Knowledge-Based Systems, 10(1997) 183-190

Automatic Induction of Rules for Text Simplification

R. Chandrasekar

Institute for Research in Cognitive Science
& Center for the Advanced Study of India
University of Pennsylvania, Philadelphia, PA 19104
mickeyc@linc.cis.upenn.edu[1]

B. Srinivas

Department of Computer & Information Sciences
University of Pennsylvania
Philadelphia, PA 19104
srini@linc.cis.upenn.edu

Long and complicated sentences pose various problems to many state-of-the-art natural language technologies. We have been exploring methods to automatically transform such sentences in order to make them simpler. These methods involve the use of a rule-based system, driven by the syntax of the text in the domain of interest. Hand-crafting rules for every domain is time-consuming and impractical. The paper describes an algorithm and an implementation by which generalized rules for simplification are automatically induced from annotated training material using a novel partial parsing technique which combines constituent structure and dependency information. The algorithm described in the paper employs example-based generalizations on linguistically-motivated structures.

1 The Need for Text Simplification

Long and complicated sentences pose various problems to many state-of-the-art natural language technologies. For example, in parsing, as sentences become syntactically more complex, the number of parses increases, and there is a greater likelihood of an incorrect parse. In machine translation, complex

[1] On leave from the National Centre for Software Technology, Gulmohar Cross Road No. 9, Juhu, Mumbai 400 049, India

sentences lead to increased ambiguity and potentially unsatisfactory translations. Complicated sentences can also lead to confusion in assembly manuals, user manuals or maintenance manuals for complex equipment. We have been exploring methods to automatically simplify complicated sentences [Chandrasekar, 1994; Chandrasekar et al, 1996]. Consider, for example, the following sentence:

Example 1: *The embattled Major government survived a crucial vote on coal pits closure as its last-minute concessions curbed the extent of Tory revolt over an issue that generated unusual heat in the House of Commons and brought the miners to London streets.*

Such sentences are not uncommon in newswire texts. Compare this with an equivalent (manually) simplified multi-sentence version:

Example 2: *The embattled Major government survived a crucial vote on coal pits closure. Its last-minute concessions curbed the extent of Tory revolt over the coal-mine issue. This issue generated unusual heat in the House of Commons. It also brought the miners to London streets.*

Most of the problems posed by complicated sentences are either eliminated or substantially reduced for the simplified version shown in (2). For instance, simpler sentences have fewer constituents, hence fewer ambiguities in identifying attachments and thus are parsed faster. Simplification would also be of great use in several areas of natural language processing such as machine translation, information retrieval and in applications where clarity of text is imperative. Of course, one may lose some nuances of meaning from the original text in the simplification process.

There has been interest in simplified English from companies such as Boeing and Xerox. Researchers at Boeing [Hoard et al, 1992], [Wojcik et al, 1993] have developed a Simplified English Checker. However, their focus is on carefully constraining the use of words in a specific domain, and in providing a tool to authors of machine maintenance/operation manuals to help them adhere to guidelines aimed at clear written communication. In contrast, our aim is to develop a system to (semi-)automatically simplify text from any domain.

The following is the outline of this paper. In Section 2, we present an architecture for simplification. The method used for analysis of input is discussed in Section 3 and supertags and LDA are described. In Section 4, we describe a method by which generalized rules are automatically induced from annotated training material of newspaper text in English. Section 5 describes how these rules are applied. We then discuss some issues pertaining to simplification in Section 6.

2 The Architecture of Simplification

We view simplification as a two stage process: analysis followed by transformation. The analysis stage provides a structural description of the input, and the transformation stage uses this representation for simplification. Our simplification system processes one sentence at a time. Discourse related issues are not considered.

The most obvious choice for the analysis stage is to use a full parser to obtain the complete structure of a sentence. If all the constituents of the sentence along with the dependency relations are given, simplification is very straightforward. However, it may not be possible to get a correct full parse of a sentence, especially if it is a complicated sentence which could benefit from simplification.

We have discussed two alternative approaches to analyzing text for simplification; [Chandrasekar, 1994] describes a using a finite state grammar approach while [Chandrasekar et al, 1996] describes a dependency based approach. We summarize the dependency based approach in the next section. Note that this approach is different from a full parsing approach in that a complete constituent structure is neither required nor created.

We define *articulation-points* to be those points where sentences may be split for simplification. Segments of a sentence between two articulation points may be extracted as simplified sentences. The nature of the segments delineated by the articulation points depends on the type of the structural analysis performed. If sentences are viewed just as linear strings of words, we could define articulation points to be, say, punctuation marks. If the words in the input are also tagged with part of speech information, we can split sentences based on the category information, for instance at relative pronouns. With part of speech information, subordinating and coordinating conjunctions may also be detected and used as articulation points. However, with just this information, the span of the subordinating/coordinating clause would be difficult to determine. On the other hand, if the sentence is annotated with phrasal bracketings, the beginnings and ends of phrases could also be articulation points.

For example, the sentence (3) with a relative clause, annotated with phrasal bracketing, can be simplified into two sentences as shown in Example (4), using a rule such as the one shown in Rule 1 that relies on skeletal phrasal structure and punctuation information.

