that describe specific concepts in detail. The generation pipeline consists of four main stages: content selection, input conversion, realisation and referring expression generation, as illustrated in Fig1.

3.1 Content Selection

Question answering and reasoning algorithms that return answers or other content in AURA are not engineered to satisfy the purposes of natural language generation. The output of these algorithms can be best thought of as pointers to concepts in the KB, which need to be described to provide an answer to the user. In order for the answer to be complete in a given context, the output of reasoning algorithms have to be extended with additional relations, depending on the specific question that was asked, and the context in which the answer was found in the KB. The relations selected from the KB also vary depending on the type of concept that is being described (event, entity, role, property). For example, a user might ask "What is a catalyst?". To answer this question, AURA will retrieve entities from the KB ("role players") which play the role of catalyst in various events. For example, it will find "adenylyl cyclase", which is defined in the KB as a universal catalyst, i.e., this information is encoded on the concept map of Adenylyl cyclase and is regarded as a "universal truth". In this case, our content selection algorithm will return a single `plays` triple, and the NLG system will produce *"Adenylyl cyclase is a catalyst'.'*
Another entity that will be returned in response to the question is "ribosomal RNA". However, ribosomal RNA is a catalyst only in specific situations, and therefore we need to give more detail on the contexts in which it can play the role of a catalyst. This includes the event in which ribosomal RNA is a catalyst, and perhaps the larger process during which this event occurs. Accordingly, content selection here will return a number of relations (including `agent`, `object`, `subevent`), and our NLG system will produce:
"In translation elongation, ribosomal RNA is a catalyst in the formation of a peptide bond by the ribosomal RNA and a ribosome."
Similarly, for "triose phosphate dehydrogenase" we will produce
"In energy payoff phase of glycolysis, NAD plus is converted by a triose phosphate dehydrogenase to a hydrogen ion, an NADH and a PGAP. Here, the triose phosphate dehydrogenase is a catalyst."

For "cellulose synthase" the situation is slightly different, because the event in which this entity plays the role of catalyst is not part of a larger process but the function of the entity. So we need slightly different information to produce the correct sentence: *"The function of cellulose synthase is conversion of a chemical in a cell to cellulose. Here, a cellulose synthase is a catalyst."*
The task of the AURA content selection module is to determine what information to include for each entity or event that was returned as the answer to the question. We do this by retrieving sets of relations from the KB that match contextual patterns. We also filter out relations which contain overly generic classes (e.g., Tangible-Entity), and any duplication arising from the presence of inverse relations or inferences in the KB. The output of content selection is a structured bundle (Fig. 2), which contains
(1) the relations that form the input to NLG
(2) information about concepts in the input: what class(es) they belong to, cardinality constraints
(3) parameters influencing the style of output texts.

3.2 Input Conversion

The realisation phase in our system is carried out by the GenI surface realizer (Kow, 2007), using a Tree-Adjoining Grammar (Joshi and Schabes, 1997). The task of the input conversion module is to interpret the structured bundles returned by content selection, and to convert the information to GenI's input format. We parse the structured bundles, perform semantic aggregation, interpret parameters in bundles which influence the style of the generated text, and convert triples to semantic literals as required by GenI.

