The Interplay of Syntactic and Semantic Node Labels in Partial Parsing

David D. McDonald

Dept. of Computer Science, Brandeis University
14 Brantwood Road, Arlington MA 02174, USA
email: mcdonald@chaos.cs.brandeis.edu

Abstract

Our natural language comprehension system, "Sparser", uses a semantic grammar in conjunction with a domain model that defines the categories and already-known individuals that can be expected in the sublanguages we are studying, the most significant of which to date has been articles from the Wall Street Journal's "Who's News" column. In this paper we describe the systematic use of default syntactic rules in this grammar: an alternative set of labels on consitutents that are used to capture generalities in the semantic interpretation of constructions like the verbal auxiliaries or many adverbials. Syntactic rules form the basis of a set of schemas in a Tree Adjoining Grammar that are used as templates from which to create the primary, semantically labeled rules of the grammar as part of defining the categories in the domain models. This design permits the semantic grammar to be developed on a linguistically principled basis since all the rules must conform to syntactically sound patterns.

1 Partial parsing and semantic grammars

The rate-limiting step in advancing how much machines can understand from a natural language text is the size and thoroughness of the semantic models used to make sense of the information the texts contain. At the same time, the state of the art in parsing and information extraction mandates that the texts being analyzed can no longer be artificial — made up by linguists to test their grammatical theories — but should be actual texts written by people for other people; indeed, the right kind of test corpus today will be downloaded from an online service on the day of the test.

To accomodate this combination of unrestricted sources and limited semantic competence, one focuses on sublanguage analysis (in the sense of Kittredge & Lehrberger 1982). This may be simply a restriction in the source of one's texts to just one topic and register, such as in the present instance single paragraph articles taken from the "Who's News" column of the Wall Street Journal, which contain almost nothing beyond announcements of the promotions or retirements of executives in commercial businesses. It may more broadly be a concentration on a given kind of information and its standard sublanguage but with no constraint on the sources in which it appears, e.g. extracting executive job change information in articles of any length appearing anywhere in the Journal or other business news sources available online, while at the same time functioning robustly dispite the unknown words and unparseable constructions in other portions of the texts.

This then is the sence in which the "partial parsing" of this paper's title should be understood: the parses the system makes, and its comprehension of the parsed segments, should be both complete and thorough, but those segments will typically constitute only a portion of the whole text being analyzed. Since the intrusion of off- topic segments can be at any granularity — apposatives or adjuncts within clauses, modifiers within NPs, etc., a partial parser in this sense will incorporate some heuristic techniques for compensating for such gaps in its analysis, but for the most part its parsing machinery can be conventional and its only substantial differences from a parser intended to always parse every part of its text will lie in its grammar.

In order to constrain the grammar to only those topics and sublanguages within the corpus that the system's semantic models will understand, one is naturally drawn to the use of semantic grammars. Here the grammar will only provide an analysis if the model will comprehend it — no parse node will be allowed to be formed if it does not also have a semantic interpretation. No effort is wasted in trying to make sense of portions of the text not on the target topic since the grammar will leave them unanalyzed and no interpretations will be initiated.

Introduced originally by Dick Burton as the basis of the natural language interface to the Sophie ICAI system (Burton 1976; Burton & Brown 1975) and later used for the interfaces in Ladder (Hendrix et al. 1978), Planes (Waltz et al. 1976), and many other systems, semantic grammars have the advantage of high specificity, easy, almost common-sense definition, and extremely low run-time ambiguity. At the same time, they also suffer from brittleness, minimal carryover and significant effort in extending them to other domains, and a striking lack of ability to appreciate linguistic generalizations.

In this paper we will describe a version of semantic grammar that addresses these problems while retaining the formalism's strengths. We provide a systematic basis for relating the semantic categories used by the parser to the categories and representational structures used in the semantic model that provides the text's interpretation. We also provide the parser with a parallel set of standard syntactic categories shadowing the semantic categories and providing the basis of domain-independent default rules with a common semantics. This is all brought together in a set of general schemas, derived from a Tree Adjoining Grammar, that allow the definition of the grammar to be done as part of the definition of the semantic model, ensuring consistency and reducing much of the labor of writing semantic grammars in the process.

We will begin by motivating the way in which we link parsing rules to their interpretations, and with that introduce the notion of schemas that are used to define the two simultaneously. We will then describe the 'form' rules that provide the syntactic defaults, and present the algorithm for checking for possible constituent combinations in some detail since it is the means by which the two kinds of rules are interleaved.

2 Parsing to objects

What is the goal of a parser? The customary answer is that it is to determine the form of a text — its decomposition into constituents and the grammatical relations among them, which is to then serve as the basis of its semantic interpretation — the recovery and representation of its meaning. The line between form and interpretation begins to blur, however, once one adopts a semantic grammar as the basis of the analysis of text form. A constituent in a semantic grammar has a label that reflects its category when taken as an thing or event-type in the world: 'person' rather than NP, 'elect' or 'get-position' rather than verb. Given a multi-level analysis by the parser, with transformations or their equivalent to render a surface-level representation into a canonical form, such a choice of categorizations will yield a representation not too disimilar from some conceptions of logical form, with the constituent labels providing the predicate names applied to variables introduced by the transformations.