Example 3: *[Talwinder Singh]:NP, who:RelPron masterminded:V [the 1984 Kanishka crash]:NP, [was killed]:V [in [a fierce two-hour encounter]:NP]:PP.*

Example 4: *Talwinder Singh was killed in a fierce two-hour encounter.*
Talwinder Singh masterminded the 1984 Kanishka crash.

Rule 1:
`W X:NP, RelPron Y, Z` $\rightarrow$ `W X:NP Z. X:NP Y.`

The rule is interpreted as follows. If a sentence starts with some segment W and a noun phrase (`X:NP`), and is then followed by a phrase of the form (`, RelPron Y ,`) followed by some (`Z`), where `Y` and `Z` are arbitrary sequences of words, then the sentence may be simplified into two sentences, namely the sequence (`W X`) followed by (`Z`), and the sequence (`X`) followed by (`Y`).

However, the rule shown above does not handle reduced relatives, such as the one in sentence (5).

Example 5: *[The creator of Air India, Mr. JRD Tata]:NP, [believes]:V*
[that]:COMP
[the airline]:NP, [known]:V [for [its on-board service]:NP]:PP,
[could return]:V [to [its old days of glory]:NP]:PP.

To solve such problems, we use rules based on a representation which combines dependency information with constituent structure, providing attachment and scope information. This representation is described in the next section.

We need a variety of rules to simplify text from any particular domain. However, hand-crafting simplification rules is time-consuming and not very practical. While some of the rules are likely to be common across domains, several are likely to be domain-specific. We ideally need a method to develop rules which can be easily induced for a new domain. In this paper, we present an algorithm and an implementation to automatically induce rules for simplification given an annotated aligned corpus of complex and simple text.

In addition to developing rules, we need gap-filling routines. For example, if we separate a relative clause from a sentence (as in example (4)), we must insert a copy of the head noun at the gap in the relative clause. The exact choice of the gap fillers is a complicated task based on a variety of pragmatic factors, and will not be discussed in this paper.

3 Analysis of Input

Our approach to the analysis stage of simplification uses rich syntactic information, based on a simple dependency representation provided by Lexicalized Tree Adjoining Grammar (LTAG) [Joshi, 1985], [Schabes et al, 1988] and uses the “supertagging” techniques described in [Joshi and Srinivas, 1994]. In this section, for convenience, we outline the basics of LTAG, supertagging and related ideas.

3.1 Lexicalized Tree-Adjoining Grammars

The basic elements of Lexicalized Tree-Adjoining Grammar (LTAG) [Joshi, 1985; Schabes et al, 1988; Kroch and Joshi, 1985] are called *elementary trees.* Each elementary tree is associated with at least one lexical item called the *anchor* of that tree. All the arguments of the anchor are realized as substitution or adjunction slots within an elementary tree. Thus an elementary tree serves as a complex description of the anchor and provides a domain of locality over which the anchor specifies syntactic and semantic (predicate-argument) constraints. Elementary trees are of two types: (a) *initial trees* (α trees in Figure 2) that represent non-recursive linguistic structures such as NPs and PPs; and (b) *auxiliary trees* (β trees in Figure 2) that represent recursive structures which are adjuncts to basic structure (e.g. relative clauses, sentential adjuncts, adverbials).

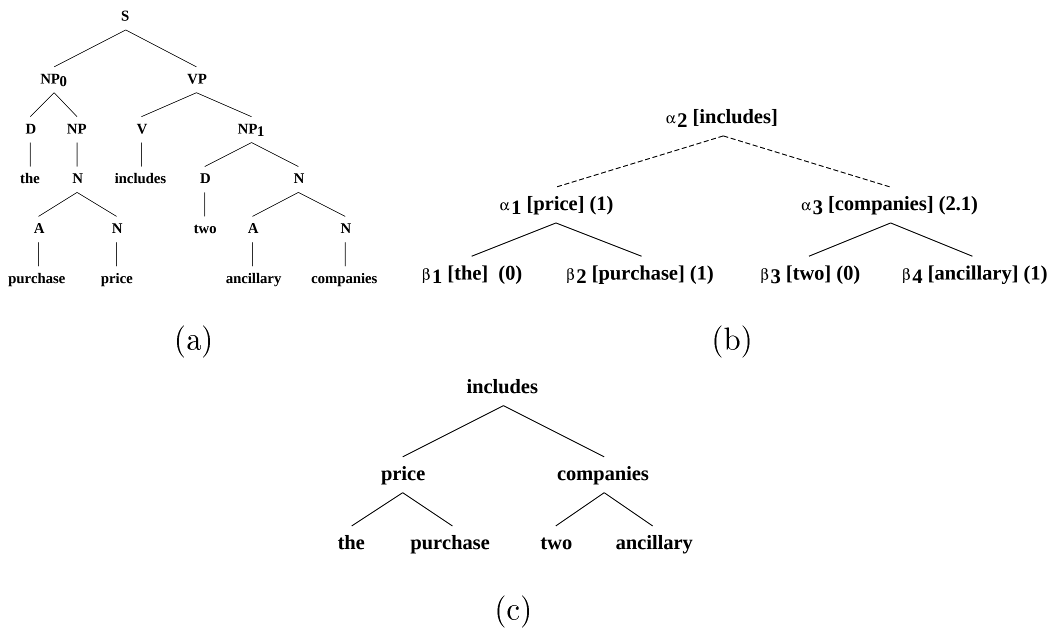


Fig. 1. (a): Derived tree (b): Derivation structure (c): Dependency tree for the sentence *the purchase price includes two ancillary companies*