4 Handling Unseen Combinations of Relations

As Fig 3 shows, a combination of event-to-entity relations are associated with elementary trees in the grammar to produce a full sentence. The domain of the relations associated with the same tree is the event which specifies the main predicate of the sentence and the range of the relations are entities that fill in the individual argument and modifier positions. Depending on the event, different relations can be used to fill in the subject and object positions, and verbs might determine the prepositions needed to realize some of the arguments. Ideally the mapping between sets

```
(TRIPLES-DATA
   :TRIPLES
     ((|_Cell56531|              |has-part|      |_Ribosome56523|)
      (|_Ribosome56523|          |has-part|      |_Active-Site56548|)
      (|Enzyme-Synthesis17634|   |base|          |_Cell56531|)
      (|Enzyme-Synthesis17634|   |raw-material|  |_Free-Energy56632|)
      (|Enzyme-Synthesis17634|   |raw-material|  |_Monomer56578|)
      (|Enzyme-Synthesis17634|   |raw-material|  |_Activation-Energy56580|)
      (|Enzyme-Synthesis17634|   |raw-material|  |_Monomer56581|)
      (|Enzyme-Synthesis17634|   |raw-material|  |_Amino-Acid56516|)
      (|Enzyme-Synthesis17634|   |result|        |_Free-Energy56575|)
      (|Enzyme-Synthesis17634|   |result|        |Protein-Enzyme17635|))
   :CONSTRAINTS
      ((|Enzyme-Synthesis17634|  |raw-material| (|at-least| 3 |Amino-Acid|)))
   :INSTANCE-TYPES
     ((|_Ribosome56523|          |instance-of| |Ribosome|)
      (|_Active-Site56548|       |instance-of| |Active-Site|)
      (|_Cell56531|              |instance-of| |Cell|)
      (|_Free-Energy56632|       |instance-of| |Free-Energy|)
      (|_Monomer56578|           |instance-of| |Monomer|)
      (|_Activation-Energy56580| |instance-of| |Activation-Energy|)
      (|_Monomer56581|           |instance-of| |Monomer|)
      (|_Amino-Acid56516|        |instance-of| |Amino-Acid|)
      (|_Free-Energy56575|       |instance-of| |Free-Energy|)
      (|Enzyme-Synthesis17634|   |instance-of| |Enzyme-Synthesis|)
      (|Protein-Enzyme17635|     |instance-of| |Protein-Enzyme|)
      (|Free-Energy|             |subclasses|  |Energy|)
      (|Activation-Energy|       |subclasses|  |Energy|)
      (|Free-Energy|             |subclasses|  |Energy|))
   :CONTEXT NIL
   :OUTPUT-PARAMETERS NIL)
```

A protein enzyme is synthesized in an active site of a ribosome of a cell using at least 3 amino acids and 2 monomers. This process transforms activation energy and free-energy to another free-energy.

Enzyme synthesis – a protein enzyme is synthesized in an active site of a ribosome of a cell using at least 3 amino acids and 2 monomers. This process transforms activation energy and free-energy to another free-energy.

Synthesis of a protein enzyme in an active site of a ribosome of a cell using at least 3 amino acids and 2 monomers. This process transforms activation energy and free-energy to another free-energy.

Figure 2: An example input bundle and the three outputs generated by our system for this input

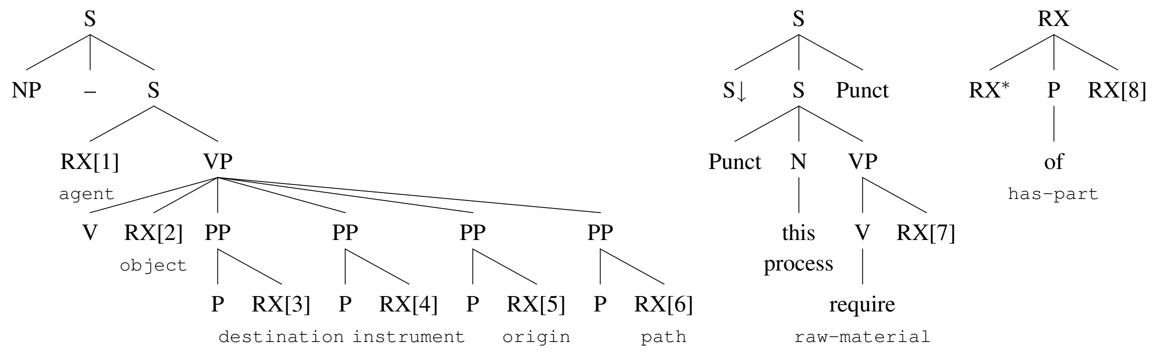


Figure 3: Tree selection

of event-to-entity relations and sentences would be given based on encoding guidelines used to create the knowledge base. However, the goal of our project is to continuously expand the knowledge base with more information, encoding new types of events, and enriching existing events with more detail as we go along (e.g., by specifying energy consumption and regulation mechanisms for processes), therefore our encoding guidelines are continuously revised. In order to produce output, our realizer requires a generation lexicon, which maps sets of relations onto elementary trees in the grammar. Determining this mapping would require knowing the number of entities that can be associated with each event type, and the relations that can be used to express them. However, because our knowledge base is continuously changing, neither the number of entities linked to specific events, nor the types of relations used are stable and therefore it was impossible to build such a generation lexicon from the KB. Instead, we adopted an approach where we detect "event frames" in the input of the system, and automatically create entries for them in the generation lexicon, guessing sentence structure and ordering based on the event participants. An event frame is a set of event-to-entity relations which have the same event in the domain of the relations, and participating entities in the range. We currently distinguish between two types of event frames, depending on the type of the entities in the range of relations: participant frames (ranges are of type Tangible-Entity) and energy frames (ranges are type Energy). An example of a participant frame and an energy frame extracted from the input illustrated in section 4.2 is illustrated below:

Participant frame:

```
(Uptake07 path Plasma-membrane78)
(Uptake07 origin Extracellular-Side52)
(Uptake07 destination Cytoplasm39)
(Uptake07 agent Cell-Surface-Receptor79)
(Uptake07 instrument Coated-Vesicle49)
(Uptake07 object Cholesterol08)
```

Energy frame:

```
(Uptake07 raw-material Chemical-Energy70)
(Uptake07 raw-material Free-Energy89)
```

Our input conversion module detects event frames and automatically creates an entry in GenI's generation lexicon for each frame, anchored on a noun or verb associated with the event in our concept-to-word mapping lexicon. The entries link the sets of relations in the frame to a tree with the same number of arguments, attempting to place entities that play `agent` and `object` participants into subject/object positions in the tree if they exist. Our algorithm also attempts to determine the best syntactic construction for the specific combination of participant relations, and decides between selecting an active sentential tree, a passive sentential tree, a complex noun phrase, or a combination of these. This process also involves deciding based on the event participants whether the tree will be anchored on a transitive verb, an intransitive verb, or a verb with a prepositional object, and assigning default prepositions to event participants (unless we have more detail specified in the lexicon, as described in the next section). The elementary trees in the grammar are named after the number of referring expressions and prepositional phrases in the tree, and we use this naming convention to automatically generate tree names (or tree family names) for lexical entries, thereby linking trees in the grammar to GenI's generation lexicon. The two S-rooted trees in Fig 3 were selected based on automatically generated lexical entries for the two frames above.

4.1 Realisation

The GenI surface realizer selects elementary TAG trees for (sets of) relations in its input and combines them using the standard operations of substitution and adjunction to produce a single derived tree. We have developed a feature-based lexicalized Tree Adjoining Grammar to generate sentences from relations in the KB. Our grammar has two important properties, following the approach in (Banik, 2010):
(1) our grammar includes discourse-level elementary trees for relations that are generated in separate sentences, and
(2) instead of the standard treatment of entities as nouns or NPs substituted into elementary trees, our grammar treats entities as underspecified referring expressions, leaving the generation of noun phrases to the next stage. The underspecified referring expressions replace elementary trees in the grammar, which the generator would otherwise have to combine with substitution. This underspecification saves us computational complexity in surface realisation, and at the same time allows us to make decisions on word choice at a later stage when we have more information on the syntax of the sentence and discourse history.

The output of the realizer is an underspecified text in the form of a sequence of lemma - feature structure pairs. Lemmas here can be underspecified – instead of an actual word, they can be an index or a sequence of indices pointing to concepts in the KB. The syntax and sentence boundaries are fully specified, and the output can be one or more sentences long. The feature structures associated with lemmas include all information necessary for referring expression generation and morphological realisation, which is performed in the next phase. To give an example, the set of relations below would produce an output with 8 underspecified referring expressions (shown as RX), distributed over two sentences:

```
(Uptake07 path   Plasma-membrane78)
(Uptake07 origin Extracellular-Side52)
```

```
(Uptake07 destination Cytoplasm39)
(Uptake07 agent Cell-Surface-Receptor79)
(Uptake07 instrument Coated-Vesicle49)
(Uptake07 object Cholesterol08)
(Uptake07 raw-material Chemical-Energy70)
(Uptake07 raw-material Free-Energy89)
```

NP(Uptake07) – RX[1] absorb RX[2] to RX[3] of RX[8] with RX[4] from RX[5] through RX[6]. This process requires RX[7].

The elementary trees selected by the realizer for this output, and the correspondences between relations and referring expressions are illustrated in Fig.3.

