Approximate Generation from Non-Hierarchical Representations

Nicolas Nicolov,* Chris Mellish, Graeme Ritchie
Dept of AI, Univ. of Edinburgh
80 South Bridge, Edinburgh EH1 1HN
{*nicolas*, *chrism*, *graeme*}@*aisb.ed.ac.uk*

Abstract

This paper presents a technique for sentence generation. We argue that the input to generators should have a non-hierarchical nature. This allows us to investigate a more general version of the sentence generation problem where one is not pre-committed to a choice of the syntactically prominent elements in the initial semantics. We also consider that a generator can happen to convey more (or less) information than is originally specified in its semantic input. In order to constrain this approximate matching of the input we impose additional restrictions on the semantics of the generated sentence. Our technique provides flexibility to address cases where the entire input cannot be precisely expressed in a single sentence. Thus the generator does not rely on the strategic component having linguistic knowledge. We show clearly how the semantic structure is declaratively related to linguistically motivated syntactic representation.

1 Introduction

Natural language generation is the process of realising communicative intentions as text (or speech). The generation task is standardly broken down into the following processes: content determination (what is the meaning to be conveyed), sentence planning[1] (chunking the meaning into sentence sized units, choosing words), surface realisation (determining the syntactic structure), morphology (inflection of words), synthesising speech or formatting the text output.

In this paper we address aspects of sentence planning (how content words are chosen but not how the semantics is chunked in units realisable as sentences) and surface realisation (how syntactic structures are computed). We thus discuss what in the literature is sometimes referred to as tactical generation, that is "how to say it"—as opposed to strategic generation—"what to say". We look at ways of realising a non-hierarchical semantic representation as a sentence, and explore the interactions between syntax and semantics.

Before giving a more detailed description of our proposals first we motivate the non-hierarchical nature of the input for sentence generators and review some approaches to generation from non-hierarchical representations—semantic networks (Section 2). We proceed with some background about the grammatical framework we will employ—D-Tree Grammars (Section 3) and after describing the knowledge sources available to the generator (Section 4) we present the generation algorithm (Section 5). This is followed by a step by step illustration of the generation of one sentence (Section 6). We then discuss further semantic aspects of the generation (Section 7) and the implementation (Section 8). We conclude with a discussion of some issues related to the proposed technique (Section 9).

2 Generation from Non-Hierarchical Representations

The input for generation systems varies radically from system to system. Many generators expect their input to be cast in a tree-like notation which enables the actual systems to assume that nodes higher in the semantic structure are more prominent than lower nodes. The semantic representations used are variations of a predicate with its arguments. The predicate is realised as the main verb of the sentence and the

*Supported by Faculty of Science and Engineering Scholarship 343 EE06006 at the University of Edinburgh.

[1]Note that this does not involve planning mechanisms!

arguments are realised as complements of the main verb—thus the control information is to a large extent encoded in the tree-like semantic structure. Unfortunately, such dominance relationships between nodes in the semantics often stem from language considerations and are not always preserved across languages. Moreover, if the semantic input comes from other applications, it is hard for these applications to determine the most prominent concepts because linguistic knowledge is crucial for this task. The tree-like semantics assumption leads to simplifications which reduce the paraphrasing power of the generator (especially in the context of multilingual generation).[2] In contrast, the use of a non-hierarchical representation for the underlying semantics allows the input to contain as few language commitments as possible and makes it possible to address the generation strategy from an unbiased position. We have chosen a particular type of a non-hierarchical knowledge representation formalism, conceptual graphs [24], to represent the input to our generator. This has the added advantage that the representation has well defined deductive mechanisms. A graph is a set of concepts connected with relations. The types of the concepts and the relations form generalisation lattices which also help define a subsumption relation between graphs. Graphs can also be embedded within one another. The counterpart of the unification operation for conceptual graphs is maximal join (which is non-deterministic). Figure 1 shows a simple conceptual graph which does not have cycles. The arrows of the conceptual relations indicate the domain and range of the relation and do not impose a dominance relationship.

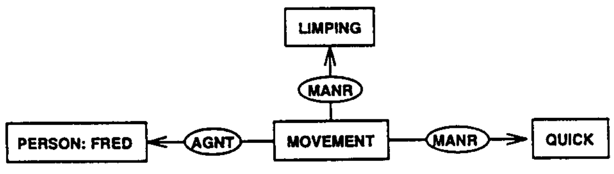


Figure 1: A simple conceptual graph

The use of semantic networks in generation is not new [21, 18]. Two main approaches have been employed for generation from semantic networks: *utterance path traversal* and *incremental consumption*. An utterance path is the sequence of nodes and arcs that are traversed in the process of mapping a graph to a sentence. Generation is performed by finding a cyclic path in the graph which visits each node at least once. If a node is visited more than once, grammar rules determine when and how much of its content will be uttered [23]. Under the second approach, that of incremental consumption, generation is done by gradually relating (consuming) pieces of the input semantics to linguistic structure [3, 13]. Such covering of the semantic structure avoids some of the limitations of the utterance path approach and is also the general mechanism we have adopted (we do not rely on the directionality of the conceptual relations *per se*—the primitive operation that we use when consuming pieces of the input semantics is maximal join which is akin to pattern matching). The borderline between the two paradigms is not clear-cut. Some researchers [22] are looking at finding an appropriate sequence of expansions of concepts and reductions of subparts of the semantic network until all concepts have realisations in the language. Others assume all concepts are expressible and try to substitute syntactic relations for conceptual relations [2].