Elementary trees are combined by two operations, *substitution* and *adjunction.* The result of combining the last row of elementary trees of Figure 2 is the *derived tree* (Figure 1(a)). But the more important structure in an LTAG parse is the *derivation tree* (Figure 1(b)) which represents the process of combining the elementary trees to yield a parse. The derivation tree can also be interpreted as a *dependency tree* (Figure 1(c)) with unlabeled arcs between words of the sentence.

A wide-coverage English grammar has been implemented in the LTAG framework. This grammar has been used to parse sentences from the Wall Street Journal, IBM manual and ATIS domains. A detailed description of this system and its performance results are presented in [Srinivas et al, 1996].

3.2 Supertags

The elementary trees of LTAG localize dependencies, including long distance dependencies, by requiring that all and only the dependent elements be present within the same tree. As a result of this localization, a lexical item may be (and almost always is) associated with more than one elementary tree, where each tree corresponds to a particular role played by the word. The example in Figure 2 illustrates the set of elementary trees assigned to each word of the sentence *the purchase price includes two ancillary companies*. We call these elementary trees *supertags*, since they contain more information (such as subcategorization and agreement information) than standard part-of-speech tags. Supertags for recursive and non-recursive constructs are labeled with βs and αs respectively.

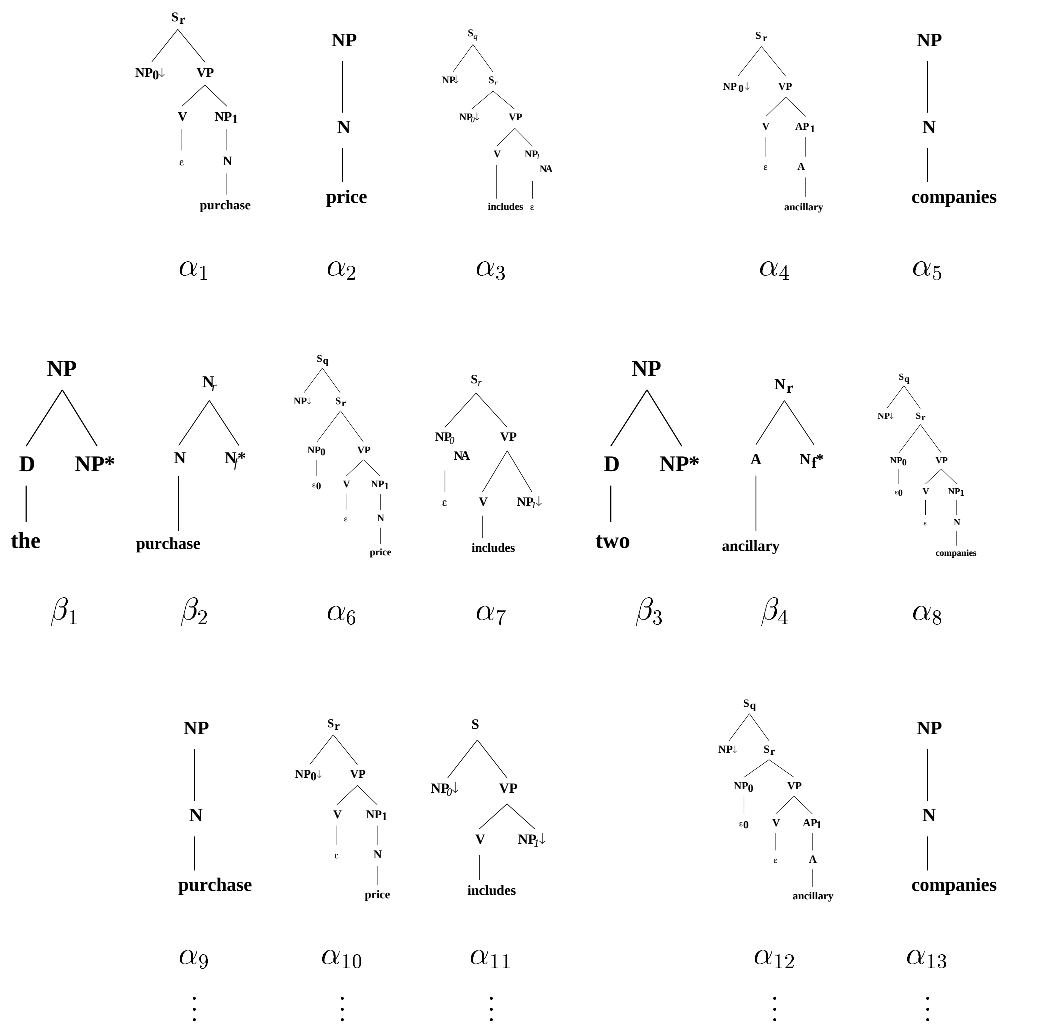
Parsing this lexicalized grammar can be viewed as a two-step process. The first step is to select the appropriate supertags for each word of the input and the second step is to combine the selected supertags with substitution and adjunction operations. We call the first step *supertagging*. Note that, as in standard part-of-speech disambiguation, supertagging could have been done by a parser. However, just as carrying out part-of-speech disambiguation prior to parsing makes the job of the parser much easier and therefore faster, supertagging substantially reduces the work of the parser.

