Deep parsing in Watson

M. C. McCord
J. W. Murdock
B. K. Boguraev

Two deep parsing components, an English Slot Grammar (ESG) parser and a predicate-argument structure (PAS) builder, provide core linguistic analyses of both the questions and the text content used by IBM Watson™ to find and hypothesize answers. Specifically, these components are fundamental in question analysis, candidate generation, and analysis of passage evidence. As part of the Watson project, ESG was enhanced, and its performance on Jeopardy!™ questions and on established reference data was improved. PAS was built on top of ESG to support higher-level analytics. In this paper, we describe these components and illustrate how they are used in a pattern-based relation extraction component of Watson. We also provide quantitative results of evaluating the component-level performance of ESG parsing.

Introduction

Two deep parsing components, an English Slot Grammar (ESG) parser and a predicate-argument structure (PAS) builder, provide core linguistic analyses of both the questions and the text content used by IBM Watson* to find and hypothesize answers. Specifically, these components are fundamental in question analysis, candidate generation, and analysis of passage evidence [1–3].

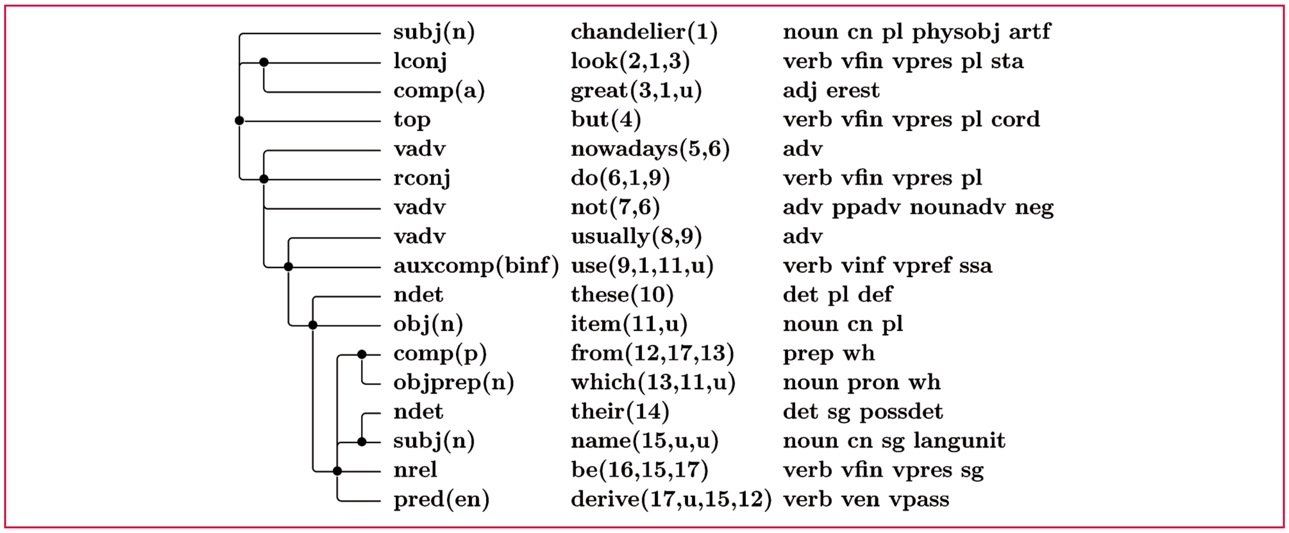
ESG [4–7] is a deep parser in the sense that the parse trees it produces for a sentence (or segment of any phrasal category) show a level of logical analysis (or deep structure). However, each parse tree also shows a surface-level grammatical structure (surface structure), along with the deep structure. The parse trees for a segment are ranked according to a parse scoring system (described below), and for Watson, we use only the highest-ranked parse. (A parse score roughly corresponds to the likelihood that the parse is a correct one.) In this paper, we provide an overview of Slot Grammar (SG) in its current state, discussing new features and special adaptations made for the Jeopardy!** question-answering (QA) task. Most of the improvements motivated by the Jeopardy! challenge are applicable to general English and other applications. The adaptations that are really special to Jeopardy! questions can be controlled by flag settings, which are off by default and can be turned on when ESG is used for the Jeopardy! task.

Parse analysis by ESG is followed by the application of a PAS builder, which simplifies and abstracts from the ESG parse in a variety of ways; for example, it drops some terms (e.g., auxiliary verbs) that are rarely very important for the tasks that our downstream components perform. The active/passive alternations such as "John sold a fish" and "A fish was sold by John" have slightly different structures in ESG but the same structure in PAS.

The deep parsing suite for Watson consists of ESG, followed by the PAS builder. Deep parsing results are pervasively used in the Watson QA system, in components within every stage of the DeepQA architecture [8]: question analysis, question decomposition, hypothesis generation, hypothesis and evidence scoring, etc. Here are a few specific examples.

- Relation extraction (see [9] and the section on relation extraction below) identifies semantic relationships (in the sense of that section) among entities using the results of deep parsing.
- Question analysis [1] uses results of deep parsing to identify the type of answer that a question is seeking.
- The keyword search component [2] uses semantic relations in the question to identify keywords that have some strong semantic connection to whatever the question is asking for; those keywords are given a higher weight in the search query.
- Passage-scoring components [3] use the results of deep parsing on both the question text and the passages found by keyword search to determine whether a passage aligns well to the question and thus provides evidence in support of some candidate answer.

Digital Object Identifier: 10.1147/JRD.2012.2185409



```
subj(n)         chandelier(1)         noun cn pl physobj artf
lconj           look(2,1,3)           verb vfin vpres pl sta
comp(a)         great(3,1,u)          adj erest
top             but(4)                verb vfin vpres pl cord
vadv            nowadays(5,6)         adv
rconj           do(6,1,9)             verb vfin vpres pl
vadv            not(7,6)              adv ppadv nounadv neg
vadv            usually(8,9)          adv
auxcomp(binf)   use(9,1,11,u)         verb vinf vpref ssa
ndet            these(10)             det pl def
obj(n)          item(11,u)            noun cn pl
comp(p)         from(12,17,13)        prep wh
objprep(n)      which(13,11,u)        noun pron wh
ndet            their(14)             det sg possdet
subj(n)         name(15,u,u)          noun cn sg langunit
nrel            be(16,15,17)          verb vfin vpres sg
pred(en)        derive(17,u,15,12)    verb ven vpass
```

Figure 1

ESG parse of the Jeopardy! question "Chandeliers look great but nowadays do not usually use these items from which their name is derived."

- Some of the type coercion components [10] use the PAS to compare the type requested to answer types found in natural-language text.
- The results of PAS (and relation extraction) across a large corpus are aggregated in the PRISMATIC knowledge base [11], which is used by a variety of search [2] and answer-scoring [3, 10] components.

The pervasive usage of deep parsing results reflects the fact that these components provided the core natural-language processing capabilities for the Watson QA system.

In this paper, we first describe SG parsing for Watson, and then we discuss PAS, followed by an approach to relation extraction that illustrates the use of ESG and PAS. These main sections are followed by sections on use in Watson, evaluation, related work, and conclusion and future work.

SG parsing

The SG parsing system is divided into a large language-universal *shell* and language-specific grammars for English, German, French, Spanish, Italian, and Portuguese. Some of the SG features described in this section are in the shell, and some are specific to English (ESG); all of the examples are for ESG. We discuss 1) the pipeline of SG parsing, 2) the nature of SG parses, 3) the lexical system, and 4) syntactic analysis. At the end of this paper, we describe an evaluation of ESG performance.

Pipeline of SG parsing

The main steps of SG parsing are (A) tokenization and segmentation, (B) morpholexical analysis, and (C) syntactic analysis. Step (A) is self-contained and can handle various tagging systems (such as HTML). Its output is directly used in some components of Watson. Unlike some parsers, SG uses no part-of-speech (POS) tagger; the corresponding information simply comes out of syntactic analysis. In the following, after describing the nature of SG parse trees, we concentrate on the lexicon and syntactic analysis.

Nature of SG analyses

Figure 1 shows a sample Jeopardy! question and its ESG parse tree. We look at the example and give an overview of SG parses in general.

