RC23977 (W0607-020) July 10, 2006
Computer Science

IBM Research Report

The Slot Grammar Lexical Formalism

Michael C. McCord
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

The Slot Grammar Lexical Formalism

Michael C. McCord
IBM T. J. Watson Research Center

Abstract

The purpose of this report is to describe a formalism, **SLF**, used for building Slot Grammar (SG) lexicons. SLF provides a rich descriptive language for lexical entries, with a system of defaults that also allows very simple entries in the cases where less detail is called for. Word sense specifications in entries can include any of the following ingredients, where only the first one is obligatory: (a) Part of speech, (b) (complement) slot frame, (c) features (both morphosyntactic and semantic), (d) subject area tests, (e) sense rewards or penalties (feeding into parse scoring), and (f) sense name. Lexical entries can also specify support verb constructions (or analogs for other parts of speech), and can provide idiosyncratic inflectional information. Index words of entries can be single words or multiwords. The slot frames of SLF lexical entries are the most distinctive and most powerful ingredients. Slots have a dual role – indicating logical arguments of word sense predicates, and acting as grammatical relations. A great deal of SG analysis is under the control of lexical slot frames. A slot specification in SLF includes the slot's name plus optionally a list of *options* offering a disjunctive choice of ways the slot can be filled. Each slot option specifies tests on the filler, and there is an expressive sublanguage of SLF for describing these tests.

1 Introduction

In this report we describe the formalism used for Slot Grammar lexicons, which we call **SLF** (**S**lot Grammar **L**exical **F**ormalism). Slot Grammar (SG) is dependency-oriented, and a great deal of the analysis is under the control of complement slot frames associated in the lexicon with senses of the head words of phrases. Slots (the members of slot frames) have a dual role. They indicate logical arguments of word sense predicates, and they act as grammatical relations. Examples of slots are `subj` (subject), `obj` (direct object) and `iobj` (indirect object). Parsing consists of filling slots. The main rules in the syntactic component of a Slot Grammar are slot-filling rules – where the slots are either complement slots coming from the lexicon, or adjunct slots decided upon by the syntactic component itself.

This report has two companion reports, [12] and [13]. The first of these, "A Formal System for Slot Grammar", describes a formalism (SGF) for writing the syntax rules for a Slot Grammar. The second report, “Using Slot Grammar”, contains a general description of SG and of an API for it, with an emphasis on how to use SG parsers in applications. Those two reports and the current one form a kind of triad which should give a good picture of the current state of Slot Grammar, especially the syntax rules and the lexicon. The three reports complement one another, and one can get the full picture best by reading all three. Nevertheless, each of the reports is written in a fairly self-contained way.

SLF provides a rich descriptive language for lexical entries, which can be quite detailed or quite simple, depending on the sophistication and the mix of syntax and semantics that one wants. The notion of word *sense* can vary, depending on how much detail one puts into semantic constraints. In typical SG lexicons, the word “senses” are differentiated on the top level by parts of speech, and within each POS (part of speech) one may see several sense frames with differentiating slot frames. Sometimes the different slot frames for a given word and a given POS are enough to differentiate real semantic word senses, but quite often they do only a partial job of this. If one wants to make a deeper differentiation, one can accomplish this through use of semantic types in lexical entries, and SLF allows this. Semantic types (features) and grammatical features can be marked on word sense frames (asserted of them), and tests for such features on slot fillers can be specified in a flexible way (with boolean combinations of elementary tests) within slots. However, typical existing SGs take the “middle route” of sense differentiation only on the level of predicate argument structure and do not use too many semantic type tests. Then real word sense disambiguation might be done in a postprocessing way (after parsing), as in [11].

SLF lexical entries can be created for either single words or multiwords. Multiwords can be inflected by inflecting head word components indicated for them.

Most lexical entries deal with the specification of sense frames, but idiosyncratic inflectional information may also be given.