The result of supertagging is almost a parse in the sense that the parser need 'only' link the individual structures to arrive at a complete parse. We present such a simple linking procedure (*a Lightweight Dependency Analyzer*) in Section 3.3. This method can also be used to parse sentence fragments where it is not possible to combine the disambiguated supertag sequence into a single structure.

3.2.1 Trigram Model for Supertagging

The task of supertagging is similar to part-of-speech tagging in that, given a set of tags for each word, the objective is to assign the appropriate tag to each word based on the context of the sentence. As in part-of-speech tagging, we use a trigram model [Weischedel et al, 1993; Church, 1988] to disambiguate

(a)



(b)

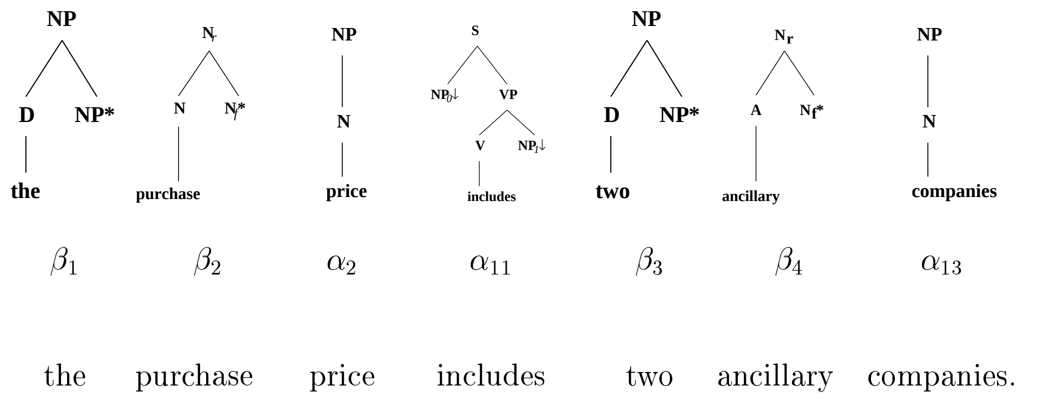


Fig. 2. (a) A selection of the supertags associated with each word of the sentence *the purchase price includes two ancillary companies* and (b) Most appropriate supertag sequence for the sentence.

supertags. The objective in a trigram model is to assign the most probable supertag sequence for a sentence given the approximation that the supertag for the current word is only influenced by (a) the lexical preference of the current word and (b) the contextual preference based on the supertags of the preceding two words.

The lexical and contextual preferences are estimated from a corpus of sentences where the words are tagged with the correct supertag. The estimates for unseen events are arrived at using a smoothing technique. We use Good-Turing discounting technique [Good, 1953] combined with Katz's back-off model [Katz, 1987] for smoothing. We use word features similar to the ones used in [Weischedel et al, 1993], such as capitalization, hyphenation and endings of words, for estimating the unknown word probability. Trained on 200,000 words of Wall Street Journal corpus with each word tagged with the appropriate supertag, the trigram supertagger associated correct supertags to 90% of the 20,000 word Wall Street Journal test corpus.

3.3 Lightweight Dependency Analyzer

Supertagging associates each word with a unique supertag. To establish the dependency links among the words of the sentence, we interpret the dependency requirements encoded in the supertags. Substitution nodes and foot nodes in supertags serve as slots that must be filled by the arguments of the anchor of the supertag. A substitution slot of a supertag is filled by the complements of the anchor while the foot node of a supertag is filled by a word that is being modified by the supertag. These argument slots have a polarity value reflecting their orientation with respect to the anchor of the supertag. Also associated with a supertag is a list of internal nodes (including the root node) that appear within the supertag. Using the structural information along with the argument requirements of a supertag, a sentence can be annotated with dependency links [Srinivas 1997].

An example illustrating the output of the LDA is shown in Table 1. The first column lists the word positions in the input; the second column lists the words, and the third column lists the names of the supertags assigned to each word by a supertagger. The slot requirement of each supertag is shown in column four and the dependency links among the words is shown in the fifth column. The $*$ and the . beside a number indicate the type of the dependency relation, $*$ for modifier relation and . for complement relation.