Other work addressing surface realisation from semantic networks includes: generation using Meaning-Text Theory [6], generation using the SNePS representation formalism [19], generation from conceptual dependency graphs [26]. Among those that have looked at generation with conceptual graphs are: generation using Lexical Conceptual Grammar [15], and generating from CGs using categorial grammar in the domain of technical documentation [25].

This work improves on existing generation approaches in the following respects: *(i)* Unlike the majority of generators this one takes a non-hierarchical (logically well defined) semantic representation as its input. This allows us to look at a more general version of the realisation problem which in turn has direct ramifications for the increased paraphrasing power and usability of the generator; *(ii)* Following Nogier & Zock [14], we take the view that lexical choice is essentially (pattern) matching, but unlike them we assume that the meaning representation may not be entirely consumed at the end of the generation process. Our generator uses a notion of approximate matching and can happen to con-

[2]The tree-like semantics imposes some restrictions which the language may not support.

vey more (or less) information than is originally specified in its semantic input. We have a principled way to constrain this. We build the corresponding semantics of the generated sentence and aim for it to be as close as possible to the input semantics. *(i)* and *(ii)* thus allow for the input to come from a module that need not have linguistic knowledge. *(iii)* We show how the semantics is systematically related to syntactic structures in a declarative framework. Alternative processing strategies using the same knowledge sources can therefore be envisaged.

3 D-Tree Grammars

Our generator uses a particular syntactic theory—D-Tree Grammar (DTG) which we briefly introduce because the generation strategy is influenced by the linguistic structures and the operations on them.

D-Tree Grammar (DTG) [16] is a new grammar formalism which arises from work on Tree-Adjoining Grammars (TAG) [7]. In the context of generation, TAGs have been used in a number of systems MUMBLE [10], SPOKESMAN [11], WIP [27], the system reported in [9], the first version of PROTECTOR [12], and recently SPUD (by Stone & Doran). In the area of grammar development TAG has been the basis of one of the largest grammars developed for English [4]. Unlike TAGs, DTGs provide a uniform treatment of complementation and modification at the syntactic level. DTGs are seen as attractive for generation because a close match between semantic and syntactic operations leads to simplifications in the overall generation architecture. DTGs try to overcome the problems associated with TAGs while remaining faithful to what is seen as the key advantages of TAGs [7]: the extended domain of locality over which syntactic dependencies are stated and function argument structure is captured.

DTG assumes the existence of elementary structures and uses two operations to form larger structures from smaller ones. The elementary structures are tree descriptions[3] which are trees in which nodes are linked with two types of links: domination links (d-links) and immediate domination links (i-links) expressing (reflexive) domination and immediate domination relations between nodes. Graphically we will use a dashed line to indicate a d-link (see Figure 2). D-trees allow us to view the operations for composing trees as monotonic. The two combination operations that DTG uses are subsertion and sister-adjunction.

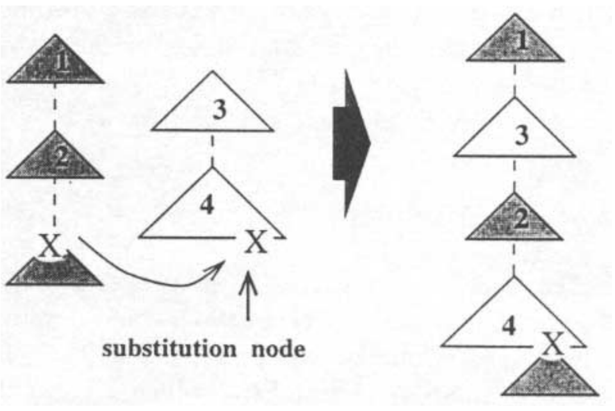


Figure 2: Subsertion

Subsertion. When a d-tree α is subserted into another d-tree β, a component[4] of α is substituted at a frontier nonterminal node (a substitution node) of β and all components of α that are above the substituted component are inserted into d-links above the substituted node or placed above the root node of β. It is possible for components above the substituted node to drift arbitrarily far up the d-tree and distribute themselves within domination links, or above the root, in any way that is compatible with the domination relationships present in the substituted d-tree. In order to constrain the way in which the non-substituted components can be interspersed DTG uses subsertion-insertion constraints which explicitly specify what components from what trees can appear within a certain d-links. Subsertion as it is defined as a non-deterministic operation. Subsertion can model both adjunction and substitution in TAG .