Lexical entries can test for active document-level subject areas in deciding on which sense frames to allow.

Slot Grammar uses numerical parse scoring to arrive at most likely parses. Lexical entries can have components that feed into this scoring – indicating for example that some sense frames should be rewarded or penalized. These rewards or penalties may appear within semantic type tests or subject area tests, making these tests “soft” and not absolute requirements.

There is a system for specifying support verb constructions (and analogs for any other part of speech).

The examples in this report will be given mainly for English, but the general description of SLF is applicable to the SG for any language.

SLF can be used both for building base lexicons for SG parsers and for building user (addendum) lexicons for SG applications.

The remainder of this report is organized as follows:

- Section 2, page 3: "Overall format"
- Section 3, page 4: "Index words"
- Section 4, page 5: "Sense frames"
- Section 5, page 6: "Parts of speech"
- Section 6, page 7: "Slots"
- Section 7, page 10: "Inventory of slots"
- Section 8, page 15: "Inventory of slot options"
- Section 9, page 20: "Option tests"
- Section 10, page 23: "Features"
- Section 11, page 32: "Subject area tests"
- Section 12, page 35: "Sense names"
- Section 13, page 36: "Inflectional elements"
- Section 14, page 37: "Support word frames"

2 Overall format

A lexical entry in SLF consists of an *index word* followed by one or more *elements*. Example:

```
access
   < v obj
   < n (p to)
```

Here the index word is `access`. There are two elements, `v obj` and `n (p to)`. The first one says that `access` can be a verb taking an (optional) direct object (`obj`). (The verb will also have a subject slot (`subj`) by default.) The second element says that `access` can be a noun taking an (optional) `to`-prepositional phrase complement.

With some exceptions, index words should be citation forms (lemmas) for words (single words or multiwords). The index word for an entry must start in column 1. The remainder of the entry can use whitespace (blanks, tabs, newlines) anywhere (except in the middle of atomic symbols), but must use a blank or tab in the first column of any continuation line. The elements of a lexical entry are each preceded by the symbol `<`.

A lexicon file can contain blank lines or comment lines anywhere. A comment line begins with `/*`, preceded possibly by whitespace. The comment extends only to the end of the line. The ending `*/` is not actually required, but it is good practice to use it. It is also good practice to precede comments within an entry by some blanks. It is not possible to start a comment on a line after some non-comment material.

The syntax of elements is Cambridge Polish (Lisp-like) except for the very top level of the element. We will call expressions in Cambridge Polish *terms*. The general form of a term can be described quite simply. A *term* is either an *atomic term* or a *list term*. *Atomic terms* are sequences of characters not including whitespace or (round) parentheses, except that a backslash `\` can be used as an escape character to allow those symbols also. A *list term* consists of a left parenthesis followed by possible whitespace, followed by any sequence of terms separated by whitespace, followed by possible whitespace, followed by a right parenthesis. (In SLF we do not use the dot notation sometimes used in Cambridge Polish.) The empty list can be denoted by either `()` or `nil`.

The very top level of an element is a sequence of terms, separated by whitespace. The SG interpreter for SLF actually makes this sequence into a single list term before working with it, by (in effect) surrounding it with parentheses. In the external syntax for SLF, there is no need for these because the symbols `<` in entries delimit their elements.

3 Index words

An index word can be a single word or a multiword. Multiwords are groups of two or more words, separated by blanks. As an example, the entry

```
data base < n
```

has a multiword index word `data base` (and there is one element that says it is a noun).

