REVERSIBLE NLP BY DERIVING THE GRAMMARS FROM THE KNOWLEDGE BASE

David D. McDonald

Content Technologies, Inc.
14 Brantwood Road, Arlington, MA 02174
(617) 646-4124, MCDONALD@BRANDEIS.EDU

ABSTRACT

We present a new architecture for reversible NLP. Separate parsing and generation grammars are constructed from the underlying application's semantic model and knowledge base. By having two grammars we are free to use process-specific representations and control techniques, thereby permitting highly efficient processing. The single semantic source ensures the parsimony of development and matched competence that make reversible NLP attractive.

INTRODUCTION

Most natural language processing systems are initially built in a single direction only; most are parsers (understanding systems), a few are generators. These systems are often then embedded in full, bi-directional interfaces, whereupon a new, almost non-technical kind of problem arises if differences in the two uni-directional subsystems are not controlled. The full system may not understand the same wording or syntactic constructions that it can generate; or generation and parsing development teams may both have to work on extensions and modifications to their grammars, with the likely result that still further differences will be introduced.

These practical problems bolster an intuition that many have that knowledge of parsing and generation is the same knowledge in a person's mind, or at least that the two faculties draw on a single representation of their language even if it is engaged in different ways. This has led to the goal of reversible NLP systems. The common approach has been to take the computational artifact constructed by one of the single-direction projects, typically its grammar, and to adapt it for use in the other direction.

At ISI, for example, their massive systemic grammar for generation, NIGEL, (Mann & Matthiessen 1985) has since been adapted for use as a parser (Casper 1989). With the conceptual basis of the transformation in place, the development of further extensions and modifications is done on the generation grammar, and then that grammar is re-transformed to yield the new parsing grammar.

The other well-known approach to reversible NLP is of course to use the very same computational artifact in both processing directions. Thus far this artifact has invariably been a grammar, typically some kind of specification of the text-stream -- logical form relation that can be used as a transducer or can supply the data for it.

Parsers and generators draw on their grammars as their predominant knowledge source. The grammar thus becomes a bottleneck for the processing if it is not designed with efficiency of processing in mind. When virtually the same computational representation of the grammar is used in both processes and it is given an active role, e.g. when the grammar is couched in a unification formalism, this bottleneck can be substantial since the "common denominator" processing architecture that must be employed in order for the grammar to be literally usable by both processes will be markedly less efficient than architectures that work from single-direction representations of the grammar.

By their nature as information processing systems, language understanding and generation are quite different kinds of processes. Understanding proceeds from texts to intentions. The "known" is the wording of the text and its intonation. From these, the understanding process constructs and deduces the propositional content conveyed by the text and the probable intentions of the speaker in producing it. Its primary effort is to scan the words of the text in sequence, during which the form of the text gradually unfolds. This requirement to scan forces the adoption of algorithms based on the management of multiple hypotheses and predictions that feed a representation that must be expanded dynamically. Major problems are caused by

ambiguity and under-specification (i.e. the audience typically receives more information from situationally motivated inferences than is conveyed by the actual text).

In generation, information flows in the opposite direction from understanding. Generation proceeds from content to form, from intentions and perspectives to linearly arrayed words and syntactic markers. A generator's "known" is its awareness of its intentions, its plans, and the text it has already produced. Coupled with a model of the audience, the situation, and the discourse, this provides the basis for making choices among the alternative wordings and constructions that the language provides---the principal activity in generation. Most generation systems do produce texts sequentially from left to right---just like an understanding system would scan it; but they do this only after having made decisions about the content and form of the text as a whole. Ambiguity in a generator's knowledge is not possible (indeed one of its problems is to notice that it has inadvertently introduced an ambiguity into the text). And rather than under-specification, a generator's problem is to choose from its over-supply of information what to include and what to omit so as to adequately signal its intended inferences to the audience.

Our concern with efficiency---optimizing the two processes to fit their differing information processing characteristics---has led us to approach reversible NLP by a compilation-style route where the grammar that the processes use is not one artifact but two, each with its own representation that is deliberately tailored to the process that uses it. Like the system at ISI, our reversible knowledge source is grounded in the generation process and then projected, via a compiler, to create the representation used by the parser. The difference is that while ISI projected the grammar that the generator used, i.e. the set of system networks that is the model of the linguistic resources provided by the language and their dependencies, our system is a projection from the underlying application's conceptual model.

In generation one starts with a set of objects representing individuals, relations, propositions, etc. that have been selected from the application program as its representation of the information it wants to communicate. Accordingly, the kind of knowledge that a generator must draw on most frequently is what are the options for realizing those objects linguistically. In order to make this look-up efficient, one is naturally led to an architecture where this knowledge is stored directly with the definitions of the objects or their classes, in effect distributing a highly lexicalized grammar over the knowledge base.