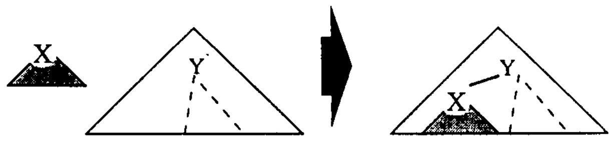


Figure 3: Sister-adjunction

Sister-adjunction. When a d-tree α is sister-adjoined at a node η in a d-tree β the com-

[3]called d-trees hence the name of the formalism.

[4]a subtree which contains only i-links.

posed d-tree γ results from the addition to β of α as a new leftmost or rightmost sub-d-tree below η. Sister-adjunction involves the addition of exactly one new immediate domination link. In addition several sister-adjunctions can occur at the same node. Sister-adjoining constraints associated with nodes in the d-trees specify which other d-trees can be sister-adjoined at this node and whether they will be right- or left-sister-adjoined.

For more details on DTGs see [16].

4 Knowledge Sources

The generator assumes it is given as input an input semantics (*InputSem*) and 'boundary' constraints for the semantics of the generated sentence (*BuiltSem* which in general is different from *InputSem*[5]). The boundary constraints are two graphs (*UpperSem* and *LowerSem*) which convey the notion of the least and the most that should be expressed. So we want *BuiltSem* to satisfy: *LowerSem* $\leq$ *BuiltSem* $\leq$ *UpperSem*.[6] If the generator happens to introduce more semantic information by choosing a particular expression, *LowerSem* is the place where such additions can be checked for consistency. Such constraints on *BuiltSem* are useful because in general *InputSem* and *BuiltSem* can happen to be incomparable (neither one subsumes the other). In a practical scenario *LowerSem* can be the knowledge base to which the generator has access minus any contentious bits. *UpperSem* can be the minimum information that necessarily has to be conveyed in order for the generator to achieve the initial communicative intentions.

The goal of the generator is to produce a sentence whose corresponding semantics is as close as possible to the input semantics, i.e., the realisation adds as little as possible extra material and misses as little as possible of the original input. In generation similar constraints have been used in the generation of referring expressions where the expressions should not be too general so that discriminatory power is not lost and not too specific so that the referring expression is in a sense minimal. Our model is a generalisation of the paradigm presented in [17] where issues of mismatch in lexical choice are discussed. We return to how *UpperSem* and *LowerSem* are actually used in Section 7.

4.1 Mapping rules

Mapping rules state how the semantics is related to the syntactic representation. We do not impose any intrinsic directionality on the mapping rules and view them as declarative statements. In our generator a mapping rule is represented as a d-tree in which certain nodes are annotated with semantic information. Mapping rules are a mixed syntactic-semantic representation. The nodes in the syntactic structure will be feature structures and we use unification to combine two syntactic nodes. The semantic annotations of the syntactic nodes are either conceptual graphs or *instructions* indicating how to compute the semantics of the syntactic node from the semantics of the daughter syntactic nodes. Graphically we use dotted lines to show the coreference between graphs (or concepts). Each graph appearing in the rule has a single node ("the semantic head") which acts as a root (indicated by an arrow in Figure 4). This hierarchical structure is imposed by the rule, and is not part of the semantic input. Every mapping rule has associated applicability semantics which is used to license its application. The applicability semantics can be viewed as an evaluation of the semantic instruction associated with the top syntactic node in the tree description.

Figure 4 shows an example of a mapping rule. The applicability semantics of this mapping rule is: $\boxed{\text{ANIMATE}} \leftarrow (\text{AGNT}) - \boxed{\odot\ \text{ACTION}} - (\text{OBJ}) \rightarrow \boxed{\text{ENTITY}}$. If this structure matches part of the input semantics (we explain more precisely what we mean by matching later on) then this rule can be triggered (if it is syntactically appropriate—see Section 5). The internal generation goals (shaded areas) express the following: (1) generate $\boxed{\text{ACTION}}$ as a verb and subsert (substitute,attach) the verb's syntactic structure at the $V\diamond$ node; (2) generate $\boxed{\text{ANIMATE}}$ as a noun phrase and subsert the newly built structure at $NP0$; and (3) generate $\boxed{\text{ENTITY}}$ as another noun phrase and subsert the newly built struc-

[5]This can come about from a mismatch between the input and the semantic structures expressible by the generator.

[6]The notation $G_1 \leq G_2$ means that G_1 is subsumed by G_2. We consider *UpperSem* to be a generalisation of *BuiltSem* and *LowerSem* a specialisation of *BuiltSem* (in terms of the conceptual graphs that represent them).