The *head (word)* of a multiword is the component that gets inflected. For instance, `base` is the head of `data base`, because the plural `data bases` is formed by pluralizing `base`. In specifying a multiword, you need to indicate its head word. The head can be indicated by putting an `=` sign right before it; but there is a convention that if no `=` sign appears in the multiword, its head will be taken to be the last word. This is the case with our example for `data base` above, but you could also write:

```
data =base < n
```

An example where the `=` sign is needed is:

```
=editor in chief < n
```

Here the plural would be `editors in chief`. In some cases, a multiword has a part of speech, like preposition, that doesn't get inflected. In such cases you can basically take the head word to be any component you want; but it is better to choose one of the rarer components. For example, it would be good with `ahead of` to take `ahead` as the head, like so:

```
=ahead of < prep
```

because `of` is a much more common word than `ahead`. The reason for taking a rarer word as head is that when the lexicon is stored internally for SG, each multiword is actually indexed under its head word. If `ahead of` were indexed under `of`, then there would be an attempted look-up every time the word `of` occurs in the sentence.

A multiword can contain the special symbol `num`, which stands for an arbitrary number. Example:

```
=less than num < adj
```

This allows formation of multiword adjectives like `less than three`, `less than 3`, and `less than two hundred and five`.

4 Sense frames

There are three kinds of possible elements in a lexical entry – *sense frames*, *inflectional elements*, and *support word frames*. We describe sense frames in this section. A sense frame basically specifies a sense of the index word – "sense" so far as is important for SG. All of the examples of elements so far have been sense frames.

A sense frame can have the following six kinds of members:

- Part of speech (POS)
- Slot
- Feature
- Subject area test
- Sense score
- Sense name

We will describe these in the following sections. The only obligatory member is the POS, and it must come first. The others can be intermixed and can come in any order, except that the relative order of the slots makes a slight difference. In makes little difference for basic parsing, but the parse display shows word senses with "arguments" that correspond to the slots in the sense frame used. It is generally good practice to use the following overall order for the members of a sense frame:

IndexWord < *POS Slots Features SubjectAreaTest Score SenseName*

An example that has all of the first four kinds of members is:

```
=Attorney General < propn (p of) h (sa gov)
```

In this sense frame for `Attorney General`, the POS is `propn` (proper noun); `(p of)` is a slot that is filled by an `of` prepositional phrase; `h` is a feature that stands for "human"; and `(sa gov)` is a subject area test that checks whether the subject area is `gov(ernment)`.

5 Parts of speech

There are actually seventeen different "parts of speech" that can be specified as the first member of a sense frame, as follows:

> `n, propn, pron, num, v, modal, adj, adv, qual, det, prep, subconj, conj, infto, subinf, forto, thatconj`

Some of these are actually subcategories of normal parts of speech; e.g. `propn` (proper noun) is a kind of noun. Some of them are very special, with only one or two words of that category. Let us now explain each of the kinds of part of speech.

`n` (common noun). This is used only for common nouns – nouns that name a class of objects.

`propn` (proper noun). Proper nouns usually name a specific object – at least in a given context. The most typical proper noun is a human name, like `Mary`. Proper nouns are normally capitalized, though not always.

`pron` (pronoun). These are words like `she` and `somebody` that act like "variables" in naming entities.

`num` (number). This POS is used for literal number words like `three`.

`v` (verb). This is used for all verbs except modal verbs (see next).

`modal` (modal verb). For SG the English modal verbs are `will`, `shall`, `can`, `may`, `must`, `dare`, and `need`. And they have past forms `would`, `should`, etc., which are specified by inflectional elements.

`adj` (adjective). Examples: `good`, `happy`, `clever`.

`adv` (adverb). Examples: `happily`, `probably`, `fast`.

`qual` (qualifier). Qualifiers are adverb-like words that can modify adverbs. Examples are `very`, `how`, `more`, `less`, `quite`.

`det` (determiner). Determiners are words like `the`, `a`, `this`, `that`, and `each` that introduce noun phrases.

`prep` (preposition). Examples: `in`, `on`, `with`.

`subconj` (subordinate conjunction). Examples: `if`, `since`, `after`.

`conj` (coordinating conjunction). Examples: `and`, `or`, `but`.

`infto` (infinitive to). There is only one example: `to`, when used to introduce infinitive verbs, as in *to go* and *to be or not to be*.