The present work, however, employs a denotational rather than descriptive notion of semantic interpretation. Here the process of parsing a text incrementally accesses or contructs a set of unique semantic entities — individuals in a model of the world and the ongoing discourse as represented in the mind (code) of the language user. The interpretation function is applied rule by rule as the parse proceeds, rather than having to wait for it to end. Many of the individuals and all of the base types of this model will exist before the text is processed, and the task of the language understanding system is taken to be to recover from its analysis of the text the objects representing these individuals, and to add to the model any newly established individuals and any new relations or properties over them and pre-existing individuals.

As a parser then, the Sparser system does recover a phrase structure-based analysis of a text's form. But this form is only a means to an end, namely to project from the words of the text, as channeled by the constituent structure, directly to the objects in the world/discourse model without any interveening intermediate level of representation of the meaning as another expression such as a logical form. Since this projection happens incrementally as the parse proceeds from beginning to end in one pass through the text, the parser may as well be said to be parsing to objects (in something of the manner of Reisbeck and Martin 1985), since it will not complete a constituent without also recovering that constituent's counterpart in the model.

This being the case, and given the goal of a partial parser to tightly coordinate what the parser's grammar is able to analyze with what its semantic model is able to understand, a natural way to proceed is to have the rules of the grammar written as a side-effect of defining the types of the model. What this means is that when one defines, say, a category of event like 'confirm-in-position', that definition should include the definition of the rules needed to parse texts about that event-type. (Confirm-in-position is intended here as the denotation of the verb "confirm" as used in a sentence like "*J. Gordon Strasser, acting president and chief executive officer of this gold mining company, was confirmed in the posts* ...", which is typical of the "Who's News" texts we have worked with.) Concombinantly, the parsing rules created by this definition should spell out not only how the constituents of the text compose, but also how those compositions should be projected to the discourse model to incrementally construct an individual to represent the confirmation event rule by rule as the parse progresses.

Let us consider what has to go into these definitions so that establishing 'confirm-in-position' as an event-type in the model will at the same time provide the rules that the grammar needs to handle a sentence like the example. For its own purposes, the semantic definition of an event or other relations will specify a set of necessary and optional participants, including value restrictions constraining what categories of individuals those participants can be. For 'confirm-in-position', as evidenced by news articles in the Wall Street Journal, the agent that causes a confirmation can be either a company or its board of directors; the participant who gets confirmed is a person; and that person is confirmed to some position. We can use these categories directly to provide the labels for the corresponding constituents in the semantic grammar, where they take on the force of selectional restrictions.

Beyond the selectional restrictions on the participants, the definitions must supply the words that have the intended meaning as one of their senses (i.e. the semantic object under definition) as well as the syntactic subcategorization frames that the words take. Given the choices of lexical realization, syntactic information is then needed to spell out the phrasal patterns corresponding to the subcategorizations and how the participants project onto them.

With few exceptions, the subcategorization frames are shared by a great many words, and consequently can be schematized and shared among the definitions (e.g. transitive verb, intransitive, noun taking quantification of number, noun holding a position in a sequence, etc.). The patterns of verbs especially are most naturally given in terms of syntactic phrasal categories, and can thus also provide the default syntactic categories that are provided with the rules of the semantic grammar, as we shall see.

We fill the requirement to fit all of the participants of a relation to a single sharable syntactic schema by adopting Lexicalized Tree Adjoining Grammars as the linguistic formalism (Joshi 1985; Shabes, Abeille & Joshi 1988). TAG's extended domain of locality allows the configurational and grammatical relations of all of a lexical head's arguments to be given at once, which is important here since the semantic relationship of each argument to its matrix relation must be specified in the schema, and this is easier to do if all of the participants of a domain relation can be paired with their syntactic counterparts in one statement. The schemas we adopt are a notational variant on the TAG Tree Families of Abeille 1988, which give all of the transformational variations of a subcategorization frame in one structure. The particulars are described in section five.

To summarize the framework so far, we are employing in the Sparser system a grammar of rules of syntactic form which employ semantic categories as their primary labels, with standard syntactic labels provided with them as defaults. Each rule has an accompanying semantic inter-

```
(1)   job-event  ->  board-of-directors  job-event\agent
(2)   job-event  ->  company  job-event\agent
(3)   job-event  ->  job-event  in-position
(4)   job-event  ->  job-event  as-position
(5)   job-event  ->  person  confirm-in-position+passive
(6)    job-event  ->  person+pos  confirm-in-position+nominalization
(7)   job-event  ->  confirm-in-position+nominalization  of-person
(8)   job-event\agent  ->  confirm-in-position  person
(9)  confirm-in-position  ->  "confirm"
(10) confirm-in-position  ->  "confirms"
(11) confirm-in-position  ->  "confirmed"
(12) confirm-in-position  ->  "confirming"
(13) confirm-in-position+nominalization  ->  "confirmation"
```

Figure 1: Semantically labeled rules

pretation function that will find or construct the denotation of the constituent it creates. The grammar rules and the set of model-level object types are created at the same time by definitions that recruit the value restrictions on the participants in a relation to supply the semantic labels on the grammar's rules, where they implicitly serve the role of selectional restrictions ensuring that the rules cannot complete unless their constituents will make sense to the world model. The set of rules created by a type definition is determined by instantiating the TAG tree family(s) corresponding to the subcategorization frames of the word or words that can realize the relation-type in a text.

The effect on the final grammar is as though we had taken a conventional syntactic grammar that used syntactic preterminal categories and performed two multiplications: the first takes its preterminal categories for content words and multiplies them across all of the lexical choices specified by the definitions of the model's domain-specific types. The second takes the rules composing normal syntactic nonterminals and multiplies them across all of the sets of selectional restrictions dictated by the semantic categories of the individuals that can participate in the domain relations. The resulting grammar is dramatically larger than the original, but it will show a equally dramatic increase in runtime efficiency because of its minimal ambiguity and its immediate and incremental connection to the semantic interpretation, which, after all, is the ultimate purpose of the grammar in the first place.