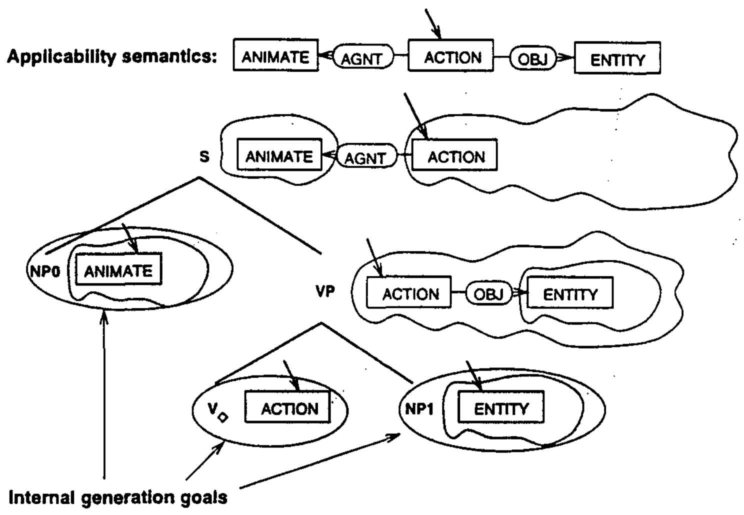


Figure 4: A mapping rule for transitive constructions

ture at $NP1$. The newly built structures are also mixed syntactic-semantic representations (annotated d-trees) and they are incorporated in the mixed structure corresponding to the current status of the generated sentence.

5 Sentence Generation

In this section we informally describe the generation algorithm. In Figure 5 and later in Figure 8, which illustrate some semantic aspects of the processing, we use a diagrammatic notation to describe semantic structures which are actually encoded using conceptual graphs.

The input to the generator is *InputSem*, *LowerSem*, *UpperSem* and a mixed structure, *Partial*, which contains a syntactic part (usually just one node but possibly something more complex) and a semantic part which takes the form of semantic annotations on the syntactic nodes in the syntactic part. Initially *Partial* represents the syntactic-semantic correspondences which are imposed on the generator.[7] It has the format of a mixed structure like the representation used to express mapping rules (Figure 4). Later during the generation *Partial* is enriched and at any stage of processing it represents the current syntactic-semantic correspondences.

We have augmented the DTG formalism so that the semantic structures associated with syntactic nodes will be updated appropriately during the subsertion and sister-adjunction operations. The stages of generation are: (1) building an initial skeletal structure; (2) attempting to consume as much as possible of the semantics uncovered in the previous stage; and (3) converting the partial syntactic structure into a complete syntactic tree.

5.1 Building a skeletal structure

Generation starts by first trying to find a mapping rule whose semantic structure matches[8] part of the initial graph and whose syntactic structure is compatible with the goal syntax (the syntactic part of *Partial*). If the initial goal has a more elaborate syntactic structure and requires parts of the semantics to be expressed as certain syntactic structures this has to be respected by the mapping rule. Such an initial mapping rule will have a syntactic structure that will provide the skeleton syntax for the sentence. If Lexicalised DTGis used as the base syntactic formalism at this stage the mapping rule will introduce the head of the sentence structure—the main verb. If the rule has internal generation goals then these are explored recursively (possibly via an agenda—we will ignore here the

[7]In dialogue and question answering, for example, the syntactic form of the generated sentence may be constrained.

[8]via the maximal join operation. Also note that the arcs to/from the conceptual relations do not reflect any directionality of the processing—they can be 'traversed'/accessed from any of the nodes they connect.

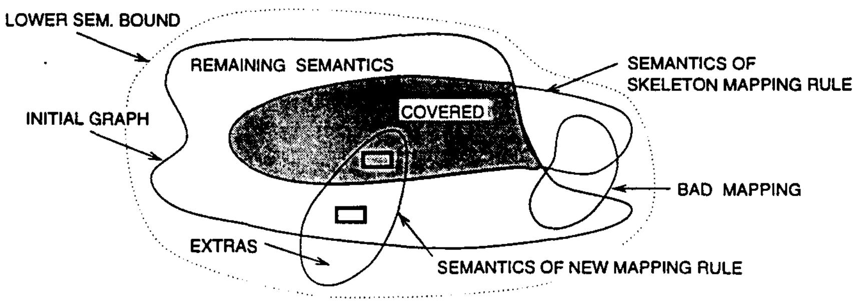


Figure 5: Covering the remaining semantics with mapping rules

issue of the order in which internal generation goals are executed). Because of the minimality of the mapping rule, the syntactic structure that is produced by this initial stage is very basic—for example only obligatory complements are considered. Any mapping rule can introduce additional semantics and such additions are checked against the lower semantic bound. When applying a mapping rule the generator keeps track of how much of the initial semantic structure has been covered/consumed. Thus at the point when all internal generation goals of the first (skeletal) mapping rule have been exhausted the generator knows how much of the initial graph remains to be expressed.