An SG parse tree is a *dependency tree*: Each tree node N is centered on a *headword*, which is surrounded by its left and right *modifiers*, which are, in turn, tree nodes. Each modifier M of N fills a *slot* in N. The slot shows the grammatical role of M in N. In our example, the node with headword "chandelier" fills the `subj` (i.e., subject) slot for the coordinated VP (verb phrase) node with headword "but". This modifier tree structure is the *surface structure* of the parse analysis. In our sample parse display, you can see, on the left side, the surface structure tree lines—each line connecting a node M to its mother node N and showing the slot filled by M in N.

Slots are of two kinds: complement slots and adjunct slots. Complement slots, such as `subj` and `obj` (i.e., direct object) for verbs, are idiosyncratic to senses of their headwords and are associated with these senses in their lexical entries. Adjunct slots, such as `vadv` (verb-modifying adverbial), are associated with the POS of the headword sense in the SG syntax module. Adjunct slot-fillers can typically modify any node of the category (POS) with which they are associated. Complement slots play a dual role in SG: They can name grammatical roles, as mentioned. In addition, they can name logical arguments of word senses, as described later in this section.

In the sort of parse display given in Figure 1, the lines/rows correspond 1-to-1 to tree nodes.

We now describe the five main ingredients associated with a parse node, and then we state which parts constitute deep structure and which constitute surface structure.

(1) *The headword of the node*—The internal parse data structure stores several versions of the headword, including a) the form of it as it occurs in the text (inflected, mixed case, etc.); b) the lemma (citation) form; and c) the SG word sense of the node, which we explain below in the "SG lexicons" subsection. Typically, the headword comes from a single-word token, but it may be a multiword or a punctuation symbol acting as a coordinator or a special symbol for a special kind of node, such as a quote node as described below. In the above form of parse display, the headword is shown in lemma form, but there are display options to show other forms. The headword is seen in the middle column as a predicate followed by arguments, for example, in

```
derive(17, u, 15, 12).
```

We call this predication the *word-sense predication* for the node, and it is the main vehicle for showing the deep structure of the parse.

(2) *The ID of the node*—This is an integer that, in most cases, is the word number of the headword in the segment, but there are exceptions, with the most common being for multiwords, where the ID is the word number of the head of the multiword. In this parse display, the node ID is shown as the first argument of the word-sense predication, for example, `17` in `derive(17, u, 15, 12)`.

(3) *The (logical or deep) argument frame of the node*—In the internal parse data structure, this consists of the list of complement slots of the word sense, each slot being associated with its filler node (or `nil` if it has no filler). In the "derive" node of our example, this list of pairs would be `(subj: nil, obj: ph15, comp:ph12)`, where `ph15` is the phrase with node ID 15, spanning "their name", and `ph12` is the phrase with ID 12, spanning "from which". The `subj` slot has no overt filler. Note that "derive" is given in the passive, but these three slot-fillers constitute the *logical* (active form) arguments of the verb. For example, `ph15` is the logical `obj` of "derive", although grammatically, it is a `subj` (of "be"). That is why we speak of the *logical* or *deep* argument frame of the node. For a verb, the first member of its argument frame is always its logical subject.

Now we can say what the word-sense predication of a node is in the above form of parse display. The predicate name is the word sense or, optionally, the citation form. The first argument is the node ID. The remaining arguments are the IDs of the filler nodes in the argument frame or `u` (for "unfilled" or "unknown") if there is no filler.

The word-sense predication can be directly translated into a logical predication. We can replace the numerical arguments by similarly indexed logical variables, for example, as in derive$(e17, x, x15, x12)$, where, in general, derive(e, x, y, z) means that e is an event where x derives y in manner z. Hence, the node ID argument can be thought of as an event argument or, more generally, an *entity* argument for the predication. Note that, in the example, "chandeliers" (node 1) is shown as the logical `subj` of the predicates for "look", "do", and "use", although in surface structure, its only role is as the (grammatical) `subj` of the coordinated node 4. In handling coordination, the SG parsing algorithm can "factor out" slots of the conjuncts. This happens with nodes 2 and 6, providing the common `subj` filled by node 1, but still showing 1 as logical `subj` for each conjunct. SG parsing also fills in implicit arguments for many nonfinite VPs, and this results in 1 being the logical `subj` of node 9.

The sample parse shows two other kinds of implicit arguments filled in: a) The predication `great(3, 1, u)`, where "chandeliers" (`1`) fills the first slot `(asubj)` of the adjective "great", directly shows that "great" applies to "chandeliers" (under the context of "look"). b) The predication `which(13, 11, u)`, where "items" (`11`) fills the first slot (`nsubj`) of the relative pronoun "which", is interpreted as showing that the relative pronoun is co-referent with "items". Then, in building a logical form, the relative pronoun's variable can simply be replaced throughout with the variable for "items".

(4) *The features of the node*—In our parse display, the node's features are listed to the right of the headword. These can be morphosyntactic or semantic features. (In this example, some features were omitted for brevity's sake.) The first feature listed is the POS of the node. Most of the features come from those of the headword sense, as obtained from morpholexical analysis of the headword, but some may be added during syntactic analysis when the node acquires modifiers.

(5) *The (surface) modifier structure for the node*—In the internal parse data structure, a node N has two associated lists—for its left modifiers (premodifiers) and its right modifiers (postmodifiers), where each modifier node is paired with the slot it fills in N. In our parse display,

it should be clear how to read the tree structure from the lines and dots on the left of the display (picture a tree diagram in standard form turned on its side). The slot shown closest to the headword of a node *N* is the slot that *N* fills in its mother node. For each complement slot *S*, the *slot option* used for *S* is shown in parentheses after *S*. For instance, node 17, for "derived", fills slot `pred(en)`, meaning that node 17 fills a past-participial form of the `pred` (predicate) slot—for "be" (in node 16). More information about slot options is given in the next subsection.

The core of the SG deep structure is the set of word-sense predications described in (3) above, since these are close to logical predications. Deep structure information also exists in the semantic features of nodes. However, even some morphosyntactic features, such as tense and number, matter for logical form. The core of the surface structure lies in the headword information (1) and (2), the morphosyntactic features, and the surface modifier structure (5). However, adjunct slots appearing in (5) can also be of relevance to deep structure, because, e.g., determiners may produce quantifiers in logical form.

SG lexicons

In this subsection, we describe the SG lexical system and improvements made to it to benefit Watson. Much of the SG analysis process is driven by the lexicons used, particularly because SG lexicons specify (complement) slot frames for word senses, and the main step in syntactic analysis is slot-filling.

SG lexical entries are typically indexed by citation forms of words or multiwords. Morpholexical analysis of tokens does efficient lookup in the lexicons, along with morphological analysis, both inflectional and derivational. ESG morphology currently handles 29 derivational affixes.

For any language version of SG (such as ESG), there is a main lexicon called the *base* lexicon. The system allows any number of lexicons; ones besides the base lexicon would typically be user addendum lexicons. The ESG base lexicon has approximately 87,000 entries, but many more word forms are recognized because of the derivational and inflectional morphology.

In the work on Watson, we have developed a way of augmenting (i.e., expanding and improving) the ESG base lexicon automatically from other sources, particularly Princeton WordNet** [12, 13]. The process of augmentation is done before run time for any new version of the base lexicon and takes only about 5 seconds on a standard desktop. We describe the augmentation methods in this subsection.

In the following, we describe (A) the form of SG lexical entries and then (B) improvements made during the work on Watson.

Form of SG lexical entries

The following is a sample entry (slightly simplified) from the ESG base lexicon:

```
talk < v (obj n (p about)) (comp (p to with))
     < v obj1 (comp1 (p into))
     < n nsubj (nobj n (p about))
       (ncomp (p to with))
```

In general, a lexical entry has an *index word*, given in citation (lemma) form, and can be a single word or a multiword—`talk` in our example. This is followed by a sequence of *sense frames* for the word—three in our example, two verb frames and one noun frame. Each sense frame can specify any of the following seven kinds of items, all of which are optional except the first: (1) POS, (2) complement slot frame, (3) features—both semantic and syntactic—(4) word-sense name, (5) numerical score, (6) subject area test, and (7) generalized support verb construction. Our sample shows only (1) and (2) in the sense frames. Each sense frame defines an *SG word sense* for the index word. The ESG lexical word senses are rather syntactic in nature, although the differing slot frames do constrain the possible semantic word senses. However, the SG framework allows finer semantic distinctions in its word senses, because slot options can make semantic type tests on the slot's fillers. This is done to some extent in the ESG lexicon.

Now, let us look in more detail at the seven kinds of items in a sense frame.