The supertag associated with each word provides the constituent structure information and the lightweight dependency analyzer provides dependencies between constituents. For the purpose of simplification, the constituent infor-

Position	Word	Supertag	Slot req.	Dependency links
0	The	β_1	+NP$*$	2$*$
1	purchase	β_2	+N$*$	2$*$
2	price	α_2	–	
3	includes	α_{11}	–NP. +NP.	2. 6.
4	two	β_3	+NP$*$	6$*$
5	ancillary	β_4	+N$*$	6$*$
6	companies	α_{13}	–	

Table 1
Output of the Lightweight Dependency Analyzer

mation is used to determine whether a supertag contains a clausal constituent and the dependency links are used to identify the span of the clause. Thus embedded clauses can easily be located and extracted, along with their arguments. Punctuation can be used to identify constituents such as appositives which can also be separated out.

4 Induction of Rules for Simplification

Our approach to automatically inducing rules from training data is described in this section. The training data is an aligned text corpus that links complex sentences to corresponding simplified sentences. This data is analyzed using LDA, and simplification rules are induced which are subsequently generalized using techniques similar to those used in Explanation Based Learning [Harmelen and Bundy 1988]. Figure 3 provides a schematic overview of the training and application procedures.

The training procedure for rule induction is detailed below, and illustrated with a running example.

(i) The training data consists of a set of input sentences (such as (6)) paired with a set of equivalent manually simplified sentences (such as (7)) corresponding to each of the input sentences.

Example 6: *Talwinder Singh, who masterminded the 1984 Kanishka crash, was killed in a fierce two-hour encounter.*

Example 7: *Talwinder Singh was killed in a fierce two-hour encounter. Talwinder Singh masterminded the 1984 Kanishka crash.*

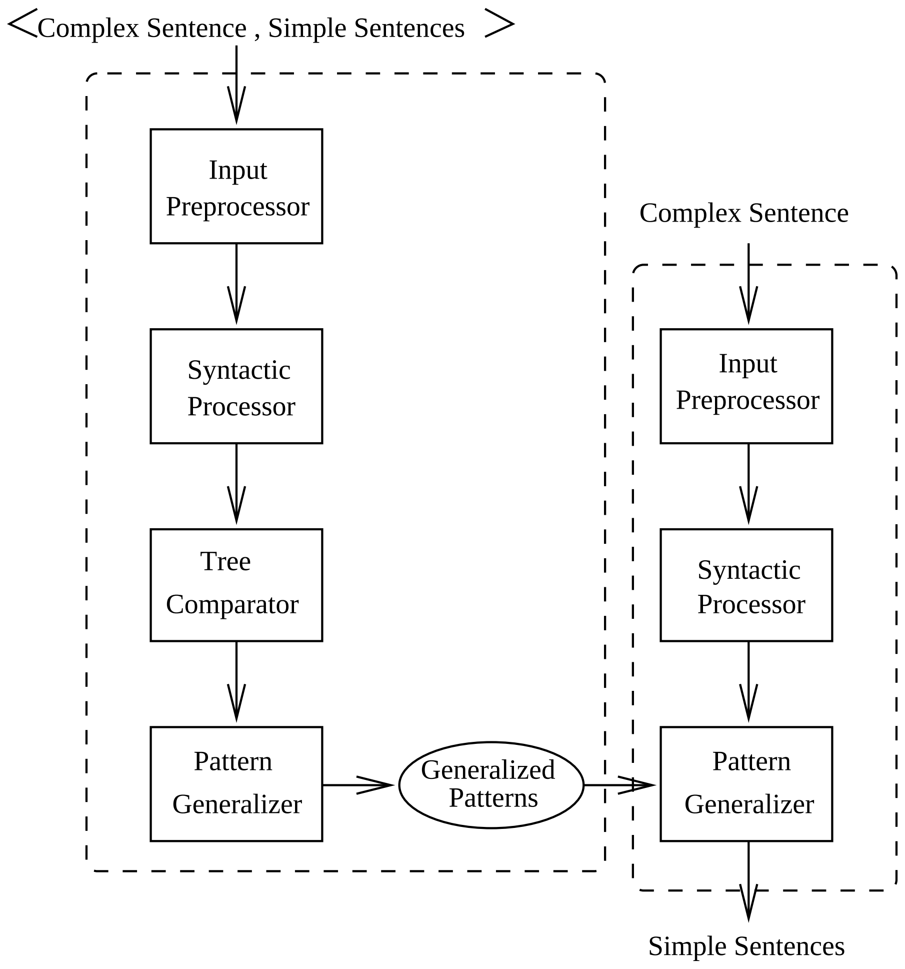


Fig. 3. Training and Application Procedures

(ii) The sentences in the training data are preprocessed to identify phrases that denote names of people, names of places or designations. These phrases are converted effectively to single lexical items.
(iii) Each training sentence S_i, along with its associated j (simplified) sentences S_{i1} to S_{ij}, is then processed using the Lightweight Dependency Analyzer (LDA).
(iv) The resulting dependency representations of S_i and S_{i1} through S_{ij} are 'chunked'. Chunking collapses certain substructures of the dependency representation (noun phrases and verb groups) and allows us to define the syntax of a sentence at a coarser granularity. Chunking also makes the phrasal structure explicit, while maintaining dependency information. Thus this approach has the benefit of both phrasal and dependency representations.