3 Integrating rewrite rules and denotations

Before introducing the notation and mechanisms of the model-level definitions and the subcategorization schemas, we should look at some examples of the grammar rules — productions — that Sparser uses. We will start with some semantically labeled rules based on a type definition — the output of a schema, and then look at the syntactically labeled rules that complement it. We start with the set of rules produced for 'confirm-in-position', all of which are listed just in terms of their semantic labels in figure 1 We will then look at the details of what is represented in one of these rules and show how the syntactic and denotational information is represented and put to use. We will end the section by sketching how the rules are used in a parse of a text that is completely within the job-change sublanguage so that one can get a sense of the interplay between form and denotation in the Sparser system.

```
#<phrase-structure-rule psr11
  :lefthand-side  #<category confirm-in-position>
  :righthand-side  ( #<word ''confirmed''> )
  :syntactic-form  #<category verb+ed>
  :referent  #<category confirm-in-position> >
```

Figure 2: Rewrite rule 11

```
#<edge   e22 ;; a designator to use as the edge's name
  :starts-at #<position 18>
  :ends-at #<position 19>
  :category #<category confirm-in-position>
  :left-daughter #<word ''confirmed''>
  :right-daughter :unary-rule   ;; i.e. only the 'left' daughter matters
  :used-in nil  ;; later points to the edge this is a daughter of
  :form #<category verb+ed>
  :referent  #<category confirm-in-position> >
```

Figure 3: An edge in the chart

This rule set (Figure 1) consists of five unary lexical rules (9 - 13), and eight binary rules over non-terminals corresponding to the conventional rules: S -> NP VP, VP -> VG NP, S -> NP VG, S -> S PP, and two NP rules for nominalizations. These semantically labeled rules are complemented by a set of generically applicable syntactically labeled 'form rules', e.g. for the "be + ed" of the passive, other auxiliaries and standard adverbs, and for the definite or indefinite articles that can occur with the nominalizations. We take these up in a moment.[1]

In parsing our example, "*J. Gordon Strasser, acting president and chief executive officer of this gold mining company, was confirmed in the posts*", the instantiation of a model-level 'confirm-in-position' event will start when the word "*confirmed*" is scanned, firing rewrite rule 11 (figure 2). That rule has the internal representation shown below. The '#<>' notation indicates pointers to objects, with the first term in the brackets indicating the object's type and the rest a summarizing print form, typically the symbol or string used to name it.

Note that the 'referent' given with the rule is a pointer directly into the system's semantic model. It is available as part of the statement of the rule because the rule is created as a side-effect of defining that event-type. Notice also that the rule also indicates the default syntactic label (category) that is to be given to the constituents it forms, i.e. verb+ed, complementing the primary semantic label given as the rule's lefthand side.

When completed, this rule creates a nonterminal node — an edge in Sparser's chart — as shown in figure 3. For purposes of illustration we have the example text start at chart position zero, which puts the word "*confirmed*" between positions 18 and 19. (In the actual text, this was the first clause of the first sentence of a one paragraph article, with an extensive set of headers and with the name and location of the company where the change was going on announced in a dateline just

[1] There are not as many phrasal rules in this set as one might expect for a transitive verb given the notion of a transformation family. This is because Sparser handles the relative pronouns of relative clauses by rewriting them with the semantic label of their NP head, making them equivalent here to rules 2 and 5, and it treats other WH constructions using 'form rules' as described below. Other predictable constructions such as it-clefts or topicalization have not yet occured anywhere in the studied portions of the Who's News corpus and have been omitted just because we have not had the examples to study for their semantic analysis.

before the sentence started — all of which was also entered into the chart and processed.)

Note that the edge essentially just carries over the information given in the rule. The category given as the 'lefthand-side' of the rule becomes the edge's 'category' — it is this semantic label which will be the primary label used the course of parsing. The category given as the 'syntactic-form' of the rule becomes the edge's 'form' — its syntactic label, which will be used only if the semantic label cannot participate in a composition with any of the neighbor edges, i.e. it is the default.

The semantic analysis at work here holds that content words (nouns, verbs, adjectives) name categories of individuals in the semantic model. The denotation of "*confirmed*" is, as shown, the category ascribed to events of that type (which we also use as the semantic label on its edge, a common but not universal practice in this grammar since particularities of subcategorization and applicable substitution contexts can override it and call for more specific labelings). The denotation of the word "*company*" in the example is the category used for all individuals representing companies and similar enterprises; "*posts*" denotes the category for positions, specialized in this case to note that more than one title and/or company is involved; and so on.

The composition of modifiers onto a head constituent may specialize the category of the head to one of its subcategories (as done implicitly with the plurality of "*posts*"), or may predicate attributes to an individual. The composition of some discourse anchor — tense for verbs or determiners for nouns — will change the denotation to now pick out some individual fitting the category of the phrase accumulated on the headline to that point. This happens when "*was*" and "*confirm*" are composed, where we will establish an individual of domain-type (category) 'confirm-in-position'. This involves a 'rule of form' which we take up in the next section.

In the interests of space we will jump ahead to the point when the major constituents of the example have been formed so that we can concentrate on how the rules for confirm-in-position are applied and how they take the new individual in the model — an instance of this event-type — and fill its participant roles with the individuals denoted by those constituents.