`subinf`. This includes multiwords like `in order to` and `so as to` that behave syntactically like `infto`.

`forto`. This is the POS for the sense of `for` in examples like *for the chairperson to be there would be impossible.*

`thatconj`. This is the POS for the sense of `that` in examples like *He said that she was there.*

Particles are certain prepositions and adverbs. The preposition particles are:

```
aboard, about, above, across, along, around,
behind, below, beneath, beside, between,
by, down, from behind, in, inside, near,
off, on, on board, out, outside, over,
past, round, through, under, underneath, up
```

The adverb particles are:

```
abreast, ahead, apart, aside, away, back,
backward, backwards, clockwise, counterclockwise,
downstairs, downward, downwards,
forth, forward, forwards, home,
inward, inwards, outward, outwards,
over backward, over backwards,
together, upstairs, upward, upwards
```

6 Slots

In the example

```
give < v obj iobj
```

the verb `give` is shown as having two slots, `obj` (direct object) and `iobj` (indirect object). There is also a subject slot (`subj`) for `give`, but we usually don't need to specify that in a verb sense frame, because the SG lexical preprocessor adds it to the slot frame by default. Example sentences using this sense frame for `give` are:

$$\textit{Alice gave } \underbrace{\textit{the book}}_{\texttt{obj}}\ \underbrace{\textit{to Bob}}_{\texttt{iobj}}.$$

$$\textit{Alice gave } \underbrace{\textit{Bob}}_{\texttt{iobj}}\ \underbrace{\textit{the book}}_{\texttt{obj}}.$$

We say that in the first sentence the slot `obj` is filled by the NP (noun phrase) *the book*, and the slot `iobj` is filled by the PP (prepositional phrase) *to Bob.* But in the second sentence, `iobj` is filled by the NP *Bob*, in a different position, and `obj` is filled by *the book* (also in a different position). Even though the category of the filler of `iobj` may vary (to-PP or NP), we view the filler phrase as filling the same slot, because the logical role of the filler is the same; *Bob* is the one to whom the book is given in both of the examples.

In general, slots correspond to logical *arguments* of the word senses with which they are associated. But they have a syntactic nature as well as a semantic one, because they can be filled only by phrases of certain categories (like NP or to-PP in the case of `iobj`), and these filler phrases may have certain constraints on their positions in the sentence. The constraints on position are handled in the grammar, on the basis of the particular slots involved.

A slot for a word sense w can be filled only once in a given sentence (for that occurrence of w.) For example, an occurrence of `give` can have only one direct object and only one indirect object. However, the same slot (name) might appear more than once in a complement slot frame. This happens e.g. with the slot `comp` for verbs. In such cases, each *occurrence* of the slot can be filled only once.

A slot always has certain *options* associated with it, either overtly or implicitly. In the following example:

```
buy < v obj (iobj n for)
```

the `iobj` slot is shown with options `n` and `for`. The `n` option means that the `iobj` can be filled by an NP (as in *buy me a book*), and the `for` option means that the `iobj` can be filled alternatively by a `for`-PP (as in *buy a book for me*). The `obj` slot in this example shows no option overtly, but the default is that `obj` has an `n` option if no option is specified. In other words, the preceding example could have been written as follows:

```
buy < v (obj n) (iobj n for)
```

In a similar way, the `iobj` slot has default options `n` and `to`. So we might spell out

```
give < v obj iobj
```

more fully as:

```
give < v (obj n) (iobj n to)
```

Below, we will describe the various available options for slots. But let us continue with more general facts about slots.

Strictly speaking, in a slot specification like `(obj n)`, `obj` is the slot name, `n` is an option, and the whole expression `(obj n)` specifies the slot. If the slot is specified only as `obj`, then we really consider it as shorthand for the complete expression `(obj n)`.