5.2 Covering the remaining semantics

In the second stage the generator aims to find mapping rules in order to cover most of the remaining semantics (see Figure 5) . The choice of mapping rules is influenced by the following criteria:

Connectivity: The semantics of the mapping rule has to match (cover) part of the covered semantics and part of the remaining semantics.

Integration: It should be possible to incorporate the semantics of the mapping rule into the semantics of the current structure being built by the generator.

Realisability: It should be possible to incorporate the partial syntactic structure of the mapping rule into the current syntactic structure being built by the generator.

Note that the connectivity condition restricts the choice of mapping rules so that a rule that matches part of the remaining semantics and the extra semantics added by previous mapping rules cannot be chosen (e.g., the "bad mapping" in Figure 5). While in the stage of fleshing out the skeleton sentence structure (Section 5.1) the syntactic integration involves subsertion, in the stage of covering the remaining semantics it is sister-adjunction that is used. When incorporating semantic structures the semantic head has to be preserved—for example when sister-adjoining the d-tree for an adverbial construction the semantic head of the top syntactic node has to be the same as the semantic head of the node at which sister-adjunction is done. This explicit marking of the semantic head concepts differs from [20] where the semantic head is a PROLOG term with exactly the same structure as the input semantics.

5.3 Completing a derivation

In the preceding stages of building the skeletal sentence structure and covering the remaining semantics, the generator is mainly concerned with consuming the initial semantic structure. In those processes, parts of the semantics are mapped onto partial syntactic structures which are integrated and the result is still a partial syntactic structure. That is why a final step of "closing off" the derivation is needed. The generator tries to convert the partial syntactic structure into a complete syntactic tree. A morphological post-processor reads the leaves of the final syntactic tree and inflects the words.

6 Example

In this section we illustrate how the algorithm works by means of a simple example. Suppose

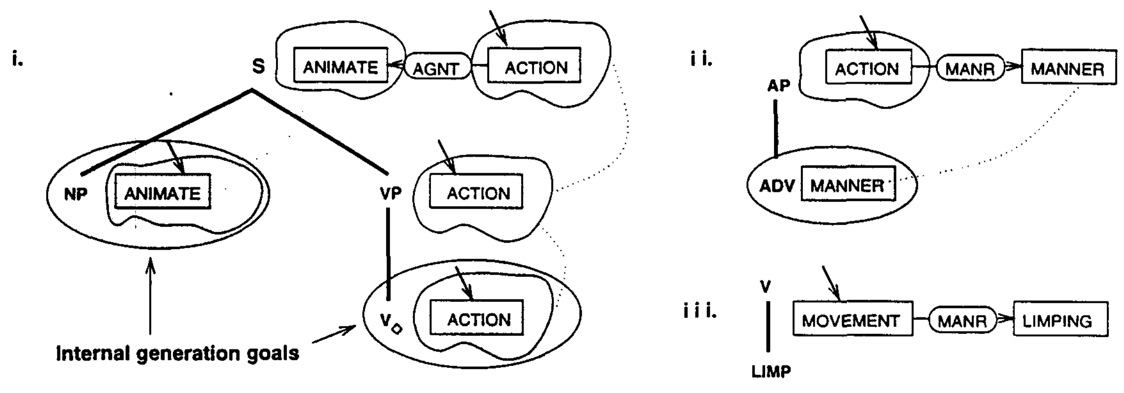


Figure 6: Mapping rules

we start with an initial semantics as given in Figure 1. This semantics can be expressed in a number of ways: *Fred limped quickly*, *Fred hurried with a limp*, *Fred's limping was quick*, *The quickness of Fred's limping* ..., etc. Here we show how the first paraphrase is generated.

In the stage of building the skeletal structure the mapping rule *(i)* in Figure 6 is used. Its internal generation goals are to realise the instantiation of $\boxed{\text{ACTION}}$ (which is $\boxed{\text{MOVEMENT}}$) as a verb and similarly $\boxed{\text{PERSON:FRED}}$ as a noun phrase. The generation of the subject noun phrase is not discussed here. The main verb is generated using the terminal mapping rule[9] *(iii)* in Figure 6.[10] The skeletal structure thus generated is *Fred limp(ed)*. (see *(i)* in Figure 7).

An interesting point is that although the internal generation goal for the verb referred only to the concept $\boxed{\text{MOVEMENT}}$ in the initial semantics, all of the information suggested by the terminal mapping rule *(iii)* in Figure 6 is consumed. We will say more about how this is done in Section 7.

