Generation of single-sentence paraphrases from predicate/argument structure using lexico-grammatical resources

Raymond Kozlowski, Kathleen F. McCoy, and K. Vijay-Shanker
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716, USA
kozlowsk,mccoy,vijay@cis.udel.edu

Abstract

Paraphrases, which stem from the variety of lexical and grammatical means of expressing meaning available in a language, pose challenges for a sentence generation system. In this paper, we discuss the generation of paraphrases from predicate/argument structure using a simple, uniform generation methodology. Central to our approach are lexico-grammatical resources which pair elementary semantic structures with their syntactic realization and a simple but powerful mechanism for combining resources.

1 Introduction

In natural language generation, producing some realization of the input semantics is not the only goal. The same meaning can often be expressed in various ways using different lexical and syntactic means. These different realizations, called *paraphrases*, vary considerably in appropriateness based on pragmatic factors and communicative goals. If a generator is to come up with the most appropriate realization, it must be capable of generating all paraphrases that realize the input semantics. Even if it makes choices on pragmatic grounds during generation and produces a single realization, the ability to generate them all must still exist.

Variety of lexical and grammatical forms of expression pose challenges to a generator ((Stede, 1999); (Elhadad et al., 1997); (Nicolov et al., 1995)). In this paper, we discuss the generation of single-sentence paraphrases realizing the same semantics in a uniform fashion using a simple sentence generation architecture.

In order to handle the various ways of realizing meaning in a simple manner, we believe that the generation architecture should not be aware of the variety and not have any special mechanisms to handle the different types of realizations[1]. Instead, we want all lexical and grammatical variety to follow automatically from the variety of the elementary building blocks of generation, lexico-grammatical resources.

We have developed a fully-operational prototype of our generation system capable of generating the examples presented in this paper, which illustrate a wide range of paraphrases. As we shall see, the paraphrases that are produced by the system depend entirely on the actual lexicon used in the particular application. Determining the range of alternate forms that constitute paraphrases is not the focus of this work. Instead, we describe a framework in which lexico-grammatical resources, if properly defined, can be used to generate paraphrases.

2 Typical generation methodology

Sentence generation takes as input some semantic representation of the meaning to be conveyed in a sentence. We make the assumption that

[1]Ability to handle variety in a uniform manner is also important in multilingual generation as some forms available in one language may not be available in another.

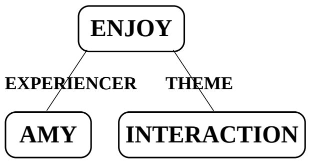

Figure 1: The semantics underlying (2a-2b)

the input is a hierarchical predicate/argument structure such as that shown in Fig. 1. The output of this process should be a set of grammatical sentences whose meaning matches the original semantic input.

One standard approach to sentence generation from predicate/argument structure (like the semantic-head-driven generation in (Shieber et al., 1990)) involves a simple algorithm.

1. decompose the input into the top predicate (to be realized by a (single) lexical item that serves as the syntactic head) and identify the arguments and modifiers
2. recursively realize arguments, then modifiers
3. combine the realizations in step 2 with the head in step 1

In realizing the input in Fig. 1, the input can be decomposed into the top predicate which can be realized by a syntactic head (a transitive verb) and its two arguments, the experiencer and the theme. Suppose that the verb *enjoy* is chosen to realize the top predicate. The two arguments can then be independently realized as *Amy* and *the interaction*. Finally, the realization of the experiencer, *Amy*, can be placed in the subject position and that of the theme, *the interaction*, in the complement position, yielding (2a).

Our architecture is very similar but we argue for a more central role of lexico-grammatical resources driving the realization process.

3 Challenges in generating paraphrases

Paraphrases come from various sources. In this section, we give examples of some types of paraphrases we handle and discuss the challenges they pose to other generators. We also identify types of paraphrases we do not consider.

3.1 Paraphrases we handle

Simple synonymy The simplest source of paraphrases is simple synonymy. We take simple synonyms to be different words that have the same meaning and are of the same syntactic category and set up the same syntactic context.