Generally, then, we may specify a slot by showing only its name (and then taking the default options), or we may write:

$(SlotName\ \ Option_1\ \ Option_2\ \ \cdots\ \ Option_n)$

The options are viewed as alternatives – as if they are connected by or's.

A slot may be *obligatory* (must receive a filler in a grammatical sentence) or *optional* (may or may not get a filler). The default is that the slot is optional, and obligatoriness is indicated by suffixing the digit `1` to the slot name. For example, `obj` is the optional form of the direct object slot (name) and `obj1` is the obligatory form. In some treatments of word senses, there is a distinction between *transitive* and *intransitive* verbs. For ESG, the single sense frame

```
eat < v obj
```

for the verb `eat` allows both the transitive and intransitive uses of `eat`, since `obj` is optional.

Slot options may have tests within them, using the syntax:

$(OptionName\ \ Test_1\ \ Test_2\ \ \cdots\ \ Test_n)$

The tests are interpreted disjunctively. We will describe the range of possible option tests in more detail in Section 8, but let us mention here the most important kind of test. This occurs when the test is specified by an atomic symbol w – which will normally be a word – and then the interpretation of the test is that the head word of the filler of the slot-option is that specific word w.

For example, the following verb sense frame for `do`

```
do < ... < v (iobj1 n) (obj1 (n favor))
```

would require that the direct object has explicitly the head word `favor`. This would then allow constructions like *do somebody a favor*. It also includes *do somebody a big favor*, etc., because the option test just says that the head word of the `obj` is `favor`.

The most common case of option tests that name specific words are for slot options that require PP fillers and name specific prepositions. For example, the sense frame:

```
absorb < v obj1 (comp (p in into))
```

would allow constructions like *absorb X in Y* and *absorb X into Y*. The `comp` slot has the option `(p in into)`. The `p` option for `comp` is one that allows the `comp` to be filled by a PP. And the option tests `in` and `into` allow the head of the PP to be `in` or `into`.

The slots that appear in sense frames are, strictly speaking, called complement slots. Slot Grammar also uses adjunct slots. But adjunct slots are not specified in lexical entries; they are known to the grammar. Each word sense will have certain adjunct slots associated with it, depending only on its POS. For example, the verb POS has an adjunct slot called `vadv` which is typically filled by adverb phrases or time NPs. Here is an example sentence with both complement and adjunct slot fillers in it:

$$\underbrace{\textit{Probably}}_{\texttt{vadv}}\ \underbrace{\textit{Alice}}_{\texttt{subj}}\ \textit{gave}\ \underbrace{\textit{Bob}}_{\texttt{iobj}}\ \underbrace{\textit{the book}}_{\texttt{obj}}\ \underbrace{\textit{yesterday}}_{\texttt{vadv}}$$

So five slots for *gave* are shown. Note that an adjunct slot (like `vadv`) can be filled more than once. In the following, we will just use *slot* to mean *complement slot*, unless otherwise specified.

7 Inventory of slots

The set of (complement) slots available for a word sense is dependent on the POS of the word sense. Let us go through the parts of speech and list the available slots for each, with some explanation.

The possible slots for verbs are:

`subj`	*subject*
`obj`	*direct object*
`iobj`	*indirect object*
`pred`	*predicate complement*
`auxcomp`	*auxiliary complement*
`comp`	*complement*

We assume the reader is familiar with `subj`, `obj`, and `iobj` – especially considering the preceding examples. As mentioned, the `subj` slot need not be specified in verb sense frames. If no `subj` slot is given, the system automatically adds one as if `(subj n)` had been specified. But if there is something special about the subject of a given verb sense, one can specify a `subj` slot with special options or option tests. For example, the entry

```
amaze < ... < v (subj v) obj1
```

allows the subject of (this sense of) `amaze` to be filled by various verbal phrases like `that`-clauses. We will specify more fully below what categories of filler phrases are sanctioned by the various slot options.