4.2 Referring Expression Generation

The final stage in the NLG pipeline is performing morphological realisation and spelling out the referring expressions left underspecified by the realisation module. The input to referring expression generation is a list of lemma - feature structure pairs, where lemmas are words on leaf nodes in the derived tree produced by syntactic realisation. In our system, some of the lemmas can be unspecified, i.e., there is no word associated with the leaf node, only a feature structure. For these cases, we perform lexicon lookup and referring expression generation based on the feature structure, as well as morphological realisation. To give an example, the input illustrated in the previous section will be generated as

"Uptake of cholesterol by human cell– a cell surface receptor absorbs cholesterol to the cytoplasm of a human cell with a coated vesicle from an extracellular side through a plasma membrane. This process requires chemical energy and free-energy."

Many concept labels in our ontology are very complex, often giving a description of the concept or the corresponding biology terminology, and therefore these labels can only be used for NLG under specific circumstances. To overcome this problem, we have created a lexicon that maps concept names to words, and the grammar has control over which form is used in a particular construction. Accordingly, we distinguish between two types of underspecified nodes:

- NP nodes where the lexical item for the node is derived by normalizing the concept class associated with the node (Uptake-Of-Cholesterol-By-Human-Cell $\rightarrow$ "uptake of cholesterol by human cell")
- RX (referring expression) nodes where lexical items are obtained by looking up class names in the concept-to-word mapping lexicon (Uptake-Of-Cholesterol-By-Human-Cell $\rightarrow$ "absorb")

The feature structures on RX nodes in the output of GenI describe properties of entities in the input, which were associated with that specific node during realisation. The feature structures specify three kinds of information:

- the identifier (or a list of identifiers) for the specific instances of entities the RX node refers to
- the KB class for each entity
- any cardinality constraints that were associated with each entity for the relation expressed by the tree in which the RX node appears

We define cardinality constraints as a triple (Domain, Slot, Constraint) where the Constraint itself is another triple of the form (ConstraintExpression, Number, ConstraintClass). ConstraintExpression is one of `at least`, `at most`, or `exactly` and ConstraintClass is a KB class over which the constraint holds. There is usually (but not necessarily) one or more relations associated with every cardinality constraint. We say a triple (Domain Slot Range) is associated with a cardinality constraint (Domain, Slot, (ConstraintExpression, Number, ConstraintClass)) if

- the Domain and Slot of the associated triple is equal to the Domain and Slot of the cardinality constraint and
- one of the following holds:
  - either (Range instance-of ConstraintClass) holds for the range of the triple
  - or Range is taxonomically related to ConstraintClass (via a chain of subclass relations)

We define a referring expression language (Fig. 4) which describes groups of instance names (variables) that belong to the same KB class, and the associated cardinality constraints. Groups themselves can be embedded within a larger group (an umbrella), resulting in a complex expression which gives examples of a concept (e.g., *"three atoms (a carbon and two oxygens)"*). Expressions

```
<refex>       = <umbrella> SPACE <refex> | <umbrella>
<umbrella>    = <group> ( <refex> )| <group>
<group>       = <class> <instances> <constraints>
<instances>   = :: <instance> <instances> | <instance>
<constraints> = :  <constraint> <constraints>  | <constraint>
<constraint>  = <op> : <num> | unk : <dash-delimited-string>
<op>          = ge | le | eq
```

Figure 4: Syntax of the referring expression language

in this language are constructed from triples during the input conversion stage, when we perform semantic aggregation. The groups are then passed through elementary trees by the realisation module (GenI) to appear in the output as complex feature structures on leaf nodes of the derived tree. The referring expression generation module parses these complex feature values, and constructs (possibly complex) noun phrases as appropriate.

To illustrate some examples, the following feature value shows a simple referring expression group which encodes two entities (Monomer14 and Monomer7) and two cardinality constraints (at least 2 and at most 5). This expression will be generated as "between 2 and 5 monomers":

```
Monomer::Monomer14::Monomer7:ge:2:le:5
```