(1a) *Booth killed Lincoln.*

(1b) *Booth assassinated Lincoln.*

A generation system must be able to allow the same semantic input to be realized in different ways. Notice that the words *kill* and *assassinate* are not always interchangeable, e.g., *assassinate* is only appropriate when the victim is a famous person. Such constraints need to be captured with selectional restrictions lest inappropriate realizations be produced.

Different placement of argument realizations Sometimes different synonyms, like the verbs *enjoy* and *please*, place argument realizations differently with respect to the head, as illustrated in (2a-2b).

(2a) *Amy enjoyed the interaction.*

(2b) *The interaction pleased Amy.*

To handle this variety, a uniform generation methodology should not assume a fixed mapping between thematic and syntactic roles but let each lexical item determine the placement of argument realizations. Generation systems that use such a fixed mapping must override it for the divergent cases (e.g., (Dorr, 1993)).

Words with overlapping meaning There are often cases of different words that realize different but overlapping semantic pieces. The easiest way to see this is in what has been termed *incorporation*, where a word not only realizes a predicate but also one or more of its arguments. Different words may incorporate different arguments or none at all, which may lead to paraphrases, as illustrated in (3a-3c).

(3a) *Charles flew across the ocean.*

(3b) *Charles crossed the ocean by plane.*

(3c) *Charles went across the ocean by plane.*

Notice that the verb *fly* realizes not only going but also the mode of transportation being a plane, the verb *cross* with its complement realize going whose path is across the object realized by the complement, and the verb *go* only realizes going. For all of these verbs, the remaining arguments are realized by modifiers.

Incorporation shows that a uniform generator should use the word choices to determine 1) what portion of the semantics they realize, 2) what portions are to be realized as arguments of the realized semantics, and 3) what portions remain to be realized and attached as modifiers. Generation systems that assume a one-to-one mapping between semantic and syntactic units (e.g., (Dorr, 1993)) must use special processing for cases of overlapping semantics.

Different syntactic categories Predicates can often be realized by words of different syntactic categories, e.g., the verb *found* and the noun *founding*, as in (4a-4b).

(4a) *I know that Olds founded GM.*

(4b) *I know about the founding of GM by Olds.*

Words of different syntactic categories usually have different syntactic consequences. One such consequence is the presence of additional syntactic material. Notice that (4b) contains the prepositions *of* and *by* while (4a) does not. These prepositions might be considered a syntactic consequence of the use of the noun *founding* in this configuration. Another syntactic consequence is a different placement of argument realizations. The realization of the founder is the subject of the verb *found* in (4a) while in (4b) the use of *founding* leads to its placement in the object position of the preposition *by*.

Grammatical alternations Words can be put in a variety of grammatical alternations such as the active and passive voice, as in (5a-5b), the topicalized form, the it-cleft form, etc.

(5a) *Oswald killed Kennedy.*

(5b) *Kennedy was killed by Oswald.*

The choice of different grammatical alternations has different syntactic consequences which must be enforced in generation, such as the presence or absence of the copula and the different placement of argument realizations. In some systems such as ones based on Tree-Adjoining Grammars (TAG), including ours, these consequences are encapsulated within elementary structures of the grammar. Thus, such systems do not have to specifically reason about these consequences, as do some other systems.

More complex alternations The same content of excelling at an activity can be realized by the verb *excel*, the adverb *well*, and the adjective *good*, as illustrated in (6a-6c).

(6a) *Barbara excels at teaching.*

(6b) *Barbara teaches well.*

(6c) *Barbara is a good teacher.*

This variety of expression, often called *head switching*, poses a considerable difficulty for most existing sentence generators. The difficulty stems from the fact that the realization of a phrase (sentence) typically starts with the syntactic head (verb) which sets up a syntactic context into which other constituents are fit. If the top predicate is the excelling, we have to be able to start generation not only with the verb *excel* but also with the adverb *well* and the adjective *good*, typically not seen as setting up an appropriate syntactic context into which the remaining arguments can be fit. Existing generation systems that handle this variety do so using special assumptions or exceptional processing, all in order to start the generation of a phrase with the syntactic head (e.g., (Stede, 1999), (Elhadad et al., 1997), (Nicolov et al., 1995), (Dorr, 1993)). Our system does not require that the semantic head map to the syntactic head.