The chunked LDA representation for the example sentence and its simplified version is illustrated in Figure 4. The nodes of this representation consist of word groups which are linked by dependency information. Each node is also associated with a supertag, such as the Subject Relative Supertag (Rel β) and the Transitive Supertag (Trans α) in Figure 4.

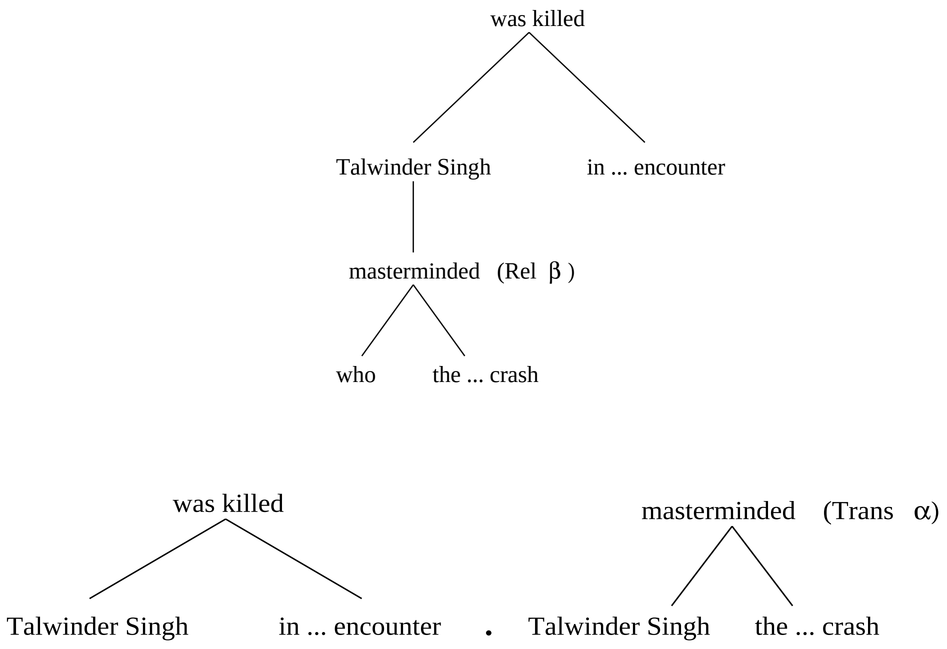


Fig. 4. Chunked LDA representation of a complex sentence and its simplified versions

(v) The chunked dependency representation of the complex sentence is compared with that of the simpler sentences using a tree-comparison algorithm. This algorithm computes the tree-to-trees transformations required to convert the sentence S_i to the sentences S_{i1} through S_{ij}. In the program, we represent the chunked dependency representation of the complex sentence (S_i), as well as the trees corresponding to the simpler sentences (S_{i1} through S_{ij}), as collections of parent-child tuples. Each tuple has two words or chunks along with their supertags. These tuple collections form the initial versions of a left hand side (LHS) and right hand side (RHS) of a transformation rule being created. In general, since the RHS will have more than one simple sentence, the tuples on the RHS are annotated to indicate the simple-sentence number.

Tree comparison works in the following fashion. First, the tuples that are common (modulo sentence number) between the LHS and the RHS are deleted. This corresponds to ignoring similar sub-trees in the tree-representation of the sentences.

We then generalize the tuples that are left, which determine the edges that will be changing during simplification. Words that occur on *both* sides of the rule are replaced by variable names such as \$1, \$2 etc, leading to lexical generalization. Words that remain on the LHS are replaced by variables but their supertag information is retained. However, punctu-

```
(a)

   (who/B_COMPs ,/B_sPU)
   (masterminded/B_N0nx0Vnx1 who/B_COMPs)
   (Talwinder_Singh/A_NXN masterminded/B_N0nx0Vnx1)
      ==>
   2.(masterminded/A_nx0Vnx1 Talwinder_Singh/A_NXN)
   2.(masterminded/A_nx0Vnx1 ./B_sPU)

(b)

   ($2/B_COMPs ,/B_sPU) ($3/B_N0nx0Vnx1 $2/B_COMPs)
   ($1/A_NXN $3/B_N0nx0Vnx1)
      ==>
   2.($3/A_nx0Vnx1 $1/A_NXN)
   2.($3/A_nx0Vnx1 ./B_sPU)
```

Fig. 5. Example of an induced rule (a) before generalization and (b) after generalization.

ation marks on either side are left intact. The attempt is to generalize the rule being induced, to make it applicable to a wider variety of sentences, but to retain its discriminating character. The strategy for generalization presented here is one of the many ways of generalizing a rule. However, other strategies may be more suitable for particular domains.

(vi) All input sentences S_i are processed using steps (ii) through (v), and duplicate rules removed. This results in a set of generalized simplification rules.

(vii) Each rule is indexed on its articulation points, and stored appropriately. The articulation point defines the link (or edge) to be cut for simplification. For example, this rule is indexed on the Subject Relative Supertag (Rel β).