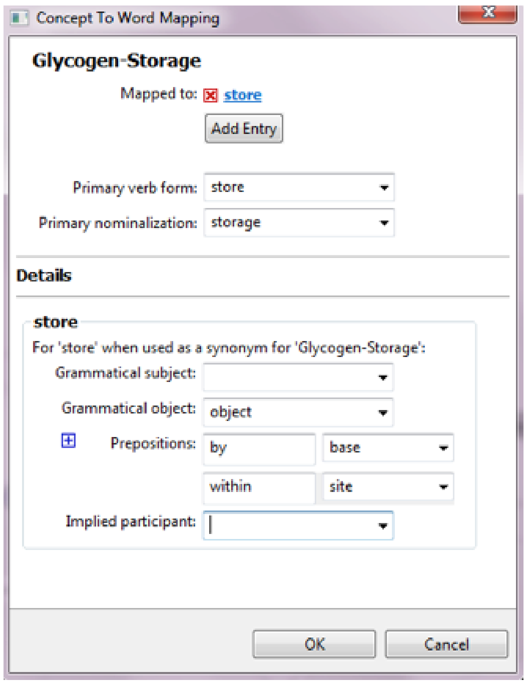
We also allow more complex cardinality constraints which give the general type of an entity and specify examples of the general type, as in *"at least 3 organic molecules (2 ATPs and an ethyl alcohol)"*:

```
Organic-Molecule:ge:3
 (ATP:: ATP80938:eq:2
  Ethyl-Alcohol:: Ethyl-Alcohol80922)
```

The referring expression generation module makes three main decisions based on the referring expression, additional feature structures on the node, and discourse history: it chooses lemmas, constructs discriminators, and decides between singular/plural form. The algorithm for discriminator choice in the referring expression generation module is illustrated in Fig 5. Our referring expression generation module, including discourse history tracking and determiner choice, is made available in the Antfarm[3] open source tool.

5 Giving Domain Experts Control over Sentence Structure

By automatically associating event frames with elementary trees we are able to generate a sentence for all combinations of event-to-entity relations


Concept To Word Mapping

Glycogen-Storage

Mapped to: store

Primary verb form: store

Primary nominalization: storage

Details

store

For 'store' when used as a synonym for 'Glycogen-Storage':

Grammatical subject:

Grammatical object: object

Prepositions: by — base

within — site

Implied participant:

Figure 6: Parameters in the concept-to-word mapping lexicon

without having to maintain the grammar and generation lexicon of the realizer as the knowledge base evolves. However sentences generated this way are not always well-formed. Events in the KB can be realized with a wide range of verbs and nouns, which require different prepositions or syntactic constructions, and different types of events may require different participants to be their grammatical subject or object. To give an example, for events that have an agent, in the majority of the cases we get a grammatical sentence if we place the agent in subject position. If the frame lacks an agent but has an object, we can usually generate a grammatical passive sentence, with the object participant as the subject. However, it is often the case that events do not have an agent, and we get a grammatical (active) sentence by placing another relation in the subject position e.g., `base` for the event Store or `instrument` for Block. Which

[3]https://github.com/kowey/antfarm

```
for each group in the referring expression do
    if all members of the group are first mentions and there are no distractors in the history: then
        if the group has cardinality constraints: then
            upper bound $M$ → at most M
            lower bound $N$ → at least N (multiple group members in this case are also interpreted as lower bound)
            both bounds → between N and M or exactly N
        else
            one group member → generate an indefinite determiner (a/an)
            more than one member → generate a cardinal
        end if
    end if
    if the group is a first mention but there are distractors in the discourse history then
        if the group has only one member then
            if the group exactly matches one previous mention → another
            if the group exactly matches $N > 1$ previous mentions → the Nth
            if there is a 2-member group in the history, and one of the members was mentioned by itself → the other
            if the discourse history has more than one distractor → a(n) Nth
        end if
        if there are multiple group members then
            if the group is a subset of a previously mentioned group which has no distractors → N of the
        end if
    end if
    if the group is not a first mention then
        if the group has upper and/or lower bounds → the same
        if the group has one member only → the
        if the group has multiple members → the N
    end if
end for
```

Figure 5: Algorithm for discriminator choice in our referring expression module

event participant can appear in subject and object positions depends not only on the type of the event, but also on the encoding guidelines which are continuously evolving.

In order to improve the quality of the generated output, and to give domain experts control over customizing the system without having to understand details of the grammar, we extended the concept-to-word mapping lexicon with parameters which control preposition choice, and allow customization of the position of participating entities. We developed a graphical user interface which allows encoders (biology domain experts) to add and edit these lexical parameters as they encode concepts in the KB.

To give an example, in the absence of a lexical item and any parameters for the event Glycogen-Storage, our system would produce the following default output, attempting to use the concept label as the main verb of the sentence in an automatically produced generation lexicon entry:

"Glycogen storage – glycogen is glycogened storage in a vertebrate in a liver cell and a muscle cell."