Different grammatical forms realizing semantic content Finally, we consider a case, which to our knowledge is not handled by other generation systems, where grammatical forms realize content independently of the lexical item on which they act, as in (7a-7b).

(7a) *Who rules Jordan?*

(7b) *Identify the ruler of Jordan!*

The wh-question form, as used in (7a), realizes a request for identification by the listener (in this case, the ruler of Jordan). Likewise, the imperative structure (used in (7b)) realizes a request or a command to the listener (in this case, to identify the ruler of Jordan).

3.2 Paraphrases we do not consider

Since our focus is on sentence generation and not sentence planning, we only consider the generation of single-sentence paraphrases. Hence, we do not have the ability to generate (8a-8b) from the same input.

(8a) *CS1 has a programming lab.*

(8b) *CS1 has a lab. It involves programming.*

Since we do not reason about the semantic input, including deriving entailment relations, we cannot generate (9a-9b) from the same input.

(9a) *Oslo is the capital of Norway.*

(9b) *Oslo is located in Norway.*

4 Our generation methodology

Generation in our system is driven by the semantic input, realized by selecting lexico-grammatical resources matching pieces of it, starting with the top predicate. The realization of a piece containing the top predicate provides the syntactic context into which the realizations of the remaining pieces can be fit (their placement being determined by the resource).

The key to our ability to handle paraphrases in a uniform manner is that our processing is driven by our lexicon and thus we do not make any a priori assumptions about 1) the amount of the input realized by a lexical unit, 2) the relationship between semantic and syntactic types (and thus the syntactic rank or category of the realization of the top piece), 3) the nature of the mapping between thematic roles and syntactic positions, and 4) the grammatical alternation (e.g., there are different resources for the same verb in different alternations: the active, passive, topicalized, etc.). Because this information is contained in each lexico-grammatical resource, generation can proceed no matter what choices are specified about these in each individual resource. Our approach is fundamentally different from systems that reason directly about syntax and build realizations by syntactic rank ((Bateman, 1997), (Elhadad et al., 1997); (Nicolov et al., 1995); (Stone and Doran, 1997)).

4.1 Our algorithm

Our generation algorithm is a simple, recursive, semantic-head-driven generation process, consistent with the approach described in section 2, but one driven by the semantic input and the lexico-grammatical resources.

1. given an unrealized input, find a lexico-grammatical resource that matches a portion including the top predicate and satisfies any selectional restrictions
2. recursively realize arguments, then modifiers
3. combine the realizations in step 2 with the resource in step 1, as determined by the resource in step 1

Notice the prominence of lexico-grammatical resources in steps 1 and 3 of this algorithm. The standard approach in section 2 need not be driven by resources.

4.2 Lexico-grammatical resources

The key to the simplicity of our algorithm lies in the lexico-grammatical resources, which encapsulate information necessary to carry through generation. These consist of three parts:

- the semantic side: the portion of semantics realized by the resource (including the predicate and any arguments; this part is matched against the input semantics)
- the syntactic side: either word(s) in a syntactic configuration or a grammatical form without words, and syntactic consequences
- a mapping between semantic and syntactic constituents indicating which constituent on the semantic side is realized by which constituent on the syntactic side

Consider the resources for the verbs *enjoy* and *please* in Fig. 2. The semantic sides indicate that these resources realize the predicate `ENJOY` and the thematic roles `EXPERIENCER` and `THEME`. The arguments filling those roles (which must be realized separately, as indicated by dashed outlines) appear as variables `X` and `Y` which will be matched against actual arguments. The syntactic sides contain the verbs *enjoy* and *please* in the active voice configuration. The mappings include links between `ENJOY` and its realization as well as links between the unrealized agent (`X`) or theme (`Y`) and the subject or the complement.