Part of speech

In parse data structures, there are 15 possible parts of speech that include `noun`, `verb`, `adj`, `adv`, `qual` (qualifier), `det`, `prep`, `subconj` (subordinating conjunction), and `conj` (coordinating conjunction). Some of these are seen in the sample parse tree of Figure 1, where the POS is listed as the first of the nodes' features. The lexicon uses these same POS names, except in the case of nouns and verbs, for the sake of brevity. For instance, the lexical POS `n` is expanded into `noun cn` (common noun) in parse trees, and `v` expands into `verb`. Other features, such as number and tense, are added on the basis of morphology.

Complement slot frame

In our sample entry, the first sense frame for `talk` shows two slots, namely, `obj` and `comp`, in its slot frame. An initial `subj` slot is implied; every verb has a `subj` slot, and this can be omitted as long as it needs no special options. The `obj` slot shown has two options: `n` and `(p about)`. The first allows NP (noun phrase) fillers for `obj` (plus some other nominals such as gerund phrases), and the second allows "about"-PPs (prepositional phrases). The options are disjunctively viewed. The `comp` slot has option `(p to with)`, which allows both "to"- and "with"-PPs. Thus, the slot frame allows variants such as "John talked (about) mathematics to/with Bill".

In general, a slot has an associated list of possible (slot) options, which are symbols that name constraints on the possible fillers of the slot. This applies to adjunct and complement slots (the options for each adjunct slot are specified along with the slot in the syntax module). The idea for complement slots is that the slot deals with one argument of the word sense, which can be realized in different ways syntactically. Slot options can specify not only the basic syntactic category of the filler but also many other kinds of tests on the filler, such as semantic type requirements, other feature tests, subject area tests, tests for specific words, or recursively any Boolean combination of tests. Our example shows how lexical entries name specific options for complement slots. If none is specified, as in the `obj1` slot in the second sense frame for `talk`, then default options are assigned by the system.

Most slots are optional by default, meaning that they are not required to be filled for a valid use of the sense frame. This applies to the two slots listed in the first sense frame of our example. A suffix `1`, as in our second sense frame, indicates that the slot is obligatory—that it must be filled.

Features
The features can be syntactic features or semantic types. ESG currently uses approximately 160 semantic types, belonging to a type hierarchy. Examples can be seen in the sample parse in Figure 1, e.g., `artf` (artifact) and `langunit` (language unit). The types are mainly high-level and include, e.g., physical object; substance; abstraction; property; natural phenomenon; event; act; change; various types for time, location, and measures; collection; living being; human; (nonhuman) animal; communication; feeling; profession; artifact; artistic composition; and others. An important type for its effect in parsing is "role entity"—an entity (usually a person) viewed as having a particular role ("teacher", "leader", "mother", etc.). For more details, see [7]. The features appearing in a sense frame get transferred to any one-word phrase having that word sense as head and used as a "starter phrase" in syntactic analysis. Several of the ESG syntax rules test on these semantic types.

Word-sense name
If this is omitted, the word-sense name is taken to be the index word of the entry with a suitable integer appended. When the current sense frame is converted to a "starter" phrase for parsing, this item becomes the word sense of that node, mentioned in the preceding subsection. Syntax rules can test on specific sense names for words.

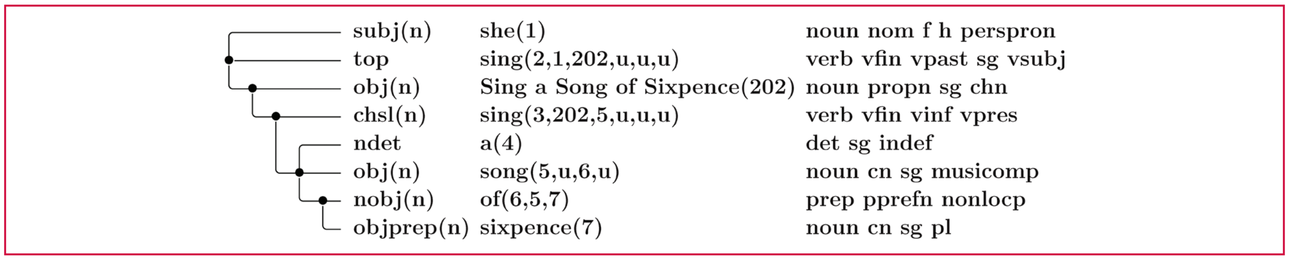
Numerical score
This can feed into the parse scoring system described below and may reward or penalize uses of the current sense frame.

Subject area test
If a subject area (domain, topic) for the current text is specified or determined automatically (e.g., `computers` or `medicine`), then this kind of test can allow/exclude the current sense or give it a numerical score.

Generalized support verb construction
In "make a reference to", the noun "reference" idiosyncratically uses the support verb "make". A special keyword allows one to store this information under the noun "reference", thus not overloading the common verb "make". The mechanism is more general than what can be handled by multiwords, because it allows variations such as interspersed modifiers and extraposition of the noun in "wh" questions. The mechanism not only applies to support verbs for nouns but can also involve any pair of parts of speech.

Improvements
Now, we describe special improvements made to the ESG lexical system during the work on Jeopardy!. The first four of these are of value for general use of ESG (including Jeopardy!), and the fifth is special to Jeopardy!.

Matching noun frames with verb frames
The ESG lexicon is well-populated with complete slot frames for all open-class word senses—for nouns, adjectives, and adverbs, as well as for verbs. Special effort was made in working on Watson to encode slot frames for de-verbal nouns (such as "celebration" for "celebrate") and other verb-related nouns in a way that properly corresponds to verb frames. Relating nouns and verbs this way can help in relation extraction or in matching questions to answers, because the same relation plus arguments may be expressed as a verb in one place but as a corresponding noun in another. This kind of correspondence can be seen in the above lexical entry for `talk`, where the slot frame of the noun sense corresponds to the first verb sense, with the noun slots `nsubj`, `nobj`, and `ncomp` corresponding, respectively, to the verb slots `subj`, `obj`, and `comp`. No options are specified for `nsubj`; hence, it takes its default options, which are `agent` and `n`. The `agent` option is filled by "by"-PPs, as in "a talk by John". The `n` option for both `nsubj` and `nobj` is filled by "of"-PPs; hence, there can be ambiguity (as in "the love of God"); however, it is disambiguated, for example, in "a talk of John's" or in "the choice of the department for their chairperson". Possessive noun premodifiers can also fill `nsubj` implicitly. For example,



```
subj(n)      she(1)                          noun nom f h perspron
top          sing(2,1,202,u,u,u)             verb vfin vpast sg vsubj
obj(n)       Sing a Song of Sixpence(202)    noun propn sg chn
chsl(n)      sing(3,202,5,u,u,u)             verb vfin vinf vpres
ndet         a(4)                            det sg indef
obj(n)       song(5,u,6,u)                   noun cn sg musicomp
nobj(n)      of(6,5,7)                       prep pprefn nonlocp
objprep(n)   sixpence(7)                     noun cn sg pl
```

Figure 2

Illustration of chunk-lexical analysis, with underlying structure of chunk.

the NP "John's talk with Mary about the book" corresponds to the VP "John talked with Mary about the book", and in the NP, "John's" fills `nsubj` for the noun "talk". Some of the nouns with verb-like frames are morphologically derived from verbs, as with "discussion"/"discuss". Others, as with "talk", may use the same word or a word that is etymologically related but not involving regular affixation.

Augmentation by WordNet
As mentioned above, WordNet is used in augmenting the ESG base lexicon. The main augmentation process is completely automatic and can be run by any ESG user with a recent version of WordNet from Princeton. The augmentation has two aspects: increasing the number of entries and marking the augmented lexicon systematically with semantic types. The new entries consist of proper nouns and certain multiword common nouns in the WordNet vocabulary that are not already in the ESG base lexicon. The extra semantic type marking is done as follows: ESG has an attached mapping of WordNet senses (or synsets) to some of the ESG semantic types. Given a lexical index word W and SG sense frame for W, the augmentation algorithm looks at the WordNet senses of W for the POS of the sense frame; and, for each such S, goes up the WordNet hypernym chain of S until it (possibly) finds a synset that is mapped to an ESG type T; and then marks T on the given sense frame of W. The process takes account of synset frequencies and the number of WordNet senses for a word, to avoid marking via too-rare senses. ESG runs well without this augmentation, but the augmentation increases a parse accuracy score, as described in the "Evaluation" section below.