Below (figure 4) is a vertical presentation of that chart state, followed by a list of the edges with their semantic and syntactic labels and their denotations. A compact notation is used here for illustration purposes. The edges names, e.g., 'e23', reflect the order in which the edges were formed. The numbers around the excerpted texts are the chart positions.

At this point SPARSER will apply rule 5 from the set above to combine edges e20 and e23 to form a clause ('S') labeled 'job-event' from position 1 to position 20, edge e30. When that edge is formed, the denotations of its daughter edges are combined following the interpretation function given with the rule (see section 5). The result, the denotation of e30, will be the same individual denoted by e23, but now that individual is augmented by having its 'agent' role bound to the person pointed to by the subject edge e20.

```
  e20     1 J. Gordon Strasser I of this gold mining company 18
  e23     18 was confirmed 20
  e29     20 in the posts 23

[e20: person, np, #<individual:person Strasser> ]

[e23: confirm-in-position+passive, vg, #<individual:confirm-in-position> ]

[e29: in-position, pp, #<individual:position:collective President I> ]
```

Figure 4: Chart state and list of edges

```
#<edge  e31
  :starts-at #<position 1>
  :ends-at #<position 23>
  :category #<category job-event
  :left-daughter #<edge 30>
  :right-daughter #<edge 29>
  :used-in nil
  :form #<category S>
  :referent
    #<individual
      :type ( confirm-in-position past get-position job-event
              transition-event )
      :bindings ( #<binding person = #<individual:person Strasser>>
                  #<binding position =
                        #<individual:position:collective President I>> )>>
```

Figure 5: Edge e31

This new edge, e30, will then be combined with the pp-adjunct edge e29 using rule 3 with a similar result. Another edge, e31, is formed as shown in figure 5. Its referent is again the same individual that was instantiated when the verb group edge was formed (i.e. the individual representing this instance of confirm-in-position). Now it gets its 'position' role bound to the individual of type position denoted by e29.

The parse is now finished syntactically since we have recovered a grammatically complete unit, a sentence. Semantically it is still incomplete, since the text did not explicitly give the agent that did the confirming and so the event-type is not yet saturated. In this genre it would be safe to infer that the agent is the company where the person now holds the position.

4 Form rules

The preponderance of the syntactic (surface form) rewrite rules in Sparser's grammar will have words or semantic categories on their righthand sides resulting from the application of some TAG schema during the definition of a domain-specific type in the world model; when the distinction must be made we will refer to them as 'semantically characterized rules'. There are also rewrite rules whose categories are taken from the conventional syntactic vocabulary and consequently are shared across all of the semantic domains; we call these 'form rules' because they make reference to general aspects of surface form rather ideosyncratic aspects that apply to only certain domains of discourse and their sublanguages. Form rules are the subject of this section. We will look at an example of a form rule and the rationale behind its use, and then go through the 'Check' algorithm that coordinates the selection of semantically characterized or form rules.

A form rule combines a word or occasionally a semantically labeled edge — call it the 'literal' — with the form category of another edge — call that edge the 'head'. The head edge will always lie on the headline of the final tree; the literal will always be a specifier, an auxiliary, or a modifying adjunct or adverb. The purpose of a form rule is to capture a general fact about phrase formation in the language, one whose implications for the semantics of the resulting phrase — its denotation — is the same regardless of the domain.

The implementation of form rules in Sparser allows them to be transparent to the rest of the rules in the grammar. The new edge created by composing the literal and the head edge will have the same category (or some linguistically principled projection of it) as that of its head edge — any semantically characterized rules that might eventually apply to the head edge will apply transparently to the new edge as though the

```
#<form-rule  fpsr1
  :righthand-side  ( #<word ''was''> #<category verb+ed> )
  :new-category :passive
  :syntactic-form  #<category verb-group>
  :referent (:head :right-edge
             :subtype #<past> ) >
```

Figure 6: The applicable form rule for "*was confirmed*"

```
#<edge   e21
  :starts-at #<position 17>
  :ends-at #<position 18>
  :category #<word ''was''>
  :left-daughter #<word ''was''>
  :right-daughter :literal-in-a-rule
  :used-in nil
  :form #<category verbal-auxiliary>
  :referent  nil >
```

Figure 7: The edge that spans the word "*was*"

literal had not been there. The observent reader will already have noticed that the set of rules above was missing the cases needed to cover many likely texts: "... *will be confirmed Wednesday in Seattle*", "... *has been confirmed*", etc. All such combinations — with time and place adjuncts, with complex verbal auxiliaries — are handled uniformly throughout the grammar by form rules.

The exceptions to this uniformity come when there needs to be a domain-specific, semantic (denotational) consequence to a particular combination, as when a verb subcategorizes specifically for a location. In such cases the grammar will simply include a semantically characterized rule for that combination, and the parser will look for that rule in preference to the form rule, as described in section 4.1.

The form rule that applies in the case of "*was confirmed*" is shown in figure 6. (Note that the referent in this rule is not a direct pointer into the model as before but a schematic function. It will take the edge on the right of the two constituents — edge-22 denoting the category 'confirm-in-position — and look in the model for one of its specializations (one of its subtypes) that represents the event having occurred in the past, which will then become the denotation of the new edge.)