What is the `comp` slot? The following ideas are closely related to the ideas of *thematic roles*, as developed especially by Jeffrey Gruber and Ray Jackendoff – see e.g. [2] and [3]. We first describe the situation in the SG framework, and then discuss the relationship with thematic roles.

When a verb describes a motion or change of state, the `comp` slot is often used to specify the new location or new state. Examples:

$$\textit{Alice drove}\ \underbrace{\textit{Betty}}_{\texttt{obj(n)}}\ \underbrace{\textit{to the store}}_{\texttt{comp(lo)}}.$$

$$\textit{Alice drove}\ \underbrace{\textit{Betty}}_{\texttt{obj(n)}}\ \underbrace{\textit{to distraction}}_{\texttt{comp(lo)}}.$$

$$\textit{Alice drove } \underbrace{\textit{Betty}}_{\texttt{obj(n)}}\ \underbrace{\textit{crazy}}_{\texttt{comp(a)}}\ .$$

In the first example, the change of state is a physical motion. The `subj` is the agent of the motion; the `obj` is the thing moved; and the `comp` is the destination (where the `obj` moves to). In the second and third examples, the "motion" is more abstract; it is still a change of state though, and the `comp` is the destination state. Many abstract word senses in human language derive from more primitive, physical word senses by metaphorical extension. We often use the same word (like *drive*) for the original physical sense and the more abstract sense. The three examples above are sanctioned (in part) by the sense frame

```
drive < v obj (comp lo a)
```

for `drive`. The option `lo` (short for "location") is used for the `comp` PP's in the first two examples. The option `a` (short for "adjective phrase") is used for the adjective *crazy* in the last example. We will explain these options in the next section.

When a verb describes a change of state, and the verb has an overt `obj`, the `obj` will typically be the thing that changes. Otherwise the `subj` is typically the thing that changes, as in:

John drove to Chicago.
John went to Chicago.
John went crazy.

When a verb describes simply a state (and not a change of state), the `comp` slot is often used to specify that state. Examples:

$$\underbrace{\textit{Alice}}_{\texttt{subj}}\ \textit{found}\ \underbrace{\textit{Bob}}_{\texttt{obj}}\ \underbrace{\textit{in the kitchen}}_{\texttt{comp}}.$$

$$\underbrace{\textit{Alice}}_{\texttt{subj}}\ \textit{found}\ \underbrace{\textit{Bob}}_{\texttt{obj}}\ \underbrace{\textit{in need of help}}_{\texttt{comp}}.$$

In the first case, the sense of *found* is physical and the `comp` is a physical location. In the second, the sense of *found* is abstract and the `comp` is an abstract "location" (a location in a state space). These two examples are sanctioned by the sense frame:

```
found < v obj (comp lo)
```

When a verb describes a state, and the verb has an overt `obj`, the `obj` will typically be the thing that is predicated to be in that state; otherwise it is typically the `subj`.

In the theory of thematic roles [2, 3], one assigns certain semantic cases (the thematic roles) to complements. The THEME is the central one. For verbs of change, the THEME is what changes, and the GOAL is the destination state (or destination location in the case of physical motion verbs). There can also be a SOURCE state or location. And there can be an AGENT, which causes the change. A given participant might have more than one role, e.g. both AGENT and THEME. For verbs describing states, the THEME is what is predicated to be in that state, and the LOCATION is the state (or physical location in the case of physical location verbs).

One can easily see how these roles map into SG complement slots for the examples given above, with THEME as `obj` or `subj`, AGENT as `subj`, and GOAL and LOCATION as `comp`. Also we use `comp` for SOURCE, as in the slot frame:

```
drive < v obj (comp (p from)) (comp lo)
```

Thinking about thematic roles can often be a useful guide in creating SG lexical slot frames, but it can be hard to find a fit for some verbs, and one gets into inventing new roles. For that reason, we just take these ideas as an informal background that is useful some of the time. In general, the author's approach to semantic representation has been to use versions of predicate logic where word senses become predicates that have arguments using positional notation – see e.g. [15]. When a word sense predicate comes from an SG lexicon, its arguments correspond, in order, to the complement slots listed for the word sense in the lexicon (and there might also be an event or entity argument). Note that, as indicated above, lexical slots should be thought of as *logical* arguments – so that e.g. the `subj` is the *logical* subject. The parser takes care of unwinding passives, (often) showing implicit subjects, remote fillers, etc. The ultimate semantics of such word senses comes from lexical meaning axioms and real world knowledge axioms involving word senses and other predicates and constants.