An example rule induced by the program (given the sentences in examples 6 and 7 as input) is shown in Figure 5, before and after generalization. The tuples indicate parent–child relations. The terms on the LHS of the rule represent a conjunction of constraints which must be satisfied for the rule to fire. The generalized tags (`B_COMPs`, `A_NXN` etc.) are the appropriate supertags assigned to the words given the context of the sentences. The number `2.` refers to tuples in the second simple sentence.

Because of the use of supertags, the same rule will apply to all sentences which have relative clauses, regardless of the argument being relativized (subject/object/indirect object). Again, it is not important if the verb in the relative clause is *masterminded* or not; the rule will apply to any verb which is associated with the subject-relative transitive supertag. In fact, it will be true of any morphological variant of the verb; so verbs such as *masterminds, mastermind* etc. will also show the same behaviour in a similar context.

In our example, the changes indicated by the rule in Figure 5(b) together correspond to the three changes between the complex and the simple versions:

- The Subject Relative Supertag (Rel β or `B_N0nx0Vnx1`) changes to the Transitive Supertag (Trans α or `A_nx0Vnx1`).
- The head of the relative clause (represented by the parent of the Rel β node in the LDA representation) is copied in place of the relative pronoun. Note that reduced relative clauses will have empty relative pronouns.
- The Subject Relative Supertag (Rel β), and its dependents are separated out.

5 Application of Simplification Rules

In the rule application phase, every new sentence is preprocessed, analyzed using the LDA, and then chunked. Every node in the chunked LDA representation is a potential articulation point. The system retrieves all rules associated with the categories of these articulation points, and attempts to apply each of them. All rules that match the given structure are applied.

Since supertags localize all the dependencies of a word to one structure, the dependents of a word in the LDA representation appear as children of that word. The simplification rules that are induced operate on these localized representations, and have a local domain of influence. Therefore, these rules do not interact with each other with regard to their applicability. Also, the result of simplification is independent of the order of rule application.

When a sentence is matched against a specific rule, the tuples on the LHS are treated as a set of conjunctive terms which have to be satisfied for the rule to be applicable. This will involve the instantiation of the variables in the LHS. The corresponding variables on the RHS are bound using these values. The dependencies in the LHS are deleted and the ones on the RHS are added to the respective simplified sentences.

Consider the sentence shown in (8):

Example 8: *The creator of Air India, Mr. JRD Tata, believes that the airline, which celebrated its 60th anniversary today, could return to its old days of glory.*

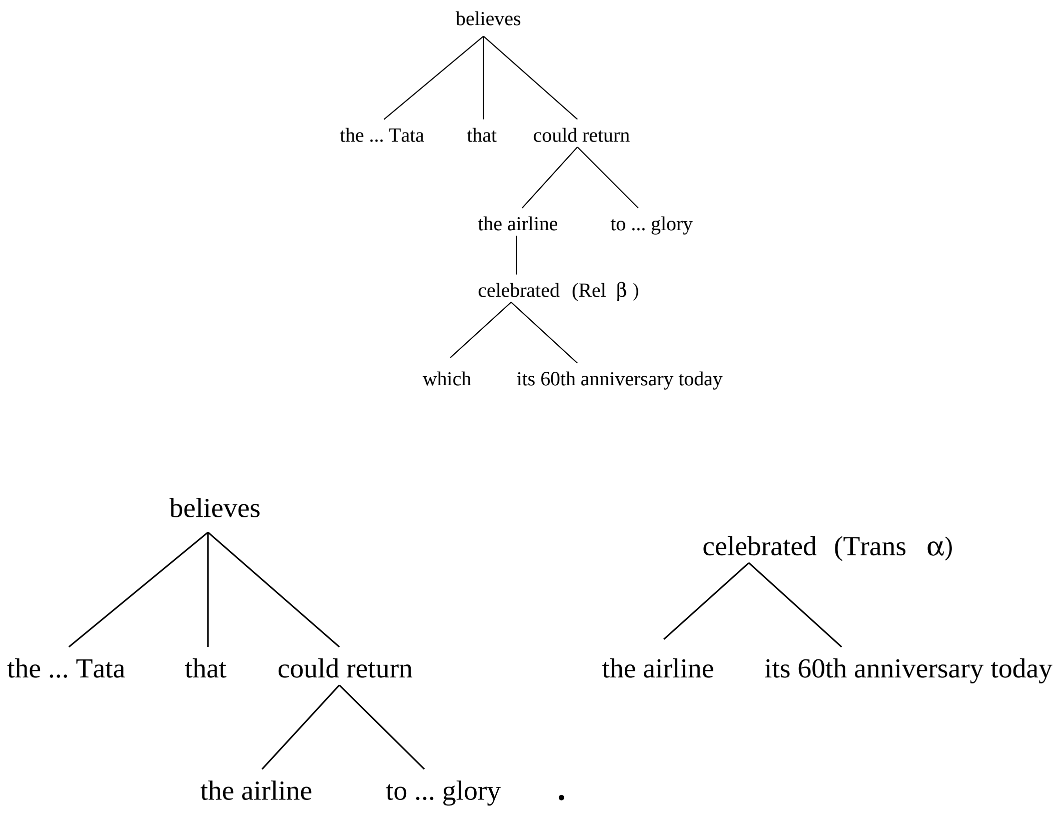


Fig. 6. Chunked LDA representation of a complex sentence and its simplified versions

Figure 6 shows the chunked LDA representations of the original text and the result of applying the rule induced in the training phase. *Note that while the structure at the sentence level is significantly different from the training example, there is a similarity in the sub-structure, and the rule is applicable on this component.*

The training data for this system was culled from a set of sixty-five stories from a leading Indian newspaper, published in English. A simplified version of these stories was manually created. For the present, we have concentrated on simplifying sentences with relative clauses. We are extending this to handle other syntactic phenomena.