Informally speaking, this rule applies in the present example because the word "*was*" is adjacent to edge-22 which has the form category 'verb+ed', and because there is no semantically characterized rule that combines "*was*" and edge-22's primary category, 'confirm-in-position'. The new edge formed by the completion of the rule will have the label 'confirm-in-position+passive', as dictated by the form rule's 'new- category' field in coordination with the schema that defined confirm-in-position and made provisions for that projection from the original category over the verb. (Had there been no new-category field in the rule, then by default the new edge would have taken the same label as the one on the head edge — edge-22.)

Before we can go into the formal algorithm for checking whether edges combine and coordinating the two kinds of rules, we must also show the edge that spans the word "*was*" and discuss its possibly unusual aspects (figure 7).

Note that the label on this edge (its 'category' field) contains a word rather than a category. There are two reasons for this. The first is that allowing words as nonterminals permits an elegant treatment of abbreviations: One can simply rewrite the combination of the abbreviation and its period as the full, unabbreviated word; thereby avoiding any need to redundantly extend grammar to include combination rules for the abbreviation as well as for the full word (e.g. "Company" -> "Co" "."). If the same abbreviation has several expansions then multiple edges will be put into the chart.

The second reason is to simplify the implementation of the parsing algorithm so that it needs to manipulate only a single data type — edges — when looking for rule applications, rather than both edges and words. Any word that is mentioned in a rewrite rule will be covered with an edge when it is scanned, the label on that edge being the word itself. This also permits a simple way to discriminate between known and unknown words in the course of a parse. A known word (one with rules in the grammar) will always have one or more edges over it, if only a trivial edge labeled with that word; unknown words will not have edges, leaving a gap among the preterminal edges of the chart, which can be used to trigger heuristics for dealing with unknown words when they appear in certain contexts.

We should also note that all of Sparser's actions are taken over unary or binary rules. Rules with more than two terms on their righthand sides are converted to a set of rules in Chomsky Normal Form, using a dotted-rule convention to provide names for the intermediate non-terminal labels created in the process. Also, in describing the Check algorithm we will not go into any detail about the larger aspects of the parsing algorithm that determines when or whether two adjacent edges will be checked. Basically the parser operates bottom up and left to right, forming constituents from their heads outwards; however, overlaying that conventional traversal of the space of possible constituents is a moderately complex control structure that reduces the search to a deterministic algorithm, in the sense that every edge formed will be part of the final analysis and every edge will have a unique parent edge in the tree. For some of the particulars see McDonald 1992.

4.1 The algorithm that coordinates checking for both kinds of rules

We can define Sparser's Check operation — its algorithm for seeing whether there is a rule in the grammar that combines two particular adjacent edges — as follows:

Consider two adjacent edges which we will call "Left" and "Right". Left ends at some position p in the chart, and Right begins at this same position p. Left is said to be earlier in the chart — closer to the beginning of the text — because it begins at some position p' (p' being strictly less that p) and because Right ends at some position p" (p" being strictly greater than p). Positions are indexed numerically starting with zero before the first word of the text and increasing in a positive direction, one position between each word. The position at which an edge begins always has a lower numerical index than the position at which it ends; no edge can begin and end at the same position.

Only adjacent edges can be combined to form a new edge. The new edge will start at position p' and end at p". The possibility of two adjacent edges combining is function of their labels, where a label is a word or category in the 'category' field of an edge, or one of a specially designated set of 'form categories' pointed to from the 'form' field of an edge. The form field may be empty.

The Check operation is based on a table compiled from the labels of the rules of the grammar. All rules are binary. The table will associate the righthand sides of those rules, taken as ordered pairs of labels, with the rules themselves. We use the table at runtime by forming an index from the labels of the adjacent constituents and looking the index up in the table to see if that pair of constituents was one of the ones defined by the grammar. If it is, we return the rule located at that index in the table; if it is not, there will be no rule at that index and we return nil indicating that the edges do not combine. We presently implement the table as a hash on the numerical index; it would lend itself to a hardware implementation as will be clear from its definition.

We form the table as follows. For each rule consider the two labels on its righthand side; call them LL and RL for the first and second (left and right) labels respectively. The rules are taken one at a time in some order: the textual order of their definition in the files of the grammar suffices. We assign each label two unique numbers. One number stands for the label when it appears as the left label of a binary rule (LL); the second stands for when it appears as the right label of a rule (RL). If a label never appears on one of the sides we assign it nil for that component. Numbers for RL labels are assigned starting at 1 and increasing by one with each successive RL up to some 'middle number' choosen to be well above the number of labels ever to appear in the grammar. Numbers for LL labels are assigned starting at that middle number plus one and adding that middle number to the prior LL index for each successive LL. The numbers are then stored with their labels for access during the Check operation.

Entries in the table are formed by adding the numbers for the two righthand side labels of each rule and assigning the rule to the table entry with that numerical index. This sum is guarenteed to be unique by the apt choice of the middle number that divides the two number sequences. It is chosen on the model of adding the two halves of a machine word on a computer. For a 32 bit word length we would select as the middle number 65,536 (two to the sixteenth), giving us a maximum of 64k labels. All of the LL labels will have zeros in their lower sixteen bits; all of the RL labels will have zeros in their upper sixteen bits; combining them with a logical AND will yield a new number unique to that combination of its two halves. Adding the two numbers in software has the same effect.

In checking whether two adjacent edges combine we first check whether their two semantic labels combine; then only if those labels do not combine will we go further and check for the possible combination of one or the other of the semantic labels with the other constituent's form label. To facilitate this, the index assignment algorithm is complicated slightly: In iterating through the grammar's rules we distinguish between regular, semantically characterized rules and form rules. A pair of indexes is stored with each label for each direction. The first number in the pair is used for combinations with a semantic label, the second with a form label. The first numbers are established by iterating through the semantically characterized rules. The second by iterating through the form rules.