Our mapping between semantic and syntactic constituents bears resemblance to the pairings in Synchronous TAG (Shieber and Schabes, 1990). Just like in Synchronous TAG, the mapping is

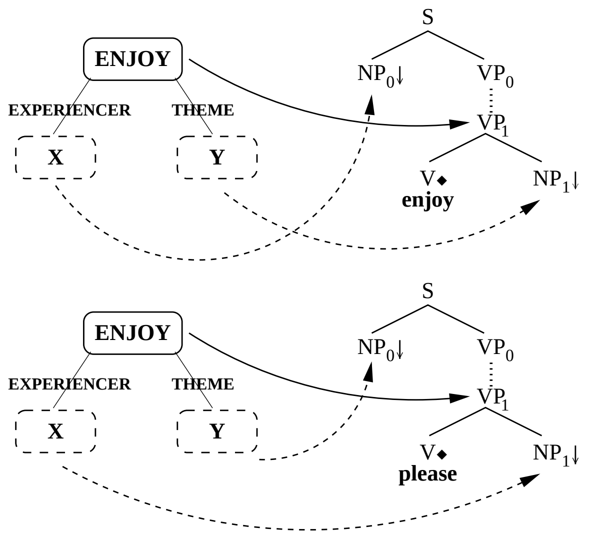

Figure 2: Two different resources for `ENJOY`

critical for combining realizations (in step 3 of our algorithm in section 4.1). There are, however, advantages that our approach has. For one, we are not constrained by the isomorphism requirement in a Synchronous TAG derivation. Also, the DSG formalism that we use affords greater flexibility, significant in our approach, as discussed later in this paper (and in more detail in (Kozlowski, 2002b)).

4.3 The grammatical formalism

Both step 3 of our algorithm (putting realizations together) and the needs of lexico-grammatical resources (the encapsulation of syntactic consequences such as the position of argument realizations) place significant demands on the grammatical formalism to be used in the implementation of the architecture. One grammatical formalism that is well-suited for our purposes is the D-Tree Substitution Grammars (DSG, (Rambow et al., 2001)), a variant of Tree-Adjoining Grammars (TAG). This formalism features an extended domain of locality and flexibility in encapsulation of syntactic consequences, crucial in our architecture.

Consider the elementary DSG structures on the right-hand-side of the resources for *enjoy* and *please* in Fig. 2. Note that nodes marked with ↓ are substitution nodes corresponding to syntactic positions into which the realizations of arguments will be substituted. The positions of both the subject and the complement are encapsulated in these elementary structures. This allows the mapping between semantic and syntactic constituents to be defined locally within the resources. Dotted lines indicate domination of length zero or more where syntactic material (e.g., modifiers) may end up.

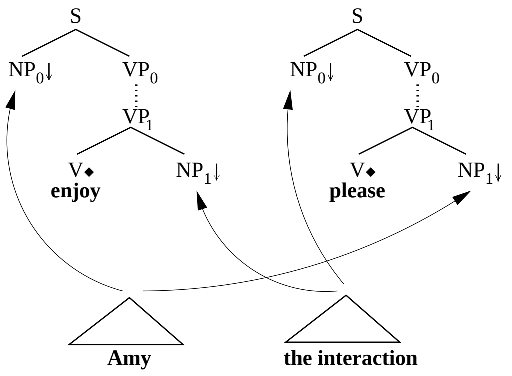

Figure 3: Combining argument realizations with the resources for *enjoy* and *please*

4.4 Using resources in our algorithm

Step 1 of our algorithm requires matching the semantic side of a resource against the top of the input and testing selectional restrictions. A semantic side matches if it can be overlaid against the input. Details of this process are given in (Kozlowski, 2002a). Selectional restrictions (type restrictions on arguments) are associated with nodes on the semantic side of resources. In their evaluation, the appropriate knowledge base instance is accessed and its type is tested. More details about using selectional restrictions in generation and in our architecture are given in (Kozlowski et al., 2002).

Resources for *enjoy* and *please* which match the top of the input in Fig. 1 are shown in Fig. 2. In doing the matching, the arguments `AMY` and `INTERACTION` are unified with `X` and `Y`. The dashed outlines around `X` and `Y` indicate that the resource does not realize them. Our algorithm calls for the independent recursive realization of these arguments and then putting together those realizations with the syntactic side of the resource, as indicated by the mapping.