Noun-verb correspondences
Augmentation of the base lexicon also includes the addition of four kinds of relationships between open-class words: `nform`, `vform`, `ernform`, and `ervform`. A verb V is given the compound feature (`nform` $N_1 \ldots N_m$) when the nouns $N_1 \ldots N_m$ are nominal forms of V. For example, the verb "defer" is given the feature

`(nform deferral deferment deference)`.

The feature `vform` provides an inverse of this. A verb V is given the feature (`ernform` $N_1 \ldots N_m$) when the nouns $N_1 \ldots N_m$ are agentive nouns for the action of V. For example, "celebrate" is given `(ernform celebrator celebrant)`. The feature `ervform` provides an inverse. The auxiliary lexical files used for this augmentation come from the WordNet link for derivationally related words, plus quite a bit of editing. These relations are not used in parsing, but they are provided in the ESG parse output and can be useful downstream when Watson is matching questions to possible answers, and the same concept is expressed as both a verb and a noun.

Chunk lexicons
SG lexical coverage can also be increased by another device—chunk lexicons—which use a storage and lookup scheme that allows a very large number of multiword entries. For the Jeopardy! application, ESG uses a chunk lexicon, ch.lx, of proper nouns derived from Wikipedia** anchor texts, with approximately 1.4 million entries. In the parse trees, the chunk entries found are treated like multiwords, but the parse also shows their underlying syntactic structures. For instance, "Sing a Song of Sixpence" is in ch.lx, and the ESG parse for "She sang Sing a Song of Sixpence" is as shown in **Figure 2**. Note that "Sing a Song of Sixpence" is shown as a multiword proper noun, but its underlying VP structure is also shown, under the slot `chsl`.

A chunk lexicon has also been created for the Unified Medical Language System (National Institutes of Health, U.S. National Library of Medicine, http://www.nlm.nih.gov/research/umls/).

LAT reward features
This improvement is specific to the Jeopardy! application. The augmentation of the ESG base lexicon also includes

the addition of compound features that help the parser choose NP analyses that lead question analysis to better identification of lexical answer types (LATs) [1] (a LAT is a term in a question that indicates what type of entity is being asked for). A *LAT reward feature* on a noun sense N is of the form (`latrwd` R), where R is a floating-point number. When syntactic analysis forms an NP with N as head and with "this" or "these" as determiner and certain other constraints are satisfied, the parse score of the NP is made better by amount $c \star R$, where c is a certain constant (experimentally determined by parse testing)—thus rewarding N in a typical LAT configuration. For a given N, the reward number R is computed from a frequency table of known LATs occurring in Jeopardy! questions. R is taken as `sqrt`(F/F_0), where F is the frequency of N in the table, and F_0 is the highest frequency occurring in the table. No `latrwd` is added if N does not occur in the table.

SG syntactic analysis

After tokenization, segmentation, and morpholexical analysis, the main steps of syntactic analysis begin.

The first step is to convert the morpholexical analyses of tokens into one-word *phrases*, where a *phrase* is the main data structure that syntactic analysis works with. These can form "starter" phrases for syntactic analysis.

The next step is multiword agglomeration, which converts some sequences of one-word phrases into phrases that span multiwords. The most common multiword agglomeration is based on multiword entries in lexicons, but it can also be done on the basis of general rules, such as rules for human names, dates, and literal numbers. The operation of lookup in chunk lexicons (described in the previous subsection) also occurs at this stage. Multiword agglomeration can also be nondeterministic, in the sense that syntactic analysis sees both agglomerated and nonagglomerated spans. Local syntactic analysis is performed at this stage for chunk multiwords.

Then the main SG syntactic analysis is performed—via bottom-up left-to-right chart parsing, where the units worked with are of data type *phrase*. The main kind of step is a binary one, where a *modifier* phrase fills a slot in an adjacent *matrix* phrase (on the left or right). The slots tried are those from the *available (complement) slots list* (ASL) of the matrix and from the adjunct slots associated with the POS of the matrix. The ASL of a one-word phrase is just its complement slot frame. When a slot from the ASL is used, it is removed from the ASL of the augmented version of the matrix with the filler modifier attached. However, the ASL in the augmented matrix can *gain* members if the slots are extraposed out of the modifier phrase, e.g., for handling "wh" phrases. Moreover, in coordination, slots can be extraposed out of the conjuncts and coalesced, as in "John saw and Mary heard the train", where "the train" simultaneously fills the `obj` slot of the two conjuncts. In the modification process, *all* slots of the matrix are tried, nondeterministically. And for each such slot, all of its options are tried, normally in an if-then-else way. *Slot-filler rules* in the grammar (which is rule-based) determine the constraints on filling slots plus options.

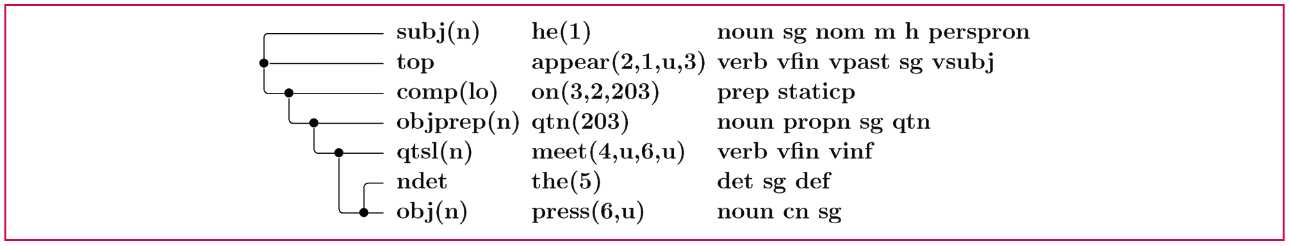
Although the slot frame is ordered, the actual fillers in a phrase may appear in various orders within the sentence and use various slot options. Fillers may come from remote parts of the sentence, as in "What did you say she sees", where "what" fills the `obj` of "see" remotely.

SG parsing uses a numerical scoring system, where each (intermediate or final) phrase is assigned a *parse score*. This has two purposes: First, the final parses of a segment are numerically ranked. Most applications of SG, including Watson, use only the highest-ranked parse for each segment. Second, during parsing, intermediate phrase analyses can be pruned away from the parse space (chart) when their scores fall too low compared with competing analyses for the same span, headword, and POS. Parse space pruning greatly increases efficiency (in both time and space) and affects the parse outcomes. However, in order to see more parses and at the risk of overload for very long sentences, one can turn pruning off by a flag setting. The base parse scores can come from several sources, for example, general rules in the shell (such as preference of complement slots over adjunct slots), specific scoring in the grammar, and rewards or penalties given in the lexicon (such as from `latrwd` features). In addition, parse scoring can take input in a bootstrapping way from data gathered from parsing (by ESG itself) of large corpora.

If at the end of the chart parsing of a segment, there is no phrase built that spans the whole segment, then SG forms a pieced-together parse on the basis of preferences for maximally spanning pieces and other heuristics involving pieces' POS.

During the main step of parsing, SG can analyze multiword units in yet another way. On the basis of (partial) capitalization and possible quoting, SG can form *quote nodes*, which are proper noun nodes, but show an internal syntactic structure underneath. For instance, for the sentence "He appeared on Meet the Press", ESG analyzes "Meet the Press" as a proper noun unit—a quote node—but also shows the VP structure underneath. The parse tree is shown in **Figure 3**. Quote node formation is similar in results to the use of chunk lexicons, as described above, but does not rely on multiword lexicons. Quote nodes are formed during regular parsing, and decisions may be made on what fits best in the overall parse.

One special adaptation of the ESG parsing process for the Jeopardy! question domain is to accommodate the relatively frequent occurrence (compared with general English) of segments in Jeopardy! that are NPs instead of complete sentences. An example is "Number of poems



```
subj(n)      he(1)              noun sg nom m h perspron
top          appear(2,1,u,3)    verb vfin vpast sg vsubj
comp(lo)     on(3,2,203)        prep staticp
objprep(n)   qtn(203)           noun propn sg qtn
qtsl(n)      meet(4,u,6,u)      verb vfin vinf
ndet         the(5)             det sg def
obj(n)       press(6,u)         noun cn sg
```

Figure 3

Parse showing a quote node.