6 Discussion

In this paper, we have presented a novel approach to induce rules for simplification of text using the representation provided by supertags, which combines phrasal and dependency information in a uniform manner.

As in many rule-based systems, hand-crafting rules is a time-consuming, tedious and error-prone process. An automated method of rule induction facilitates improved coverage of the system in terms of the phenomena handled, and the induction of rule sets for new domains with manageable effort. It provides us the opportunity to experiment with texts of different genres, and with a variety of preprocessing and post-processing software. In this work we have also integrated the transparency and interpretability afforded by rule-based representation with the robustness provided by the training process on (aligned) corpora. We believe that this is an important advance in simplification.

There are several problems of interest in the area of simplification. For example, the ordering of simplified sentences, the choice of referring or gap-filling expressions, and the maintenance of discourse coherence, all deserve attention. Another aspect that deserves attention is the evaluation of simplification. We believe that the performance of simplification can be best evaluated in the context of an application where simplification is used as a component. We are working on some aspects of these problems.

Acknowledgments

This work is partially supported by NSF grant NSF-STC SBR 8920230, ARPA grant N00014-94 and ARO grant DAAH04-94-G0426.

References

[Chandrasekar, 1994]
Chandrasekar R. *A Hybrid Approach to Machine Translation using Man Machine Communication*, PhD thesis, University of Bombay/Tata Institute of Fundamental Research, Bombay, September 1994.

[Chandrasekar et al, 1996]
Chandrasekar R, Doran C and Srinivas B. *Motivations and Methods for Text Simplification*, Poster paper. In *Proceedings of the* 16th *International Conference on Computational Linguistics (COLING'96)*, Copenhagen, Sweden, August 1996.

[Church, 1988]
Church, KW. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proc. 2nd Applied Natural Language Processing Conference, Austin, Texas, 1988,* pp. 136–143.

[Good, 1953]
Good, IJ. The population frequencies of species and the estimation of population parameters, *Biometrika* Vol. 40, Nos. 3 and 4, 1953.

[Harmelen and Bundy, 1988]
Frank van Harmelen and Allan Bundy Explanation-Based Generalization = Partial Evaluation, *Artificial Intelligence* Vol. 36, 1988.

[Hoard et al, 1992]
Hoard JE, Wojcik RH and Holzhauser KC. An automated grammar and style checker for writers of Simplified English, In PO Holt and N Williams (eds.), *Computers and Writing: State of the Art*, Kluwer, 1992.

[Joshi, 1985]
Joshi, AK. Tree Adjoining Grammars: How much context sensitivity is required to provide a reasonable structural description, In D Dowty, I Karttunen and A Zwicky (eds.), *Natural Language Parsing*, Cambridge University Press, Cambridge, UK, 1985.

[Joshi and Srinivas, 1994]
Joshi AK and Srinivas B. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing, In *Proceedings of the* 15th *International Conference on Computational Linguistics (COLING'94)*, Kyoto University, Japan, August 1994.

[Katz, 1987]
Katz S. Estimation of probabilities from sparse data for the language model component of a speech recognizer IEEE Transactions on Acoustics, speech and SignalProcessing, Vol. 35, No. 3, 1987.

[Kroch and Joshi, 1985]
Kroch AS and Joshi AK. The Linguistic Relevance of Tree Adjoining Grammars, Technical Report, MS-CIS-85-16, Department of Computer and Information Science, University of Pennsylvania, 1985.

[Schabes et al, 1988]
Schabes Y, Abeillé A and Joshi AK. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the* 12th *International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, August 1988.

[Srinivas et al, 1996]
Srinivas B, Doran C, Hockey BA and Joshi AK. An approach to Robust Partial Parsing and Evaluation Metrics, In *Proceedings of the Workshop on Robust Parsing at European Summer School in Logic, Language and Information*, Prague, August 1996.

[Srinivas, 1997]
Srinivas B. *Complexity of Lexical Descriptions: Relevance to Partial Parsing*, Doctoral Dissertation Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA, 1997.

[Weischedel et al, 1993]
Weischedel R, Schwartz R, Palmucci J, Meteer M and Ramshaw L. Comping with Ambiguity and Unknown words through Probabilistic Models, *Computational Linguistics*, Vol. 19, No. 2, pp. 359–382, June 1993.

[Wojcik et al, 1993]
Wojcik RH, Harrison P and Bremer J. Using bracketed parses to evaluate a grammar checking application. In *Proceedings of the* 31st *Annual Meeting of the Association for Computational Linguistics (ACL93)*, Ohio State University, Columbus, Ohio, 1993.