Given all this, the Check algorithm proper is the following. Given edges Left and Right as defined above, compute the index in the table of the combination's semantic labels (the 'category' fields of the two edges). For Left, retrieve the right-looking number that was assigned to it — the one it was given for when it appeared as the left of the two labels in a binary rule. Do the opposite for Right. Now add the two numbers, and use the sum as an index into the table. A rule will be returned if the two edges compose, and this rule is used as the basis of the new edge spanning them. If that index into the table is nil, indicating that no semantically characterised rule applies to that edge pair, then go on to check for combinations involving form rules. In the present example there will be no semantically characterised rule for "*was*" plus 'confirm-in-position', so we will move on to check form rules.

There will always be two possible form rules to check for: one combining the category field label of Left and the form field label of Right; the other combining the form label of Left and the category label of Right. Only one of these two pairs will have an index defined for it because of the requirement on form rules that one of the two labels on their righthand sides be either a literal word or a semantic category. We start by computing the table index for the first case, and only if that lookup into the table fails to return a rule do we try the second. If one or the other label fails to have a form option in its stored pair of index numbers then that case is undefined. If neither succeeds, then the two adjacent edges do not compose.

This completes the description of Sparser's Check algorithm. It can be summarized as follows: Its purpose is to determine whether two adjacent edges can be composed to form a new edge. This is allowed in any of three cases: first if the semantic categories of the two edges match the righthand side of some rule in the grammar; failing that if either the semantic category of the first and the form category of the second correspond to some form rule in the grammar, or visa versa. The test is made by adding numbers that were assigned to the labels as the grammar rules were defined, and using that index as the key into a table that records which pairs correspond to rules.

The prohibition on having a form rule based on a pair of form labels is there to rule out the possibility of ever writing a conventional syntactic grammar in this formalism. We rule this out for two reasons. The first is that allowing "pure" form rules would re-introduce to the grammar the massive ambiguities that syntactic grammars are prone to. The second is that doing so would miss the point of this grammar design, namely that the purpose of a grammar is to facilitate the mapping between form and meaning — between the edges formed over words or phrases and their denotations in the system's world model. We only allow form rules to be used if we are already constructing the projection to some individual in the model, i.e. if the combination is between an edge on the head line which will have a projection and one of its adjuncts (broadly construed).

We have presented the Check algorithm in this much detail because we have seldom seen the rule-lookup algorithms of other parsers described and feel that this kind of information should be added to the literature. In the present case, the very high efficiency of this constant-time lookup operation is especially important because of the very large number of rewrite rules in the grammar. A lookup algorithm that was sensitive to the size of the grammar (or for that matter the presence of any 'order of the size of the grammar' operation within the parser) would strongly penalize the use a semantic grammar. Optimizing algorithms for speed is important even with today's processors because of the ever growing volume of text that must be processed.[2]

5 Defining parsing rules as a side-effect of defining a domain object type

As already discussed, the Sparser language comprehension system works in close consort with a universe of individuals and categories (the individualsU types) that supply the denotations of the words and phrases it parses. This model of the system's world (the topic domains it is competent in) and of the ongoing discourse is based on a set of type definitions, and we are using those definitions as the source of the semantically labeled rewrite rules that Sparser uses. In this section we will look at one such definition and the TAG schema that it uses to create the rules. We begin with a discussion of the schema, shown in figure 8 on page 182.

The 'binding-parameters' and 'labels' fields hold sets of symbols that will be substituted for when the schema is applied to the type-definition: The labels are used to define the rewrite patterns of the individual rewrite rules given in the 'cases' field; the parameters are placeholders for the definition's participant roles. The initial terms in each case (e.g. :subject) are just indicators used by the grammar writer to help organize the cases.

The parenthesized expressions following the indicators are the schematic form of the rules; left and righthand sides are indicated in the obvious way. The :head and :binds indicators after the syntactic part of each rule give its semantic interpretation function. They indicate which of the two edges is the head and what role is filled by the denotation of the other edge.

The label symbols on the lefthand sides of the rules (e.g. S, VP, VG) designate the form labels that are to accompany the corresponding semantic labels that are substitued for them to construct the rules. An example of the final form of one of these rules was given earlier.

[2] One day of the Wall Street Journal averages about half a megabyte. It is available online at about six o'clock in the morning. We will want the results of our information extractions and their pragmatic analysis on the desks of clients when they come in three hours later, and it is only one of any number of online text-based information services that might be used. Sparser processes the sublanguage-rich portions of a text at about 20 words a second (Mac-II, 68020, 15.7mhz); off-topic portions run at about 100 words/sec.

```
(define-exploded-tree-family  transitive/passive
  :binding-parameters ( agent patient )
  :labels ( s vp vg np-subject np-object )
  :cases
     ((:subject (s  (np-subject vp)
                 :head right-edge
                 :binds (agent left-edge))
      (:direct-object (vp  (vg np-object)
                       :head left-edge
                       :binds (patient right-edge)))
      (:passive (s  (np-object vg+passive)
                  :head right-edge
                  :binds (patient left-edge)))
      (:pos-nominalization (s  (np-object+pos vg+nominalization)
                            :head right-edge
                            :binds (patient left-edge)))
      (:of-nominalization (s  (vg+nominalization of+np-object)
                           :head left-edge
                           :binds (patient right-edge)))))
```

Figure 8: TAG schema for rule creation

Space does not permit a full explication of the relationship between this set of independent context-free rule schemas and the full clausal trees of the corresponding TAG tree family (but see McDonald, to appear). Briefly, we independently have an extensive TAG grammar, organized as tree families, that for for many years we have been using in our work on language generation. Here what we have done (by hand at the moment) is take those trees and decompose each of them into their layers of context- free rules, avoiding duplicates since many trees have identical sentential forms at some layer such as S -> NP VP.