Emily Dickinson gave permission to publish during her lifetime". The idea is that the whole NP uniquely characterizes the answer. One heuristic (among several) used for boosting the scores for such NP analyses is to reward them if the segment contains no "reference words" such as "this", "he", "she", etc., which typically signal subphrase foci in sentence questions. These heuristics are controlled by a flag that is off by default and is turned on for the Jeopardy! task.

Predicate-argument structure

The PAS builder provides simplification and abstraction of the ESG parse that removes some details. By design, the semantic distinctions that are removed in the PAS are ones that are subtle and not essential to the coarse-grained distinctions we make in many Watson components that use the parses. Without these distinctions, those components can be more flexible and require less knowledge. For example, passive and active forms of the same assertion result in the same PAS. The ESG parse does show the active-form (logical) arguments of a passive verb, except that a logical subject is a "by"-PP. PAS building replaces such a PP by the object of the preposition and removes any auxiliary "be" verb, in order to produce exactly the active form. A relation detector can use a pattern over the PAS that was developed (either manually or statistically) for active sentences and apply them to passive sentences without any changes; in contrast, if the pattern were written exactly over the parse, then two different variants would be needed to handle active and passive sentences (as illustrated in the example in the next section).

The PAS builder does not process the original text; it merely transforms the outputs of ESG into a simpler less comprehensive form. The PAS for a segment is a labeled directed graph. Each node has a citation form, begin and end character offsets, a POS, and a link to the ESG parse node from which it was derived. In addition, PAS nodes that are derived from ESG parse nodes that have determiners have an additional string label indicating the determiner that was used (since determiners generally do not appear as distinct nodes in the PAS). Each arc in the PAS, from node A to node B, is labeled by the SG slot that B fills in A.

We expect some meaningful distinctions to be lost when moving from ESG outputs to PAS. However, the lost meaning tends to be subtle (e.g., differences in emphasis) and difficult to make use of effectively. Omitting these distinctions can make it easier to write or automatically induce patterns for extracting semantics from questions and passages, as well as to detect when passages provide answers to questions. For example, consider the following sentences.

- I heard that Edison invented the phonograph in 1877.
- I heard that Edison invented a phonograph in 1877.
- I heard Edison invented the phonograph in 1877.
- I heard that Edison was inventing the phonograph in 1877.
- I heard that the phonograph was invented by Edison in 1877.

These sentences do have different meanings, and these differences are reflected in their ESG parses. The difference between the first two is particularly important, semantically, but it is very difficult to make use of that distinction in a general and flexible way. The PAS outputs for these sentences all have identical structure. Any relation detection rule that is written over PAS that matches any one of these sentences also matches all of the others. Any question that aligns well in the PAS with some or all of any of these sentences (e.g., "Who invented the phonograph?") does so equally well with any of the others. This is mostly an advantage of PAS, although it can also be a disadvantage when it causes a system to draw incorrect conclusions (e.g., concluding that someone invented the phonograph when that person merely invented a more advanced phonograph).

Here is an additional set of text snippets that are treated as identical by PAS.

- Odysseus was bold and clever.
- bold and clever Odysseus.

- bold and clever Odysseus was.
- bold, clever Odysseus.
- bold, clever Odysseus was.

ESG has two different dimensions of structure: deep structure and surface structure. PAS collapses these two dimensions into one. The structure of PAS generally follows the deep structure, except for those types of slots that are encoded only in the surface structure. Moreover, PAS nodes have a lemma form feature that roughly corresponds to the lemma in the ESG parse, but ESG has some special encoding of derivational morphology under some circumstances (e.g., "non-Mormon" in text is encoded as "non+Mormon" in the ESG lemma). PAS removes this special encoding.

A variety of nodes from ESG are omitted from the PAS. Links to and from these nodes in the parse tree are instead redirected to related nodes in the PAS. For example, links to/from auxiliary verbs are instead treated as links to/from the corresponding main verbs. Any links between the omitted and the related node are completely omitted. For example, the link from an auxiliary verb to the corresponding main verb is omitted in the PAS. The specific types of nodes omitted are as follows:

- Auxiliary verbs (including auxiliary verbs indicating passive voice).
- All closed-class nodes that introduce VPs, such as the infinitive "to" marker, and the "that" that introduces "that" clauses.
- Determiners, except for ones on a special whitelist of "high-semantics" determiners, e.g., possessive pronouns, determiners indicating negation (but, as noted above, the string values of the determiners are retained as string values in the PAS nodes that they would be attached to).
- Forms of "be" with no predicate.
- Forms of "be" for which the predicate is an adjective.

In addition, the PAS provides a simplified POS taxonomy: Several ESG POSs are completely omitted, as noted above; ESG's `subconj` (subordinating conjunction) is marked as `prep` (preposition) in PAS. ESG's `qual` (qualifier) is marked as `adv` (adverb) in PAS. The simpler taxonomy is sufficient to make key coarse distinctions, allowing downstream processing to distinguish between nouns, verbs, and some kinds of modifying and connecting words, without making all of the more subtle distinctions among word classes that ESG provides. In ESG, coordinating conjunctions (e.g., "and") are marked with the POS of the elements they coordinate; they have a separate `cord` feature to indicate that they are conjunctions. In PAS, the POS for these nodes is `cord`; the POS of the constituents is marked only on those constituents. This change is convenient for many components that perform local processing on specific nodes; for example, a component that wants to iterate through all nouns in a sentence (and does not want to treat conjunctions as nouns) can simply check the POS of each node, without having to do a separate check to see whether the node is a coordination. Below is a summary of the PAS for the chandelier question, whose ESG parse is shown in Figure 1:

```
chandelier (1)
look (2, subj:1, comp:3)
great (3)
but (4, lconj:2, rconj:9) [top predicate]
nowadays (5)
not (7)
usually (8)
use (9, subj:1, obj:11, vadv:5, vadv:7,
  vadv:8)
item (11, nrel:17) [determiner: these]
from (12, objprep:13)
which (13)
their (14)
name (15, ndet:14) [determiner: their]
derive (17, obj:15, comp:12)
```

The notation used here provides an ID number for each node (the first number after the open parenthesis) followed by zero or more labeled arguments consisting of a label, a colon, and the ID number of the target node. For example, "chandelier" has ID 1, and "use" has an ID 9, and its first labeled argument indicates that its subject has ID 1, i.e., "chandelier" is the subject of "use." The ID numbers used here are merely an artifact of how we present PAS in a paper. In DeepQA, PAS is encoded as UIMA CAS data structures with labeled pointers among nodes; hence, no ID numbers are needed. Key differences between the PAS in this example and the corresponding ESG parse in Figure 1 include the following.

- The ESG parse in Figure 1 shows the surface structure via lines and the deep structure as logical arguments. Since the PAS does not make that distinction, all structures are shown as labeled logical arguments. In the example above, the node "derive (17...)" in the PAS has two arguments that were derived from the deep (logical) structure of the corresponding node in the ESG parse. In contrast, the node "but (4...)" has two arguments in the PAS (labeled "lconj" and "rconj") that are derived from the ESG surface structure.
- Some nodes that are present in the parse are omitted in the PAS; specifically, the following nodes in the parse are omitted: the determiner "these" and the two auxiliary verb nodes (for "do" and "be"). The determiner "their", which is a possessive pronoun, is not dropped because PAS builder considers it a high-semantics determiner.

- As noted earlier, the ESG parse argument frame includes entries labeled `u` (for "unfilled" or "unknown") if there is no filler, e.g., `derive(17,u,15,12)`, indicating that "derive" has an unfilled slot for a subject. The PAS does not explicitly encode this information.
- The nouns that have determiners attached to them in the parse have a determiner string label in the PAS.
- The ESG parse has a set of features, shown in the rightmost column in Figure 1. These are not encoded in the PAS (we expect PAS users who want these features to examine the corresponding node in the ESG parse).

Pattern-based relation extraction

In this section, we show how the analysis provided by deep parsing facilitates pattern-based relation extraction. Relation extraction identifies domain-specific semantic-level relations in a sentence, together with their arguments, which are typically typed entities. Examples of semantic relations are `authorOf`, `actorIn`, and `bornOn`, and we refer to them as *deep* relations because they abstract even further away from ESG and PAS analyses. Such abstractions are used by processes such as *answer lookup* [2], *passage scoring* [3], and *structured inference* [14].

There is great variability, both lexical and syntactic, with which a relation can be expressed. For instance, just a sample of Jeopardy! questions with instances of an `authorOf` relation are listed below. Clearly, anticipating all of them and writing individual patterns is not a productive way to recognize a relation. Using representational devices from ESG and PAS analyses, however, it turns out that most of the examples can be handled by a single rule; only two rules are required for all of the examples.