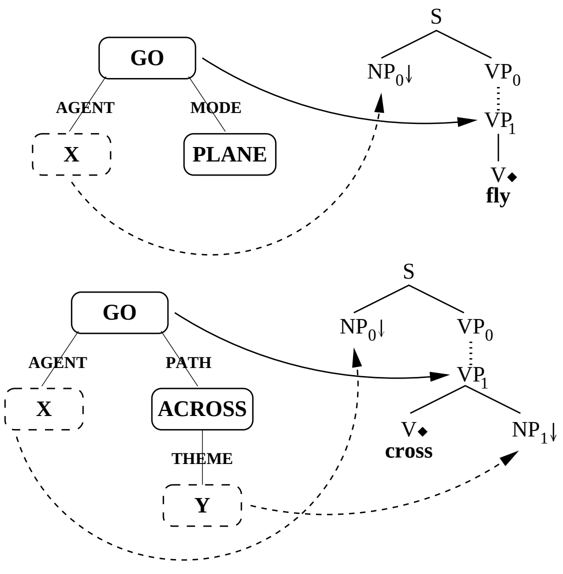

Figure 4: Two different resources for `GO`

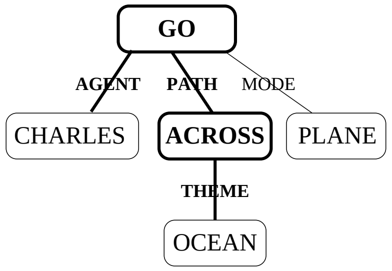

Figure 5: The semantics underlying (3a-3c) with portion realized by *cross* in bold

This is shown in Fig. 3. The argument realizations, *Amy* and *the interaction*, are placed in the subject and complement positions of *enjoy* and *please*, according to the mapping in the corresponding resources.

4.5 Driving decomposition by resources

The semantic side of a resource determines which arguments, if any, are realized by the resource, while the matching done in step 1 of our algorithm determines the portions that must be realized by modifiers. This is always done the same way regardless of the resources selected and how much of the input they realize, such as the two resources realizing the predicate `GO` shown in Fig. 4, one for *fly* which incorporates `MODE PLANE` and another for *cross* which incorporates `PATH ACROSS`.

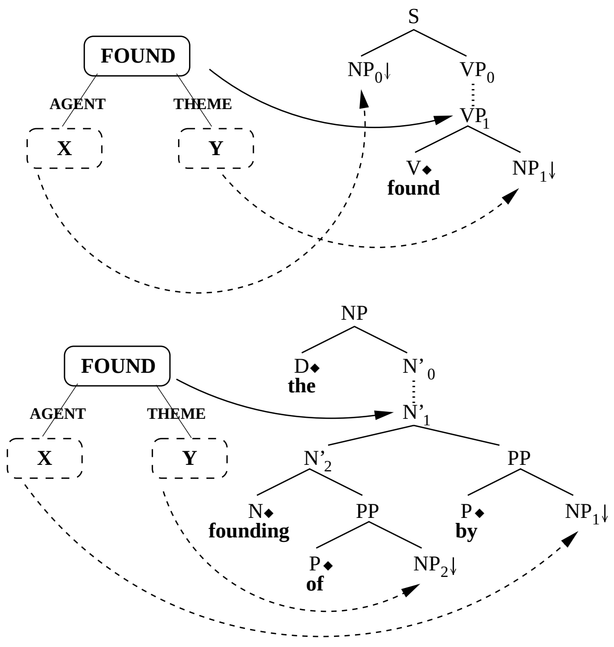

Figure 6: Two different resources for `FOUND`

Suppose the semantic input underlying (3a-3c) is as given in Fig. 5. The portion shown in bold is realized by the resource for *cross* in Fig. 4. The agent of `GO` and the theme of `ACROSS` are to be realized as arguments. The remaining thematic role `MODE` with the argument `PLANE` filling it, is to be realized by a modifier.

4.6 Encapsulation of syntactic consequences

All syntactic information should be encapsulated within resources and transparent to the algorithm. This includes the identification of arguments, including their placement with respect to the realization. Another example of a syntactic consequence is the presence of additional syntactic material required by the lexical item in the particular syntactic configuration. The verb *found* in the active configuration, as in (4a), does not require any additional syntactic material. On the other hand, the noun *founding* in the configuration with prepositional phrases headed by *of* and *by*, as in (4b), may be said to require the use of the prepositions. The resources for *found* and *founding* are shown in Fig. 6. Encapsulation of such consequences allows us to avoid special mechanisms to keep track of and enforce