The connectedness that characterizes the levels of immediate constituents in a TAG tree is obviously not present in the schema. It is reconstituted when the schema is applied: generic labels like VP that appear in all clauses syntactically and so can provide no connection across the cf rules are replaced during the application with unique semantically characterized labels like confirm-in-position+passive. These create a connection within the instantiated set of cf rules not unlike the quasi-nodes of Vijay Shankar (1993).

Moving on now, consider the definition of the domain-specific model-level type 'confirm-in-position', shown in figure 9 below. The representation language being used is called KRISP, and is described in McDonald (in press); it is a near cousin of Krypton in the Kl-One family, with special features that make it especially well suited as a source representation for language generation, and some efficient data storage and indexing features that suit it to language comprehension.

The first several fields of confirm-in-position are what you would expect in a modern frame language. The category's position in the taxonomic specialization hierarchy is given by its 'specializes' field. Its 'binds' field lists the participants that are specific to this event-type; although an individual instance can have additional participants like time and place by warrant of inheriting from the category TeventU. The participants' value-restrictions are indicated by the :v/r flag. The 'instantiates' field indicates which category above it in the hierarchy an instance should be indexed under in the discourse history; thus for purposes of searching the discourse history, a confirm event is taken to be equivalent to any other job-event, e.g. elect, appoint, resign, etc. The 'index' field specifies how individuals are stored.

```
(define-category confirm-in-position
  :instantiates  job-event
  :specializes   get-position
  :binds ((agent (:v/r :or board-of-directors company))
          (person (:v/r person))
          (position (:v/r position)))
  :index (:temporary :list)
  :realization
    (:tree-family transitive/passive
     :mapping ((agent . agent)
               (patient . person)
               (s . job-event)
               (np-subject . (board-of-directors company))
               (vp . job-event\agent)
               (vg . :main-verb)
               (np/object . person)
     :main-verb ''confirm''
     :saturation (agent person position)
     :additional-rules
        ((:adjunct (job-event  (job-event in-position)
                     :head left-edge
                     :binds (position right-edge)))
         (:adjunct (job-event  (job-event as-position)
                     :head left-edge
                     :binds (position right-edge))) )))
```

Figure 9: The domain-specific model-level type 'confirm-in-position'

The 'realization' field is the basis for instantiating the category's portion of the semantic grammar. The rules constructed from it will recognize phrases denoting instances of individuals with this category in a text, and will guide the semantic interpretation to instantiate them and populate their participant roles. The field indicates which syntactic rule schema should be used, in this case the tree-family 'transitive/passive' shown just above. In its 'mapping' field it indicates the correspondences between the roles of the category and the substitution variables in the schema. The 'main-verb' field (vs. 'head-noun' in other cases) indicates what word should be used as the lexical head of the tree family; its morphological variants are calculated automatically, though if it were irregular the special cases would be given explicitly here. The 'saturation' field indicates that those three roles must be given in a text before an individual of this type is fully defined.

When this form is executed to define the category, its realization field is interpreted and the syntactic schema (tree family) it indicates is applied to the parameters given in the field. Each of the rule cases of the schema is taken up in turn and the corresponding rewrite rule created by making the substitutions called for by the mapping field. Notice that there are two additional cases given in this category's realization field. They identify how the category's 'position' role is filled, namely with either of two different adjunct prepositional phrases, one using "*in*", the other "*as*". (They are analyzed here as attaching to the sentence rather than the verb phrase for compatibility with the way nominalizations are treated.) This kind of local augmentation of the general syntactic schemas (the set of tree families) is convenient for specifying the often ideosyncratic way that individual verbs can subcategorize for optional complements.

Let us look at one case, the first one in the schema that specifies S -> NP-subject VP. (The NP is annotated as the subject to distinguish it among the substitution variables from the other NP, the direct object.) Looking to the mapping specification we see we have to make two rules because the subject np can take either of two values.We form the lefthand side of the both by substituting the category 'job-event' for S. We form the righthand side of one by substituting the category 'board-of-directors' for the variable 'np-subject' and substituting 'job-event\agent' for 'vp'; the other rule gets 'company' substituted for its np-subject.

The rule must also have a syntactic-form category, which will be the form label of the edge created when the rule completes. It takes this from the syntactic category (substitution variable) given as the rule's lefthand side in the schema (i.e. S). The rule also must have a referent, which it gets from the corresponding part of the case in the schema, subsituting its role 'person' for the symbol 'patient' given in the schema, again as indicated by the mapping field.

We should note in closing this section that a number of largely arbitrary decisions were made in the writing of this grammar, which is to say in the selection of what semantic categories to use in stating the correspondences between semantic labels and syntactic labels in the mapping field of a definition like this. For example we have made a generalization about the semantic interpretations of a whole semantic class of verbs by deciding to specify the VP label as 'job-event\agent' rather than, say, 'confirm-in- position' (which would be as specific as we would be able to be). By choosing the more general category (which should be read as in a categorial grammar: look to compose with an agent to the left of this constituent and label the new edge 'job-event'), we are saying that all the verbs in this family (or rather their model-level denotations) make the same interpretation of their subject NP, i.e. they all map it to their equivalent of the agent role. This points out that linguistic generalizations can be made in the semantic realm as well as the syntactic, and that by using a semantic grammar we are able to express this in a very direct fashion.