(1) In 1936, he wrote his last play, "The Boy David"; an actress played the title role.
(2) Born in Winsted, he practiced law in Connecticut before he wrote "Unsafe at Any Speed".
(3) This "French Connection" actor coauthored the 1999 novel "Wake of the Perdido Star".
(4) Walter Mosley penned this mystery about Detective Easy Rawlins searching for a woman in post-WWII L.A.
(5) In December 1513, he wrote Francesco Vettori that he'd "composed a little work 'on princedoms'".
(6) A "manly" 19th century realist, she penned works like "Adam Bede", "Felix Holt" and "Daniel Deronda".
(7) Robert Louis Stevenson fell in love with Fanny Osbourne, a married woman, and later wrote this tale for her son.
(8) This friend who refused to destroy Kafka's works wrote a historical novel on Tycho Brahe.
(9) While living in Vermont, Kipling began writing this tale of an orphaned son of an Irish soldier in India.
(10) "According to" our sources, the first game book he wrote was a "Short Treatise on the Game of Whist".
(11) "The Lair of the White Worm" and "Dracula" were written by this Dubliner.
(12) "Somnium", an early work of science fiction, was written by this German & published posthumously in 1634.
(13) Rip's tale appears in "The Sketch Book" written by this man.
(14) Originally written by Alexander Pushkin as a poem, this Russian novel was later turned into an opera.
(15) This author of "Jazz" and "Tar Baby" was the first black American to win the Nobel Prize in Literature.
(16) Author of "Fathers and Sons", he was the first Russian to be widely read and admired in Europe.
(17) Beckett is a major dramatist of the "Theater of" this, which portrays a bewildered and anxious humanity.
(18) Edmund White wrote a definitive biography of this French thief, novelist & playwright of "The Maids".

Examples (1) and (2) illustrate a relatively straightforward `authorOf` expression: the verb "write" is immediately preceded by a subject (potentially an author) and followed by an object (referring to a title). Allowing for lexical variability (as we illustrate below), this observation suggests a pattern such as the following:

```
authorOf :: [ Author] [ WriteVerb] [ Work]
```

The issue is how to recognize the components of the pattern in the text. Linear sequence-based frameworks have problems identifying the headwords of constituents, which may be complex phrases themselves. This is orthogonal to the framework's need to be sensitive to alternative lexical cues for authorship, as examples (3) through (6) show (cf. "coauthor", "pen", "compose", "novel", "mystery", "bestseller", etc., as indicators for `authorOf`). An additional requirement here is that it manages to identify the specific elements to trigger a pattern match: e.g., "penned" (6) has three distinct `[Work]` arguments, signaled by coordination, and "Robert Louis Stevenson" is an `[Author]` argument to "wrote" (7), but this is difficult to detect, given the detracting effects of the intervening material.

Essentially, the problem with linear patterns is that all words "look" alike. In contrast, ESG and PAS provide the analysis backbone against which variability in textual expression can be reduced to a manageable number of patterns—because these now can target deep constituents and not surface word tokens. In most of the examples above, the syntactic analysis of the question identifies exactly the elements (typically `subj` and `obj`) that need to be examined for expressing a possible (`authorOf`) relationship.

There are numerous aspects of ESG's syntactic parse that play into structured rendering of the above pattern. For instance, consider the following: a) Multiword named entities are recognized as single nodes, making for a natural targeted match [cf. "Robert Louis Stevenson" (7), "Walter Mosley" (4)]. b) Complex NPs around relational nouns get internal structure assignment, which helps semantic interpretation [cf. "novel 'Wake of the Perdido Star'", where "novel" is the head noun, and material after it can be easily associated with the title (3)]. c) Quoted material is properly handled, within its larger context: It gets its own structured analysis suitably integrated within the overall parse—which allows the pattern to apply "across" the quotation marks in ⟨ he'd "composed a little work 'on princedoms'"⟩ (5). d) The analysis of coordinated constructions allows the matching framework to develop uniform analysis of nodes in relation instances; thus, rules need not separately cater for single, or conjoined, relation arguments. The same rule (above) would emit three distinct `[Work]` arguments to "penned" (6), or supply the right `[Author]` argument to "wrote" in (7), through its `cord` association with "fell in love", and so on.

ESG goes even further in systematically identifying and explicitly labeling long-distance relationships between constituents, thus bringing even more diverse text forms closer together. For instance, an arbitrary amount of text may interpose between a head noun and its covering verb; we already saw the need to connect "Robert Louis Stevenson" with "wrote" in (7); example (8) similarly requires identifying "This friend" as a potential `[Author]` argument, some distance apart from `[WriteVerb]`. In general, ESG attempts to share arguments—where they are semantically identical—across multiple clause boundaries, and even if the syntactic contexts differ in a variety of ways: "Kipling" is the subject of both "began" and "writing" (9); "book" is the object of both "wrote" and "was" (10).

Clearly, the value of the pattern outlined earlier is in its interpretation as a structural—as opposed to linear—template. Assuming a notation where

```
 [NodeA] -> dependencyLabel -> [NodeB]
```

refers to a dependency link between two (parse) nodes, an elaboration of the earlier pattern would target dependency trees. A single pattern, conjoining two labeled links anchored at the same (verb) node, would match all ten examples so far:

```
authorOf :: [WriteVerb] -> subj -> [Author] &
            [WriteVerb] -> obj  -> [Work]
```

We have chosen to define relation detection patterns over PAS representations, since the PAS builder both preserves information about argument clusters around nodes and introduces further normalizations of its own. An example of such additional normalization is the identical representation (in PAS) of active and passive forms of phrases: Because ESG retains the "by" marking, the above rule would need to be duplicated, if targeting the dependency tree. In contrast, if applied to PAS, the same rule (unmodified) would detect a relation in a variety of passive constructs: simple, or within, e.g., conjunctions (11) or appositives (12), relative clauses (13), finite VP premodifiers and postmodifiers (14), and so forth.

Additional simplifications of PAS over ESG, contributing to the clarity of rule writing and keeping down the number of rules, include marking coordinated constructions with a unique `cord` POS, suppressing distinctions between modifiers to the left or right of the head, and packaging of modifiers and their labels into synchronous lists. (See the section "Predicate-argument structure" above.)

As an illustration of the last point above, consider expanding our current rule base (of one pattern) to detect instances of the `authorOf` relation expressed by means of relational nouns [examples (15) through (18)].

The key, an "of"-PP attached to an `[Author]` predicate, may be syntactically analyzed as an `nobj` or `nprep` construction, potentially leading to two separate rules over ESG (and consequently PAS) dependency labels. However, the PAS convention for uniform access to a node's modifiers (where each modifier is paired with the label with which it is attached to the head) allows for a single rule, loosely stating a conjunction of two dependency links sharing an "of" node:

```
authorOf :: [Author] -> Label -> ["of"-Arg] &
                ["of"-Arg] -> objprep -> [Work]
```

`Label` acts as a wildcard over `nobj` or `nprep`. The rule would identify an argument in the list of argument nodes of `[Author]`, which is governed by an "of" preposition, linking it with an NP whose head passes the `[Work]` constraint.

This discussion shows how the conventions of the analytic style exposed by ESG and PAS facilitate the abstraction of a relatively small set of rules, which—by targeting constituent arguments and dependency link labels—are capable of handling some of the variability of relation expression in text.

Use in Watson

As noted in the introduction to this paper, the ESG parse, PAS, and semantic relations are used by many different components in Watson. Consider the example clue from earlier in the text: "Chandeliers look great but nowadays do not usually use these items from which their name is derived." This section describes a few of the many ways in which the parse and information derived from the parse are used.

Question analysis [1] attempts to identify a question focus: the term in the clue that one would substitute the correct answer for to produce a true statement. The analysis applies a variety of rules to the parse to make this assessment. In this example clue, the focus is "these items", which is inferred from the fact that it has the word "these" as a determiner. Question analysis also attempts to identify one or more LATs, often the head of the focus or some word closely connected to the focus in the parse. In the example clue, the LAT, "items", is the head of the focus, but more complex clues may have LATs that are disjoint from the focus.