In order to improve the quality of the output, one of our biology teachers has customized the parameters in the lexicon to yield:

"Glycogen storage – glycogen is stored by a vertebrate within a liver cell and a muscle cell."

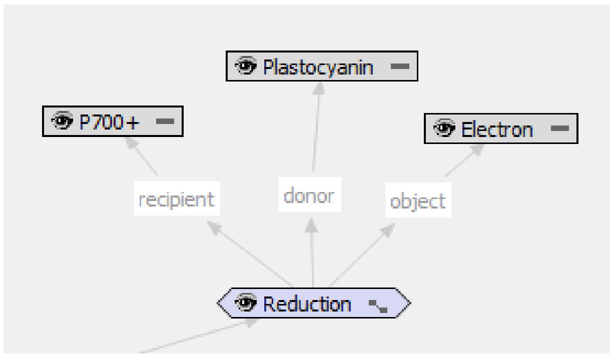
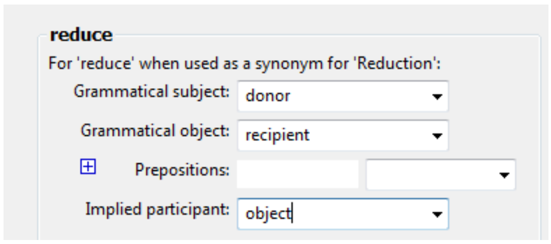


Figure 7: Concept map for the event 'Reduction'

This was achieved through a graphical user interface which is part of the tool used for knowledge encoding, and is illustrated in Fig 6. Our system allows encoders to re-generate sentences after editing the parameters to see the effect of the changes on the output. The top half of the window in in Fig 6 allows encoders to associate words or phrases with concepts, where they can add as many synonyms as they see fit. One of the synonyms has to be marked as the primary form, to be used for generation by default.[4] For events,

[4]The concept-to-word mapping lexicon is shared between the question interpretation and the NLG module, and the additional synonyms are currently only used for mapping ques-

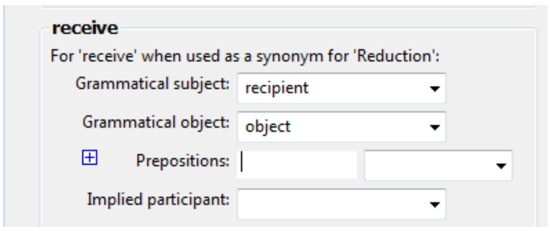


reduce

For 'reduce' when used as a synonym for 'Reduction':

Grammatical subject:	donor
Grammatical object:	recipient
Prepositions:	
Implied participant:	object

(a) “Plastocyanin reduces P700+”



receive

For 'receive' when used as a synonym for 'Reduction':

Grammatical subject:	recipient
Grammatical object:	object
Prepositions:	
Implied participant:	

(b) “P700+ receives an electron from plastocyanin.”

Figure 8: Concept-to-word mapping parameters for the two synonyms of Reduction

the primary form is a verb and its nominalization, and for entities it is a noun. The bottom half of the window shows the parameter settings for each synonym associated with the concept. Here the encoders can specify relations which link the subject and object of a verb to the event (grammatical subject/object), and assign prepositions to other event-to-entity relations for the verb, when it is used to realize the specified event. There is also an option to tell the NLG system to ignore some of the event participants when using a specific verb for the event. This functionality is used for verbs that already imply one of the participants. For example, the word polymerization already implies that the result of the event is a polymer. In these cases there is no need for the NLG system to generate the implied participant (here, result). Another example is the verb reduce, which implies that the object of the event is an electron. The editor allows the users to enter different parameter values for the synonyms of the same event. For example, the graph in Fig 7 could be described in at least three different ways:

1. P700+ is reduced by plastocyanin
2. Plastocyanin reduces P700+
3. P700+ receives an electron from plastocyanin.

Here sentences 1 and 2 make no mention of the electron involved in the process, but sentence 3 explicitly includes it. In order for the system to correctly generate sentences 1 and 2, the concept-to-word mapping parameters for “reduce” (as a synonym for Reduction) have to include an implied participant. Otherwise the system will assume that all participants should be mentioned in the sentence, and it will generate “P700+ is reduced by a plastocyanin of an electron”. Fig 8. illustrates the different concept-to-word mapping parameters needed for the two synonyms for Reduction in order to generate the above sentences correctly.

tions onto concepts in the KB.