At this stage the only concept that remains to be consumed is $\boxed{\text{QUICK}}$. This is done in the stage of covering the remaining semantics when the mapping rule *(ii)* is used. This rule has an internal generation goal to generate the instantiation of $\boxed{\text{MANNER}}$ as an adverb, which yields *quickly*. The structure suggested by this rule has to be integrated in the skeletal structure. On the syntactic side this is done using sister-adjunction. The final mixed syntactic-semantic structure is shown on the right in Figure 7. In the syntactic part of this structure we have no domination links. Also all of the input semantics has been consumed. The semantic annotations of the S and VP nodes are instructions about how the graphs/concepts of their daughters are to be combined. If we evaluate in a bottom up fashion the semantics of the S node, we will get the same result as the input semantics in Figure 1. After morphological post-processing the result is *Fred limped quickly*. An alternative paraphrase like *Fred hurried with a limp*[11] can be generated using a lexical mapping rule for the verb *hurry* which groups $\boxed{\text{MOVEMENT}}$ and $\boxed{\text{QUICK}}$ together and a another mapping rule expressing $\boxed{\text{LIMPING}}$ as a *PP*. To get both paraphrases would be hard for generators relying on hierarchical representations.

7 Matching the applicability semantics of mapping rules

Matching of the applicability semantics of mapping rules against other semantic structures occurs in the following cases: when looking for a skeletal structure; when exploring an internal generation goal; and when looking for mapping rules in the phase of covering the remaining semantics. During the exploration of internal generation goals the applicability semantics of a mapping rule is matched against the semantics of an internal generation goal. We assume that

[9]Terminal mapping rules are mapping rules which have no internal generation goals and in which all terminal nodes of the syntactic structure are labelled with terminal symbols (lexemes).

[10]In Lexicalised DTGs the main verbs would be already present in the initial trees.

[11]Our example is based on Iordanskaja *et al.*'s notion of maximal reductions of a semantic net (see [6, page 300]). It is also similar to the example in [14].

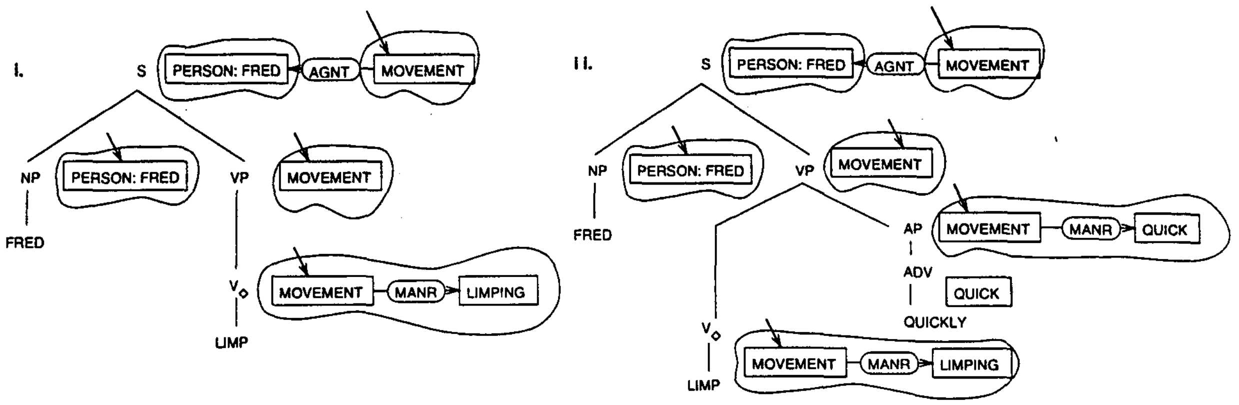
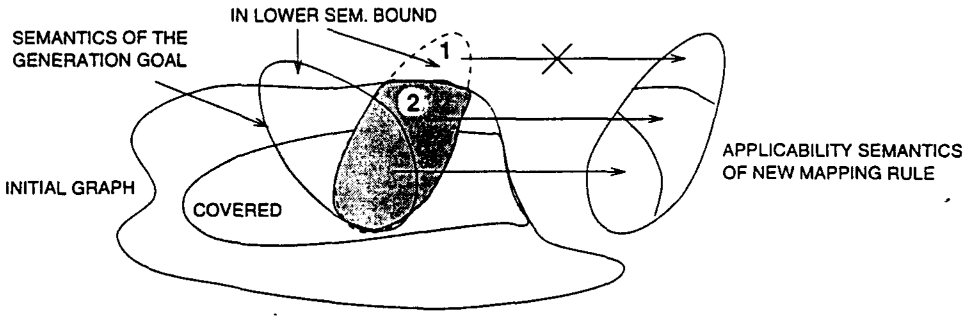


Figure 7: Skeletal structure and final structure



IN LOWER SEM. BOUND
SEMANTICS OF THE GENERATION GOAL
1
2
INITIAL GRAPH
COVERED
APPLICABILITY SEMANTICS OF NEW MAPPING RULE

Figure 8: Interactions involving the applicability semantics of a mapping rule

the following conditions hold:

1. The applicability semantics of the mapping rule can be maximally joined with the goal semantics.
2. Any information introduced by the mapping rule that is more specialised than the goal semantics (additional concepts/relations, further type instantiation, etc.) must be within the lower semantic bound (*LowerSem*). If this additional information is within the input semantics, then information can propagate from the input semantics to the mapping rule (the shaded area 2 in Figure 8). If the mapping rule's semantic additions are merely in *LowerSem*, then information cannot flow from *LowerSem* to the mapping rule (area 1 in Figure 8).