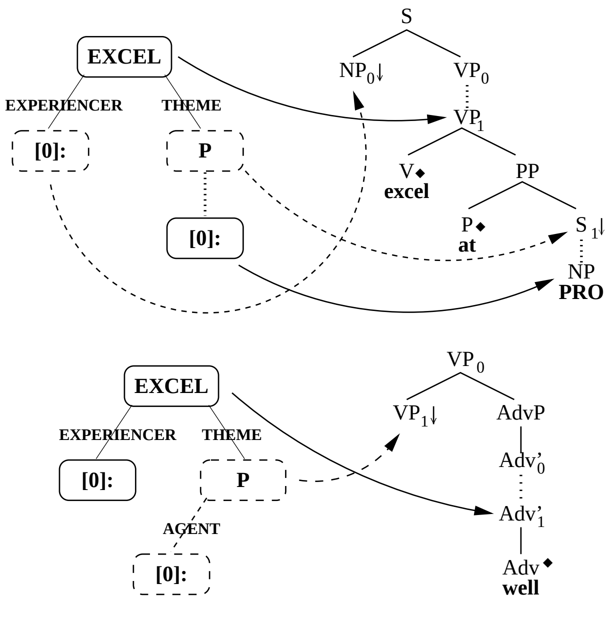

Figure 7: Two different resources for EXCEL

them for individual resources.

4.7 Syntactic rank and category

No assumptions are made about the realization of a piece of input semantics, including its syntactic rank and category. For instance, the predicate EXCEL can be realized by the verb *excel*, the adverb *well*, and the adjective *good*, as illustrated in (6a-6c). The processing is the same: a resource is selected and any argument realizations are attached to the resource.

Fig. 7 shows a resource for the predicate EXCEL realized by the verb *excel*. What is interesting about this case is that the DSG formalism we chose allows us to encapsulate the PRO in the subject position of the complement as a syntactic consequence of the verb *excel* in this configuration. The other resource for EXCEL shown in Fig. 7 is unusual in that the predicate is realized by an adverb, *well*. Note the link between the uninstantiated theme on the semantic side and the position for its corresponding syntactic realization, the substitution node VP_1[2].

Suppose the semantic input underlying (6a-6c) is as given in Fig. 8 and the *well* resource in Fig. 7 is selected to realize the top of the semantics. The matching in step 1 of our algorithm determines that the subtree of the input rooted at TEACH must be recursively realized. The realization of this subtree yields *Barbara teaches*. Because of the link between the theme of EXCEL and the VP_1 node of *well*, the realization *Barbara teaches* is substituted to the VP_1 node of *well*. This is a more complex substitution than in regular TAG (where the substitution node is identified with the root of the argument realization), and is equivalent to the adjunction of *well* to *Barbara teaches*. In DSG, we are able to treat structures such as the *well* structure as initial and not auxiliary, as TAG would. Thus, argument realizations are combined with all structures in a uniform fashion.

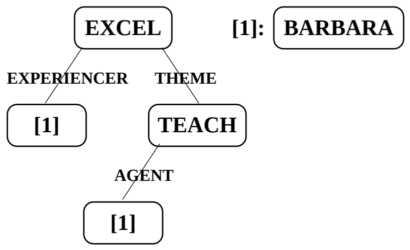

Figure 8: The semantics underlying (6a-6c)

4.8 Grammatical forms

As discussed before, grammatical forms themselves can realize a piece of semantics. For instance, the imperative syntactic form realizes a request or a command to the listener, as shown in Fig. 9. Likewise, the wh-question form realizes a request to identify, also shown in Fig. 9. In our system, whether the realization has any lexical items is not relevant.

4.9 The role of DSG

We believe that the choice of the DSG formalism plays a crucial role in maintaining our simple methodology. Like TAG, DSG allows capturing syntactic consequences in one elementary structure. DSG, however, allows even greater flexibility in what is included in an elementary structure. Note that in DSG we may have non-immediate domination links between nodes of

[2]Also notice that the experiencer of EXCEL is considered realized by the *well* resource and coindexed with the agent of the theme of EXCEL, to be realized by a separate resource.