A SIMPLE EXAMPLE

To be concrete, consider the example in Figure One below, a simple definition of the object class (category) for generic months. This expression says that a month is a kind of time stuff that can be viewed either as a point or an interval; that it has a specific number of days, a position within the year, and, especially, that it has a name and abbreviations---the primary options for realizing references to the month in natural language.

```
(def-category  month
  :specializes
     time/interval-or-point
  :slots
     ((name   (word proper-name
               :may-be-abbreviated))
      (number-of-days  number)
      (position-in-the-year
                       number)))
```

Figure One

In our system, CTI-1, the evaluation of this expression causes a number of different things to be constructed: the object representing the category, indexing and printing functions for objects with that category (instances of the class), and a defining form. One then uses the form to create objects for the twelve months, as shown in Figure Two for December.

```
(define-month
  :name "December"
  :abbreviation "Dec"
  :number-of-days 31
  :position-in-the-year 12)


#<month December
   :name  #<word "December">
   :abbreviation     #<word "Dec">
   :number-of-days   #<number 31>
   :position-in-the-year
                     #<number 12>>
```

Figure Two

As a result of evaluating this form, we get the object for December (Figure Two). When referring to this object in generation we will look it its name field and use the word object there, or perhaps the abbreviated word.

When parsing, we will see an instance of the word "December" or the phrase "Dec." and want to know what object it the application program's model it refers to. In CTI-1 this is done by the phrase structure rules in Figure Three. These rules were written automatically (compiled) as one of the side-effects of defining the object for December; the code for constructing the rules was incorporated into the

Define-month form by following the annotation in the expression that defined the category month, the same annotation that controls where the the generator looks when it wants to realize a month object.

```
#<context-free-rule
:print-form |month -> "December"|
:lefthand-side  #<category month>
:righthand-side (#<word "December">)
:syntactic-form
      #<category proper-noun>
:referent  #<month December>>

#<context-free-rule
 :print-form |month -> "Dec."|
 :lefthand-side   #<category month>
 :righthand-side  ( #<word "Dec">
                    #<word "."> )
 :syntactic-form
      #<category proper noun>
 :referent  #<month December>>
```

Figure Three

These parsing rules are part of CTI-1's semantic grammar. They are rewrite rules. When the word "December" or the two word phrase "Dec" "." is scanned, the text segment is spanned with an edge of the chart (a parse node), and the edge receives a three part label: (1) the category "month", which participates in the semantic grammar, (2) the "form" category "proper-noun", which is available to the syntactic grammar, and (3) the referent the edge picks out in the application model, i.e. the very object #<month December> that was defined by the form in Figure Two.

SUMMARY OF THE APPROACH

Before going into a more elaborate example we can briefly summarize the reversible NLP architecture we have adopted. The grammar is developed on the generation side by the linguist/semantic modeler as part of defining the classes and individuals that comprise the application's domain model. They include with the definitions annotations about how such objects can be realized in natural language.

A side-effect of definition is the automatic inversion of the generation rules specified by the annotation to construct the equivalent set of parsing rules. Parsimony and uniformity of coverage, the practical goals of reversible systems, are achieved by having the parsing grammar constructed automatically from the original forms that the linguist enters rather than having them redundantly entered by hand.

Note that what we are projecting from as we invert "the generator's rules" is the generator's representation of the form-meaning relationship---its rules for mapping from specific objects in the underlying application's domain model to their (set of) surface linguistic forms by warrant of how the model has characterized them semantically. This is not the same as a representation of the principles that constrain the valid compositional forms of the language: the constraints on how individual lexical items and syntactic constructions can be combined, what elements are required if others are present, a formal vocabulary of linguistic categories, and so on. That representation provides the framework in which the form-meaning relationship is couched, and it is developed by hand. For CTI-1 the design choices as to its categories and relations are taken from the theory of Tree Adjoining Grammar (Joshi 1985).

The simplicity and immediacy of the automatic inversion is possible because in our approach the task of parsing (determining a text's form) has been integrated with the task of understanding/semantic interpretation (determining the denotations of the text and its elements in some model). This integration is brought about by using a semantic grammar. A semantic grammar brings the categories of analysis used by the parser into the same realm as those used by the generator, namely the categories of the application domain (in the present case personnel changes), for example people, companies, dates, ages, job titles, relations such as former, new, or has-title, and event types such as appoint, succeed, retire, etc.

If the parser had been intended only to produce syntactic structural descriptions of the text, then projecting its rules from the generator would have been either impossible or trivial. An application supports a potentially vast number of categories; the syntactic categories of natural languages are fixed and relatively small. Collapsing the different kinds of things that can be realized as noun phrases down to that single category would lose the epistemological structure of the application's model and provide only minimal information to constrain or define the grammar.