Similar conditions hold when in the phase of covering the remaining semantics the applicability semantics of a mapping rule is matched against the initial semantics. This way of matching allows the generator to convey only the information in the original semantics and what the language forces one to convey even though more information might be known about the particular situation.

In the same spirit after the generator has consumed/expressed a concept in the input semantics the system checks that the lexical semantics of the generated word is more specific than the corresponding concept (if there is one) in the upper semantic bound.

8 Implementation

We have developed a sentence generator called PROTECTOR (approximate PROduction of TExts from Conceptual graphs in a declaraTive framewORk). PROTECTOR is implemented in LIFE [1]. The syntactic coverage of the generator is influenced by the XTAG system (the first version of PROTECTOR in fact used TAGs). By using DTGs we can use most of the analysis of XTAG while the generation algorithm is simpler. We are in a position to express subparts of the input semantics as different syntactic categories as appropriate for the current generation goal (e.g., VPs and nominalisations). The syntactic coverage of PROTECTOR includes: intransitive, transitive, and ditransitive verbs, topicalisation, verb particles, passive, sentential complements, control constructions, relative clauses, nominalisations and a variety of idioms. On backtracking PROTECTOR returns all solutions. We are also looking at the advantages that our approach offers for multilingual generation.

9 Discussion

During generation it is necessary to find appropriate mapping rules. However, at each stage a number of rules might be applicable. Due to possible interactions between some rules the generator may have to explore different choices before actually being able to produce a sentence. Thus, generation is in essence a search problem. In order to guide the search a number of heuristics can be used. In [14] the number of matching nodes has been used to rate different matches, which is similar to finding maximal reductions in [6]. Alternatively a notion of semantic distance [5] might be employed. In PROTECTOR we will use a much more sophisticated notion of what it is for a conceptual graph to match better the initial semantics than another graph. This captures the intuition that the generator should try to express as much as possible from the input while adding as little as possible extra material.

We use instructions showing how the semantics of a mother syntactic node is computed because we want to be able to correctly update the semantics of nodes higher than the place where substitution or adjunction has taken place—i.e., we want to be able to propagate the substitution or adjunction semantics up the mixed structure whose backbone is the syntactic tree.

We also use a notion of headed conceptual graphs, i.e., graphs that have a certain node chosen as the semantic head. The initial semantics need not be marked for its semantic head. This allows the generator to choose an appropriate (for the natural language) perspective. The notion of semantic head and their connectivity is a way to introduce a hierarchical view on the semantic structure which is dependent on the language. When matching two conceptual graphs we require that their heads be the same. This reduces the search space and speeds up the generation process.

Our generator is not coherent or complete (i.e., it can produce sentences with more general/specific semantics than the input semantics). We try to generate sentences whose semantics is as close as possible to the input in the sense that they introduce little extra material and leave uncovered a small part of the input semantics. We keep track of more structures as the generation proceeds and are in a position to make finer distinctions than was done in previous research. The generator never produces sentences with semantics which is more specific than the lower semantic bound which gives some degree of coherence. Our generation technique provides flexibility to address cases where the entire input cannot be expressed in a single sentence by first generating a "best match" sentence and allowing the remaining semantics to be generated in a follow-up sentence.

Our approach can be seen as a generalisation of semantic head-driven generation [20]—we deal with a non-hierarchical input and non-concatenative grammars. The use of Lexicalised DTG means that the algorithm in effect looks first for a syntactic head. This aspect is similar to syntax-driven generation [8].

The algorithm has to be checked against more linguistic data and we intend to do more work on additional control mechanisms and also using alternative generation strategies using knowledge sources free from control information. To this end we have explored aspects of a new semantic-indexed chart generation which also allows us to rate intermediate results using syntactic as well as semantic preferences. Syntactic/stylistic preferences are helpful in cases where the semantics of two paraphrases are the same. One such instance of use of syntactic preferences is avoiding (giving lower rating to) heavy constituents in split verb particle constructions. Thus, the generator finds all possible solutions producing the "best" first.

10 Conclusion

We have presented a technique for sentence generation from conceptual graphs. The use of a non-hierarchical representation for the semantics and approximate semantic matching increases the paraphrasing power of the generator and enables the production of sentences with radically different syntactic structure due to alternative ways of grouping concepts into words. This is particularly useful for multilingual generation and in practical generators which are fed input from non linguistic applications. The use of a syntactic theory (D-Tree Grammars) allows for the production of linguistically motivated syntactic structures which will pay off in terms of better coverage of the language and overall maintainability of the generator. The syntactic theory also affects the processing—we have augmented the syntactic operations to account for the integration of the semantics. The generation architecture makes explicit the decisions that have to be taken and allows for experiments with different generation strategies using the same declarative knowledge sources.