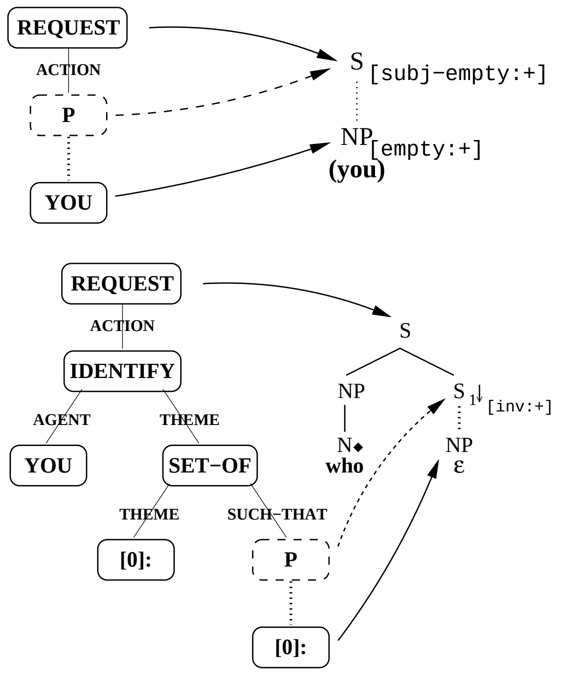

Figure 9: Two different resources for `REQUEST`

different syntactic categories (e.g., between the `S` and `NP` in Fig. 9 and also in the *excel at* structure in Fig. 7). DSG also allows uniform treatment of complementation and modification using the operations of substitution (regardless of the realization of the predicate, e.g., the structures in Fig. 7) and adjunction, respectively.

5 Conclusions

Although we only consider paraphrases with the same semantics, there is still a wide variety of expression which poses challenges to any generation system. In overcoming those challenges and generating in a simple manner in our architecture, our lexico-grammatical resources play an important role in each phase of generation. Encapsulation of syntactic consequences within elementary syntactic structures keeps our methodology modular. Whatever those consequences, often very different for different paraphrases, generation always proceeds in the same manner.

Both the algorithm and the constraints on our lexico-grammatical resources place significant demands on the grammatical formalism used for the architecture. We find that the DSG formalism meets those demands well.

References

John Bateman. 1997. Enabling technology for multilingual natural language generation: the KPML development environment. *Natural Language Engineering*, 3(1):15–55.

Bonnie J. Dorr. 1993. Interlingual machine translation: a parametrized approach. *Artificial Intelligence*, 63(1):429–492.

Michael Elhadad, Kathleen McKeown, and Jacques Robin. 1997. Floating constraints in lexical choice. *Computational Intelligence*, 23:195–239.

Raymond Kozlowski, Kathleen F. McCoy, and K. Vijay-Shanker. 2002. Selectional restrictions in natural language sentence generation. In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics, and Informatics (SCI'02).*

Raymond Kozlowski. 2002a. Driving multilingual sentence generation with lexico-grammatical resources. In *Proceedings of the Second International Natural Language Generation Conference (INLG'02) - Student Session.*

Raymond Kozlowski. 2002b. DSG/TAG - An appropriate grammatical formalism for flexible sentence generation. In *Proceedings of the Student Research Workshop at the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02).*

Nicolas Nicolov, Chris Mellish, and Graeme Ritchie. 1995. Sentence Generation from Conceptual Graphs. In *Proceedings of the 3rd International Conference on Conceptual Structures (ICCS'95).*

Owen Rambow, K. Vijay-Shanker, and David Weir. 2001. D-Tree Substitution Grammars. *Computational Linguistics*, 27(1):87–122.

Stuart M. Shieber and Yves Schabes. 1990. Synchronous Tree-Adjoining Grammars. In *Proceedings of the 13th International Conference on Computational Linguistics.*

Stuart M. Shieber, Gertjan van Noord, Fernando C. N. Pereira, and Robert C. Moore. 1990. Semantic-Head-Driven Generation. *Computational Linguistics*, 16(1):30–42.

Manfred Stede. 1999. *Lexical semantics and knowledge representation in multilingual text generation.* Kluwer Academic Publishers, Boston.

Matthew Stone and Christine Doran. 1997. Sentence Planning as Description Using Tree Adjoining Grammar. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97).*