The `pred` slot is used in ESG only by the verb `be`, whose entry is:

```
be < v (subj n v) pred
```

The slot `pred` is filled by a wide range of phrase types. Examples:

Bob is a teacher.
Bob is happy.
Bob is in love.
Bob is to leave tomorrow.
Bob is leaving tomorrow.
Bob was taken to the station.

The `auxcomp` slot is used in English mainly for the modal verbs, where `auxcomp` is filled by a bare infinitive, for the auxiliary *do*, which also takes a bare infinitive, and for the perfect sense of *have*, where the filler is an active past participle. Two of these cases are illustrated in the parse in Figure 1, which also contains the use of `pred` for the progressive and the passive. In Slot Grammar, auxiliary verbs are viewed as *higher verbs*, as one can see in this parse.

In a verb sense frame there may sometimes be more than one occurrence of a `comp` slot; but for all the other verb (complement) slots there should be at most one occurrence of the slot in a given sense frame.

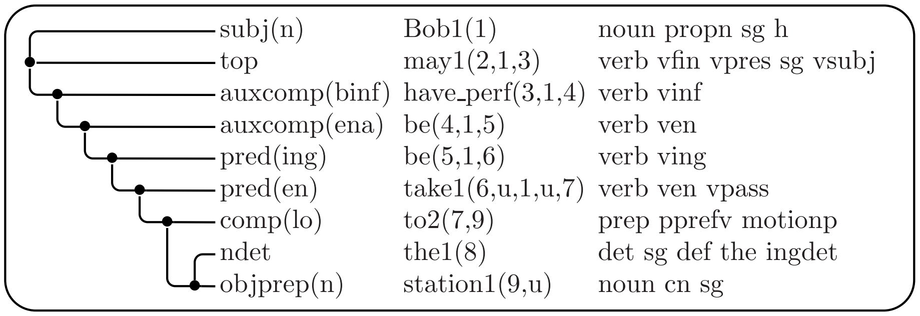
For the noun POS, the situation for available slots is as follows. Although there are four different symbols for noun parts of speech used in the lexicon (`n`, `propn`, `pron`, and `num`), they all actually have the same POS in the grammar, `noun`; and they share the same set of slots. The distinction in the grammar between the four types of noun is shown by features: `cn` is used for common nouns, and the three other types use the same features (`propn`, `pron`, and `num`) as in the lexicon. Thus the correspondence is:

Lexicon	Grammar (POS, feature)
`n`	`noun`, `cn`
`propn`	`noun`, `propn`
`pron`	`noun`, `pron`
`num`	`noun`, `num`

There are just two noun slots (or rather slot names): `obj` and `nid`. The `obj` slot can have many different options, as we will see in the next section. When it is written without any explicit option, the default option is taken as `n`. This is filled by `of`-PPs. Examples are:

```
brother < n obj
department < n obj
construction < n obj
```

The reason for using the name `obj` in this way for nouns is that it often corresponds to the direct object of a verb, when the noun is related semantically



```
subj(n)          Bob1(1)              noun propn sg h
top              may1(2,1,3)          verb vfin vpres sg vsubj
auxcomp(binf)    have_perf(3,1,4)     verb vinf
auxcomp(ena)     be(4,1,5)            verb ven
pred(ing)        be(5,1,6)            verb ving
pred(en)         take1(6,u,1,u,7)     verb ven vpass
comp(lo)         to2(7,9)             prep pprefv motionp
ndet             the1(8)              det sg def the ingdet
objprep(n)       station1(9,u)        noun cn sg
```