Some passage-scoring components [3] use the results derived from the ESG parse (specifically, the PAS and the semantic relations) from both the question text and the text of various passages believed to contain supporting evidence for some candidate answer. These components use that information to determine the strength of the evidence from that passage. A part of this assessment involves determining how well the candidate answer in the passage aligns with the focus of the question. In our example, the PAS shows that the focus, "these items", is the object of the verb "use", which has subject "chandeliers" and modifier "not". If a passage contains the verb "use" with the object being a candidate answer, the subject being "chandeliers", and a modifier "not", then these passage scorers will assign a very high score to this candidate answer. If the passage has similar words (e.g., synonyms) or similar structure, then the passage scorers will provide partial credit. Different passage scorers have different restrictions on how well the terms and the connections among terms (e.g., from PAS or semantic relations) must match to receive different levels of credit; see [3] for more details.

The PRISMATIC knowledge base [11] includes statistics relating to how often sets of words are connected to each other via specific PAS structures and semantic relations. For example, PRISMATIC has a record of how often the word "chandelier" is the subject of the verb "use" in a large corpus and also of how often any given word is the object of the verb "use" with subject "chandelier". This data is used for finding [2] and evaluating [3, 10] candidate answers.

Some components [10] apply ESG and the PAS annotator to analyze the content of various sources of answer type information. One of the many sources used for typing information is Wikipedia categories, which are short text strings describing the type of entity that the page is about. For example, the Wikipedia page for "Toronto, Ohio" has as one of its categories "Populated places in Jefferson County, Ohio". Given this label, we would want to conclude that "Toronto, Ohio" could be a legitimate answer to a question asking for a "place," but we would not want to conclude that "Toronto, Ohio" could be a legitimate answer to a question asking for a "county" (although both of these words appear in the category text). The ESG parse shows "places" as the syntactic head of the NP "Populated places in Jefferson County, Ohio." Consequently, our type coercion algorithms can conclude that this category is asserting that Toronto, Ohio, is a place but would not conclude that Toronto, Ohio, is a county.

As noted earlier, this is just a small sample of how Watson uses the ESG parse and information derived from it. Information about how words are connected to each other (both in the question and in a variety of evidence sources) is vital to the operation of the DeepQA technology and is thus pervasively used in Watson.

Table 1 Parser comparison.

Parser	*Test set A (Jeopardy!)*	*Test set B (Wikipedia)*
ESG	92.0%	88.7%
ESGbase	84.2%	84.4%
Charniak parser	83.6%	81.1%

Evaluation

ESG was evaluated for parse correctness on two test sets: (A) 100 Jeopardy! question segments and (B) 100 segments from Wikipedia. Both test sets were randomly chosen, and ESG had no training on them. [In the ESG tests, the NE (named entity) chunk lexicon extracted from Wikipedia, mentioned above, was not used.] We have also evaluated an earlier version of ESG (ESGbase) on these test sets. ESGbase is a version that dates from just prior to the commencement of work on Watson.

The Charniak parser (CP; reranking version with Mark Johnson) [15] was evaluated on the same two test sets. Although the CP delivers a different kind of parse tree [in Penn Treebank (PTB) style], these trees can be viewed as dependency trees in a straightforward way—for example, replacing `(S (NP ...)(VP v ...))` by `(VP (NP ...) v ...)`.

A node N in a dependency parse tree is called *correct* if the mother node exhibited for N is correct, and the POS of N is correct. POSs are taken on the level of PTB POSs. The score for a parser is the percentage of correct nodes it delivers across the whole test set (using only the automatically top-ranked parse for each segment). (Recall is 100%; both parsers deliver node data for all nodes in the test sets.) The scores are given in **Table 1**.

In the evaluation, more was required of CP than sometimes shows in the PTB. CP is weak on showing coordination structure, particularly in the part of an NP from the beginning up through the head noun (call this the "NP-front"). The NP-front is usually (not always) shown in a flat way by CP. However, coordination is part of surface structure. ESG shows it and is penalized if it gets any part wrong. Thus, flat CP NP-fronts with coordination are penalized—except in the situation (which often holds) that there is only one possible choice for the conjuncts. CP NP-front analysis is also weak in that it sometimes includes what should be a postmodifier of the NP-front at the end of the flat NP-front list itself, as in "The brothers Grimm". Then it is difficult to tell what the head noun should be. CP is also weak, and inconsistent, in showing whether capitalized nouns are proper or common nouns, and it sometimes shows them as adjectives. It also does not show structure for productive hyphenation, as in "group-oriented".

Note that node correctness is a matter only of surface structure; hence, the scoring in Table 1 does not reward ESG for the extra information it provides in deep structure, slot-filler marking, rich semantic type marking, etc. This additional information is useful for Watson and other applications.

We also scored on test set A the latest version of ESG without the WordNet-based augmentation described above, and the result was 91%. This is not outstandingly lower than the 92% with the WordNet augmentation, but it does not measure the value of the considerably greater semantic type marking of the latter, which is useful for downstream components and for further improvements in parsing accuracy.

ESG parsing is efficient, processing ~5,000 words per second on standard laptops; this is particularly valuable for the Jeopardy! application. The efficiency is owed to a careful implementation in C, to care in writing the lexicon and the syntax rules, and to the parse space pruning algorithm. In a test on 6,712 Jeopardy! question segments, ESG ran in a total of 17 seconds. On the same test set and the same machine (a Linux** xSeries* machine), CP ran in 1,763 seconds using one thread, or ~104 times slower than ESG. CP can run in more than one thread. With two threads, it ran ~57 times slower than ESG. The machine has four cores, but the three- and four-threaded runs of CP took longer than the one-threaded CP run. ESG is not equipped to run in more than one thread per sentence parse, but for processing a large corpus, it would be easy to divide the corpus into pieces and run the different pieces on different cores.

ESG has a small footprint. The binary code and data (not counting chunk lexicons) occupy ~5.7 MB, and the working memory normally used is ~52 MB. The same version of ESG used in Watson easily compiles and runs on a smartphone.

We have not formally evaluated the PAS builder. Given that the PAS builder provides only a simplification of the ESG parse, we believe that the correctness results for ESG above provide an adequate sense of how accurate the PAS is. The exact percentage of correct nodes would be slightly different since the PAS has fewer nodes (given that some nodes are dropped). However, in general, the PAS is correct whenever and to the extent that the ESG parse is correct; hence, we believe that conducting a distinct formal evaluation for the PAS would not be productive.

Relation extraction was formally evaluated in [9].

Related work

ESG is rule-based, linguistically oriented, head-driven, frame-oriented, and largely lexicalist. It is probably most closely related to Head-driven Phrase Structure Grammar (HPSG) [16] and Lexical Functional Grammar (LFG) [17] among commonly used computational grammatical systems. SG was one of the first computational head-driven lexicalist grammatical systems, with the first work done around 1978 and published in 1980 [4]. The combination in SG of surface structure and deep structure is analogous to LFG's constituent structure and functional structure, but a difference is that, in SG parse trees, surface and deep structure are combined in the same tree. It is natural to do this in SG because the functions (slots) play a crucial role in parsing itself. It is also useful to have the two aspects of analysis combined because some elements of surface structure can play a role in logical forms, as happens, e.g., with generalized quantifiers and focusing adverbs. PAS usefully continues the combination.

Among commonly used statistical parsers, such as the Charniak parser [15], Collins parser [18], Stanford parser [19, 20], and Minipar [21], SG is most closely related to the dependency parser Minipar. However, we have chosen to compare ESG with the Charniak parser because of the latter's relatively wide use.

The development regimen for ESG has been regular testing, observation of problems, making fixes that are as general as possible, introduction of new constructions when appropriate, and constant regression testing on a set of ~20,000 segments from various sources. (This set does not overlap with the test sets A or B of the "Evaluation" section.) There is no use of a tree bank. Imitating a tree bank can sometimes stifle introduction of useful new constructions that do not exist in the tree bank.

The term "predicate-argument structure" has been used to refer to a wide variety of parse-like constructs, usually focusing on semantic aspects of analysis rather than on syntactic ones. For example, the PAS used in VerbNet [22] uses thematic roles such as *Agent* and *Patient*. To assign thematic roles, a system would then need some sort of lexical resource describing the roles that apply to specific words in specific syntactic constructs (e.g., [22]), and/or some labeled training data illustrating the mapping by example (e.g., [23]). Other PAS instantiations simply use syntactic labels directly and thus do not need to perform this mapping—e.g., [24] and the PAS described in this paper. (However, it should be noted that our PAS includes ESG complement slot-fillers as arguments, which have a semisemantic nature and often correspond to thematic role arguments.) PAS instantiations of this sort do not necessarily require extensive lexical knowledge and can be much simpler to implement, although obviously they provide less extensive semantic insight. They still provide a more semantics-oriented level of analysis than a parse usually does, because they remove parse ingredients that are important only syntactically. The DeepQA project does include a component that determines the thematic roles that relate terms in some text, but we consider this to be a distinct level of analysis, specifically, part of "shallow" semantic relation detection [3]. By separating PAS and identification of thematic roles into distinct components, we provide separate complementary levels of analysis that downstream components can use separately or in conjunction.