6 Concluding remarks

Two substantive complaints have been made against semantic grammars in the literature (e.g. Wallace 1984). One is that they cannot be easily extended as one moves from one domain to the next, while a thorough syntactic grammar can be taken over unchanged. This observation misses a crucial point, however, namely that to understand a text one must find the correspondence between the structural analysis provided by the syntax and the individuals and domain-types in the world model of the application system that is going to use the information given in the text. There is no question that the world model of the first domain that one works with will have to be extended when one moves to the next — the next domain will involve new kinds of objects and new types of relations, and these will have to be added to the model if texts in the new domain are to be understood. There is no other possibility and the fact that there may be no change in the syntactic grammar is besides the point because it is not the whole story.

By using a semantic grammar and having it constructed directly from the world model as done here with Sparser, we will able to keep the grammar in step with the extensions to the model. In particular, we will be continually reminded, as we work on the model, that we need to work out the words and syntactic constructions used to talk about the new concepts, since each domain type should have a realization field to indicate this information.

The second substantive complaint is that semantic grammars are incapable of capturing linguistic generalizations, and so one cannot take advantage of these uniformities when writing the grammar, and one may be tempted to write rules that would be unreasonable from a linguistic perspective and so likely to be brittle in practice (e.g. defining constituents that combine an np with a following preposition, leaving the preposition's complement stranded). This is the more legitimate of the two complaints, but it is also specifically what the present system was designed to address.

By keeping the grammarian (domain modeler) from writing rules directly except in the most ideosyncratic cases (e.g. for dates written as "3/31/93"), and instead forcing them to work by way of some well-crafted syntactic schemas, we have imposed a discipline on the grammar, forcing it to fit our conception of properly analyzed linguistic structure as captured in the schemas. At the same time we have reduced the overall effort required, since it is markedly more dispatchful to copy and specialize the realization fields of a set of related domain-type definitions than to write out the rules individually.

We have been using semantic grammars in the Sparser system since its inception three years ago. Form rules were introduced after the experiences with its first major grammar, and had a significant effect in simplifying the effort to develop new vocabulary since they allowed whole sets of rules for generic constructions to be used just by giving new semantic categories in the grammar the appropriate syntactic labels. The use of realization fields on model-level definitions and the development of the 'exploded tree families' used as schemas is comparatively recent, and we have only now redone the original grammar for the Who's News domain in its terms. The proof of the pudding, as it were, will come as the models and grammars for the next topic domains are added in the coming months (these will be for joint-ventures and quarterly earnings). If the effort to make these extensions is dramatically smaller than when the original grammars were developed by hand we will judge the design a pragmatic success. If a large family of syntactic schemas, especially for noun phrases, can be developed that addresses the bulk of the construction-types that we see in the business texts we work with then we will have gotten a long way towards solving the problem of extending grammars to new domains.

References

Abeille, A. (1988) *A French Tree Adjoining Grammar*. technical report. Department of Computer & Information Science, University of Pennsylvania.

Burton, R. (1976) *Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems*. Report No. 3453, Bolt Beranek and Newman Inc, Cambridge, MA.

_____ — J. S. Brown (1975) "Multiple Representations of Knowledge for Tutorial Reasoning", In: Bobrow & Colins (eds.) *Representation and Understanding*. New York: Academic Press.

Hendrix, G. — E. Sacerdoti — D. Sagalowicz, — J. Slocum (1978) "Developing a natural language interface to complex data" ACM TODS. 3(2), 105 - 147.

Joshi, A..K. (1985) "How much context-sensitivity is required to provide reasonable structural descriptions: tree adjoining grammars". In: Dowty, D.R., L. Kartutunen, A.M. Zwicky (eds) *Natural Language Processing*. Cambridge, U.K.: Cambridge University Press.

Kittredge, R. — J. Lehrberger (1982) *Sublanguage: Studies of Language in Restricted Semantic domains*. Berlin: de Gruter.

McDonald, D. (1992) "An Efficient Chart-based Algorithm for Partial-Parsing of Unrestricted Texts". In: *Proceedings of the 3d Conference on Applied Natural Language Processing* (ACL). Trento, Italy. April 1992. 193 - 200.

_____ (to appear) "Reversible NLP by Linking the Grammar to the Knowledge Base". In Strzalkowski, T. (ed), *Reversible Grammar in Natural Language Processing*. Kluwer Academic.

_____ (in press) "KRISP: a representation for the semantic interpretation of real texts". *Mind and Machines*.

Riesbeck, C. — C.E. Martin (1985) "Direct Memory Access Parsing". technical report DCS/RR 354. Department of Computer Science, AI Group. Yale University.

Schabes, Y. — A. Abeille — A.K. Joshi (1988) "Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars". In: *Proceedings of Coling-88*, Budapest, Hungary.

Vijay-Shanker, K. (1993) "Using Descriptions of Trees in a Tree Adjoining Grammar". *Computational Linguistics*. 18(4), 481 - 517.

Wallace, M. . (1984) *Communicating with databases in natural language*. Chichester, U.K.: Ellis Horwood.

Waltz, D. — T. Finin — F. Green — B. Goodman — G. Hadden (1976) *The PLANES system: Natural language access to a large database*. Report T-34. Department of Computer Science, University of Illinois.