References

[1] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3&4):195–234, 1993.

[2] F. Antonacci et al. Analysis and Generation of Italian Sentences. In T. Nagle, J. Nagle, L. Gerholz, and P. Eklund, editors, *Conceptual Structures: Current research and Practice*, pages 437–460. Ellis Horwood, 1992.

[3] M. Boyer and G. Lapalme. Generating paraphrases from meaning-text semantic networks. *Computational Intelligence*, 1(1):103–117, 1985.

[4] C. Doran et al. XTAG—A Wide Coverage Grammar for English. In *COLING'94*, pages 922–928, 1994.

[5] N. Foo et al. Semantic distance in conceptual graphs. In J. Nagle and T. Nagle, editors, *Fourth Annual Workshop on Conceptual Structures*, 1989.

[6] L. Iordanskaja et al. Lexical Selection and Paraphrase in a Meaning-Text Generation Model. In C.Paris, W.Swartout, and W.Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, pages 293-312. Kluwer Academic, 1991.

[7] A. Joshi. The Relevance of Tree Adjoining Grammar to Generatio n. In G. Kempen, editor, *Natural Language Generation*, pages 233–252. Kluwer Academic, 1987.

[8] E. König. Syntactic head-driven generation. In *COLING'94*, 475-481, Kyoto, 1994.

[9] K. F. McCoy, K. Vijay-Shanker, and G. Yang. A functional approach to generation with tag. In *30th Annual Meeting of ACL*, pages 48–55, 1992.

[10] D. McDonald and J. Pustejovsky. TAGs as a grammatical formalism for generation. In *23rd Annual Meeting of the ACL*, pages 94–103, 1985.

[11] M. Meteer. *The "Generation Gap": The Problem of Expressibility in Text Planning.* PhD thesis, Univ of Massachusetts, 1990. COINS TR 90-04.

[12] N. Nicolov, C. Mellish, and G. Ritchie. Sentence Generation from Conceptual Graphs. In G.Ellis, R.Levinson, W.Rich, and J.Sowa, editors, *Conceptual Structures: Applications, Implementation and Theory*, pages 74–88. LNAI 954, Springer, 1995. 3rd Int. Conf. on Conceptual Structures (ICCS'95), Santa Cruz, CA, USA.

[13] J.-F. Nogier. *Génération automatique de langage et graphs conceptuels.* Hermes, Paris, 1991.

[14] J.-F. Nogier and M. Zock. Lexical Choice as Pattern Matching. In T. Nagle, J. Nagle, L. Gerholz, and P. Eklund, editors, *Conceptual Structures: Current research and Practice*, pages 413–436. Ellis Horwood, 1992.

[15] J. Oh et al. NLP: Natural Language Parsers and Generators. In *1st Int. Workshop on PEIRCE: A Conceptual Graph Workbench*, pages 48–55, 1992.

[16] O. Rambow, K. Vijay-Shanker, and D. Weir. D-tree grammars. In *ACL*, 1995.

[17] E. Reiter. A new model of lexical choice for nouns. *Computational Intelligence*, 7(4):240–251, 1991. Special issue on Natural Language Generation.

[18] S. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *Computational Linguistics*, 2(8):12–25, 1982.

[19] S. Shapiro. The CASSIE projects: An approach to NL Competence. In *4th Portugese Conference on AI (EPIA-89)*. LNAI 390: 362–380, Springer, 1989.

[20] S. Shieber, G. van Noord, R. Moore, and F. Pereira. A semantic head-driven generation algorithm for unification-based formalisms. *Computational Linguistics*, 16(1):30–42, 1990.

[21] R. Simmons and J. Slocum. Generating English Discourse from Semantic Networks. *CACM*, 15(10):891–905, 1972.

[22] M. Smith, R. Garigliano, and R. Morgan. Generation in the LOLITA system: an engineering approach. In *7th Int. Workshop on Natural Language Generation*, pages 241–244, 1994.

[23] J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[24] J. F. Sowa. Conceptual graphs summary. In T. E. Nagle et al., editors, *Conceptual Structures: Current Research and Practice*, pages 3–51. Ellis Horwood, 1992.

[25] S. Svenberg. Representing Conceptual and Linguistic Knowledge for Multilingual Generation in a Technical Domain. In *7th Int. Workshop on Natural Language Generation*, pages 245–248, 1994.

[26] A. van Rijn. *Natural Language Communication between Man and Machine.* PhD thesis, Technical University Delft, 1991.

[27] W. Wahlster et al. WIP: the coordinated generation of multimodal presentations fro m a common representation. RR 91-08, DFKI, 1991.