Conclusion and future work

ESG can produce informative parse trees that exhibit both deep structure and surface structure in a unified way. ESG does so with broad coverage and good efficiency. The ESG parse (particularly the deep structure) feeds into the simplified and abstracted analysis of PAS. Results from ESG and the PAS builder are used in many downstream components of Watson. For example, pattern-based relation recognition uses the syntactic analysis that is generated by ESG and simplified/abstracted by the PAS builder, and we have provided an illustration of that.

We have given an overview of ESG and PAS analysis and indicated the central role of slots and slot frames. We have emphasized the new ingredients developed during the work on Watson, some of which are special to the Jeopardy! task, but most of which are of general value.

Future work on SG will include the following: 1) expansion of the semantic type system and its use in parsing; 2) incorporation of word-sense disambiguation, probably with senses of less granularity than in WordNet; 3) indication in parse trees of scoping of generalized quantifiers and focusing adverbs, etc.; 4) development of specialized lexicons and methods for handling very large lexicons; and 5) continued improvement of coverage of SG via regression testing.

References

1. A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll, "Question analysis: How Watson reads a clue," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 2, pp. 2:1–2:14, May/Jul. 2012.
2. J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty, "Finding needles in the haystack: Search and candidate generation," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 6, pp. 6:1–6:14, May/Jul. 2012.
3. J. W. Murdock, J. Fan, A. Lally, H. Shima, and B. K. Boguraev, "Textual evidence gathering and analysis," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 8, pp. 8:1–8:14, May/Jul. 2012.
4. M. C. McCord, "Slot grammars," *Comput. Linguist.*, vol. 6, no. 1, pp. 31–43, 1980.
5. M. C. McCord, "Using slots and modifiers in logic grammars for natural language," *Artif. Intell.*, vol. 18, no. 3, pp. 327–367, May 1982.
6. M. C. McCord, "Heuristics for broad-coverage natural language parsing," in *Proc. ARPA Hum. Lang. Technol. Workshop*, 1993, pp. 127–132.
7. M. C. McCord, "Using slot grammar," IBM T. J. Watson Res. Center, Yorktown Heights, NY, IBM Res. Rep. RC23978. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/papers/FB5445D25B7E3932852576F10047E1C2/$File/rc23978revised.pdf
8. D. A. Ferrucci, "Introduction to 'This is Watson,'" *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 1, pp. 1:1–1:15, May/Jul. 2012.
9. C. Wang, A. Kalyanpur, J. Fan, B. K. Boguraev, and D. C. Gondek, "Relation extraction and scoring in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 9, pp. 9:1–9:12, May/Jul. 2012.
10. J. W. Murdock, A. Kalyanpur, C. Welty, J. Fan, D. A. Ferrucci, D. C. Gondek, L. Zhang, and H. Kanayama, "Typing candidate answers using type coercion," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 7, pp. 7:1–7:13, May/Jul. 2012.
11. J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci, "Automatic knowledge extraction from documents," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 5, pp. 5:1–5:10, May/Jul. 2012.
12. G. A. Miller, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
13. C. Fellbaum, *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press, 1998.
14. A. Kalyanpur, B. K. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. M. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. M. Qiu, "Structured data and inference in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 10, pp. 10:1–10:14, May/Jul. 2012.
15. E. Charniak and M. Johnson, "Coarse-to-fine n-best parsing and MaxEnt discriminative reranking," in *Proc. 43rd Annu. Meeting Assoc. Comput. Linguist.*, 2005, pp. 173–180.
16. C. Pollard and I. A. Sag, *Head-Driven Phrase Structure Grammar*. Chicago, IL: Univ. Chicago Press, 1994.
17. R. M. Kaplan and J. Bresnan, "Lexical-Functional Grammar: A formal system for grammatical representation," in *The Mental Representation of Grammatical Relations*, J. Bresnan, Ed. Cambridge, MA: MIT Press, 1982, pp. 173–281.
18. M. Collins, "Head-driven statistical models for natural language parsing," *Comput. Linguist.*, vol. 29, no. 4, pp. 589–637, 2003.
19. D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proc. 41st Meeting Assoc. Comput. Linguist.*, 2003, pp. 423–430.
20. M.-C. de Marneffe, B. MacCartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," in *Proc. LREC*, 2006, vol. 6, pp. 449–454.
21. D. Lin, "Dependency based evaluation of MINIPAR," in *Proc. 1st Int. Conf. Lang. Resources Eval.—Workshop on the Evaluation of Parsing Systems*, Granada, Spain, 1998.
22. K. K. Schuler, "VerbNet: A broad-coverage, comprehensive verb lexicon," Ph.D. dissertation, Univ. Pennsylvania, Philadelphia, PA, Jan. 2005, Paper AAI3179808.

23. K. Kipper, M. Palmer, and O. Rambow, “Extending PropBank with VerbNet semantic predicates,” in *Proc. AMTA Workshop Appl. Interlinguas*, Tiburon, CA, Oct. 2002. [Online]. Available: http://www.mendeley.com/research/extending-propbank-verbnet-semantic-predicates/
24. R. Krestel, R. Witte, and S. Bergler, “Predicate-argument EXtractor (PAX),” in *Proc. LREC—New Challenges for NLP Frameworks*, Valletta, Malta, 2010, pp. 51–54.

Received July 27, 2011; accepted for publication November 17, 2011

Michael C. McCord *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (mcmccord@us.ibm.com)*. Dr. McCord is a Research Staff Member in the Semantic Analysis and Integration Department at the T. J. Watson Research Center. He received the Ph.D. degree in mathematics from Yale University, New Haven, CT, and spent a year at the Institute for Advanced Study, Princeton, NJ. His initial research was in mathematics (topology and foundations), and he then moved into computer science and natural-language processing, with emphasis on syntax and semantics. He has been at IBM Research since 1983. He originated the Slot Grammar parsing system, which has been applied to machine translation and grammar checking, and which is used in the Watson question-answering system. He is author or coauthor of 47 refereed articles and one book.

J. William Murdock *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (murdockj@us.ibm.com)*. Dr. Murdock is a member of the IBM DeepQA Research Team in the T. J. Watson Research Center. In 2001, he received the Ph.D. degree in computer science from Georgia Tech, Atlanta, where he was a member of Ashok Goel’s Design and Intelligence Laboratory. He worked as a Post-Doctorate with David Aha at the U.S. Naval Research Laboratory, Washington, DC. His research interests include natural-language semantics, analogical reasoning, knowledge-based planning, machine learning, and self-aware artificial intelligence.

Branimir K. Boguraev *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (bran@us.ibm.com)*. Dr. Boguraev is a Research Staff Member in the Semantic Analysis and Integration Department at the Thomas J. Watson Research Center. He received the Engineering degree in electronics from the Higher Institute for Mechanical and Electrical Engineering in Sofia, Bulgaria (1974) and the Diploma and Ph.D. degrees in computer science (1976) and computational linguistics (1980), respectively, from the University of Cambridge, Cambridge, U.K. He worked on a number of U.K./E.U. research projects on infrastructural support for natural-language processing applications, before joining IBM Research in 1988 to work on resource-rich text analysis. From 1993 to 1997, he managed the natural-language program at Apple’s Advanced Technologies Group, returning to IBM in 1998 to work on language engineering for large-scale, business content analysis. Most recently, he has worked, together with the Jeopardy! Challenge Algorithms Team, on developing technologies for advanced question answering. Dr. Boguraev is author or coauthor of more than 120 technical papers and 15 patents. Until recently, he was the Executive Editor of the Cambridge University Press book series *Studies in Natural Language Processing*. He has also been a member of the editorial boards of *Computational Linguistics* and the *Journal of Semantics*, and he continues to serve as one of the founding editors of *Journal of Natural Language Engineering*. He is a member of the Association for Computational Linguistics.