6 Conclusions

We have presented an NLG system which generates complex sentences from a biology KB. Our system includes a content selection module, which tailors the selected relations to the context in which the output is displayed, and allows the presentation module to send parameters to influence properties of generated outputs. We have developed a referring expression generation module which generates complex noun phrases from aggregated cardinality constraints and entities in the input, and keeps track of discourse history to distinguish mentions of different groups of concepts. Our system allows biology teachers to detect inconsistencies and incompleteness in the KB, such as missing cardinality constraints, errors where two instances of the concept were added unnecessarily (unification errors on entities), and missing or incorrect relations. To make the system robust, we have developed an algorithm to produce sentences and complex noun phrases for unseen combinations of event-to-entity relations in the KB by automatically generating entries in the lexicon of the GenI surface realizer. Our algorithm makes default decisions on sentence structure and ordering based on relations sent to the NLG system, expressing the event’s participants. To allow domain experts to easily improve the default outputs generated by our algorithm, we have defined a framework for adding lexical parameters to concepts, which allow non-NLG-experts to customize the structure of generated sentences for events in the KB as they are encoded. Although our system currently only produces one or two possibly complex sentences, it was designed to ultimately generate paragraph-length texts. This can be achieved simply by adding more discourse-level elementary trees to the grammar of the realizer, since our system is already able to handle referring expressions across sentence boundaries.

References

E. Banik. 2010. *A Minimalist Architecture for Generating Coherent Text*. Ph.D. thesis, The Open University, UK.

K. Barker, B. Porter, and P. Clark. 2001. A library of generic concepts for composing knowledgebases. In *Proceedings K-CAP 2001*, pages 14–21.

K. Bontcheva and Y. Wilks. 2004. Automatic report generation from ontologies: the MIAKT approach. In *9th Int. Conf. on Applications of Natural Language to Information Systems*, page 324335, Manchester, UK.

K. Bontcheva. 2004. Open-source tools for creation, maintenance, and storage of lexical resources for language generation from ontologies. In *4th Conf. on Language Resources and Evaluation*, Lisbon, Portugal.

D. Galanis and I. Androutsopoulos. 2007. Generating multilingual descriptions from linguistically annotated owl ontologies: the NaturalOWL system. In *INLG07, Schloss Dagstuhl, Germany*, page 143146.

D. Gunning, V. K. Chaudhri, P. Clark, K. Barker, Shaw-Yi Chaw, M. Greaves, B. Grosof, A. Leung, D. McDonald, S. Mishra, J. Pacheco, B. Porter, A. Spaulding, D. Tecuci, and J. Tien. 2010. Project halo update - progress toward digital aristotle. *AI Magazine*, Fall:33–58.

A. K. Joshi and Y. Schabes. 1997. Tree-Adjoining Grammars. In Grzegorz Rosenberg and Arto Salomaa, editors, *Handbook of Formal Languages and Automata*, volume 3, pages 69–124. Springer-Verlag, Heidelberg.

E. Kow. 2007. *Surface realisation: ambiguity and determinism*. Ph.D. thesis, Universite de Henri Poincare, Nancy.

C. Mellish and X. Sun. 2005. The semantic web as a linguistic resource: Opportunities for natural language generation. In *Knowledge-Based Systems*.

C.L. Paris. 1988. Tailoring object descriptions to the users level of expertise. *Computational Linguistics*, 14(3):6478. Special Issue on User Modelling.

E. Reiter, R. Robertson, and L. M. Osman. 2003. Lessons from a failure: generating tailored smoking cessation letters. *Artificial Intelligence*, 144(1-2):41–58.

A. Spaulding, A. Overholtzer, J. Pacheco, J. Tien, V. K. Chaudhri, D. Gunning, and P. Clark. 2011. Inquire for ipad: Bringing question-answering ai into the classroom. In *International Conference on AI in Education (AIED)*.

G. Wilcock. 2003. Talking owls: Towards an ontology verbalizer. In *Human Lan- guage Technology for the Semantic Web and Web Services, ISWC03*, page 109112, Sanibel Island, Florida.