TREES FOR GENERATION, BINARY RULES FOR PARSING

Consider the definition of the event type "appoint-to-position, shown in Figure Four. It's linguistic annotation amounts to the specification of a tree family in a TAG. The features given in the annotation are consulted to establish what trees the family should contain, building on the basic subcategorization frame of a verb that takes a subject and two NP complements, e.g. that it includes a passivized tree, one in participial form without its subject, and so on.

```
(def-category  appoint-to-position
 :slots ((person  person)
         (company  company)
         (position  title))
 :tree-family
  ((personnel-change
    company!___!person!title
    :verb "appoint"
    (subject -> company)
    (object1 -> person)
    (object2 -> position)
    :optional
      ((by-company -> company)
       (by-person -> new-person))
    :passivizes
    :forms-participles )))
```

Figure Four

The annotation is equivalent to binding a specific lexeme to the verb position in the trees of the family (this is a lexicalized TAG), as well as restrictions on the denotations of the phrases that will be substituted for the other constituents of the clause, e.g. that the subject picks out a company, the first object a person, etc.

A tree family plus its bindings is how this annotation looks from the generator's perspective. For the parser, this same information is represented quite differently, i.e. as a set of binary phrase structure rules. Such rules are the more appropriate representation for parsing (given the algorithm in CTI-1) since parsing is a process of serial scanning rather than the top-down refinement done in generation. During the scan, constituents will emerge successively bottom up, and the parser's most frequent operation and reason for consulting the grammar will be to judge whether two adjacent constituents can compose to form a phrase. (The rules are binary for efficiency concerns: CTI-1 does the multiplication operation for determining whether two adjacent constituents form a phrase in constant time regardless of the size of the grammar.)

The tree family defines a set of rules that are applicable to any verb and semantic bindings that share the same subcategorization frame, such as "name" or "elect". In projecting the annotation on the definition of appoint-to-position into parsing rules, the compilation process will create the rules of the family if it does not already exist, and also create a set of unary rules for the immediate non-terminals of the verb, one for each different morphological variant. One of these rules is shown in Figure Five, along with the general rule for the object-promotion aspect of passivization.

```
#<context-free-rule
 :lefthand-side
   #<category
      pc/company!___!person!title>
 :righthand-side
      ( #<word "appointed"> )
 :form   #<category main-verb/-ed>
 :referent
    #<category personnel-
      change/appoint-to-position>>

#<context-free-rule/form
 :righthand-side
      ( #<category "be">
        #<category main-verb/-ed> )
 :head  :second-constituent
 :revised-mapping
      ((object -> subject)))
```

Figure Five

The first phrase structure rule, part of the semantic grammar, ties the past participial form of the verb into the family of rules. The long category name is a convenient mnemonic for the family of rules, since it shows by its spelling what semantic categories of constituents are expected as sibling constituents in the clause as a whole.

The object promotion rule is a syntactic ("form") rule that makes reference to the form label on an edge rather than their semantic label. The rule for "appointed" has a form label showing that it is a main verb in past participle form, which is what the syntactic rule is looking for. When a segment like "was appointed" is scanned, the label "be" on the edge spanning "was" will be checked against the label "main-verb/-ed" on the edge over "appointed" and the resulting edge will carry the semantic label and referent of the phrase's head, i.e. the main verb.

Figure Six shows some of the other rules in the family so that one can get an idea about how the parsing of the whole clause will be done. The rules are given just by their print forms.

```
1.  pc/company!___!person!title
      ->   appointed

2.  pc/company!___!title
      ->  pc/company!___!person!title
          person

3.  pc/company!___
      -> pc/company!___!title
         title

4.  personnel-change
      ->  company
          pc/company!___
```

Figure Six

STATE OF DEVELOPMENT

The parsing side of this architecture for reversible NLP is implemented and running in an operational system, CTI-1. It has a mature domain model for personnel changes, and has been running the parsing grammar that is projected from that model on hundreds of articles from the Wall Street Journal ("Who's News").

The generation side of the architecture is in its infancy, waiting on a suitable domain and task where the reasons for speaking and the situation models are rich enough to motivate subtle nuances in phrasing. By the same token, my prior experience with generation leads me to believe that the design of the linguistic annotations is well-founded for generation, and that this side of the reversal will fall out once the opportunity for implementation arises. When this happens the "raw material" that the mappings discussed here will supply will be fed to a text planner like Meteer's RAVEL orchestrator in her SPOKESMAN system, and then drive a TAG realization component along the lines of Mumble-86.

REFERENCES

Joshi, A.K. (1985) How much context-sensitivity is required to provide reasonable structural descriptions: tree adjoining grammars. in Dowty et al. (eds) Natural Language Processing, Cambridge University Press.

Mann, W.C. & C. Matthiessen (1985) *A demonstration of the Nigel text generation computer program*, in Benson & Greaves (eds) **Systemic Perspectives in Discourse**, Benjamins, Amsterdam.