Figure 1: *Bob may have been being taken to the station.*

to a verb. For example, *construct X* and *construction of X* are closely related. Actually, the grammar shows the noun `obj` slot as `nobj`. And one can in fact use that symbol also in the lexicon. But it is normal and convenient to write it as `obj` in the lexicon because then related nouns and verbs sometimes have the same slot frame (or one that looks the same in the lexicon). Although a verb sense frame should have only one occurrence of an `obj` slot, a noun sense frame may have more than one occurrence (with different options). Example:

```
assignment < n obj (obj (p to) inf)
```

This allows for example for the NP *the assignment of Bob to those duties* and *the assignment of Bob to write the report*. The second kind of use of `obj` for nouns corresponds to `comp` for verbs.

The noun slot `nid` ("noun identifier") occurs in examples like *page 3* and *Appendix B*. It is filled by proper nouns and numbers.

For adjectives, there is only one (complement) slot available. It is called `aobj` in the grammar (and shows that way in parse trees), but is usually called `obj` in the lexicon. The reason is that this makes it easier to see correspondences with sense frames of nouns that are related to adjective sense frames. As with `nobj`, the slot `aobj` has many different allowable options. We will describe these in the next section.

For adverbs, the situation is much the same as for adjectives: There is one slot named `avobj`, which may also be called `obj` in the lexicon. Again, it has several allowable options, which we will describe in the next section.

Determiners and qualifiers have no complement slots.

Prepositions have just one (complement) slot, `objprep` ("object of the preposition"). This slot is obligatory, but no suffix `1` is needed. It is not normally specified in the lexicon, unless there is an unusual condition on it. Examples:

$\textit{in } \underline{\textit{the house}}$
$\textit{in } \underline{\textit{writing this program}}$

Subordinate conjunctions (`subconj`) have one slot, `sccomp` (subconj complement). As with `objprep`, this obligatory and is not normally specified in the lexicon. Example:

$\textit{if } \underline{\textit{you write this program}}$

Coordinating conjunctions (`conj`) have two complement slots, `lconj` (left conjunct) and `rconj` (right conjunct). Examples:

$$\underbrace{\textit{The cat sat on the mat}}_{\texttt{lconj}} \textit{ and } \underbrace{\textit{the dog lay on the floor}}_{\texttt{rconj}}.$$

$\textit{The cat } \underbrace{\textit{sat on the mat}}_{\texttt{lconj}} \textit{ and } \underbrace{\textit{watched the birds}}_{\texttt{rconj}}.$

$\textit{The cat } \underbrace{\textit{watched}}_{\texttt{lconj}} \textit{ and } \underbrace{\textit{then chased}}_{\texttt{rconj}} \textit{ the squirrel.}$

These slots are dealt with in the grammar and are not normally mentioned in the lexicon.

The infinitive *to*, `infto`, has the complement slot `tocomp`, as in *to* $\underline{\textit{eat the chocolate}}$. This is sanctioned by the sense frame:

```
to < ... < infto (tocomp binf)
```

The option `binf` stands for "bare infinitive verb phrase".

The POS `subinf` (analogous to `infto`) has the complement slot `subinfcomp`, as given in the entry:

```
in =order to < subinf (subinfcomp binf)
```

The POS `forto` has the two complement slots `forsubj` and `forcomp`, as given in the entry:

```
for < ... < forto (forsubj n) (forcomp inf)
```

Example:

$\textit{This would allow for } \underbrace{\textit{your sister}}_{\texttt{forsubj}} \underbrace{\textit{to be there earlier}}_{\texttt{forcomp}}.$

Finally, the POS `thatconj` has the slot `thatcomp`, as given in the entry:

```
that < ... < thatconj (thatcomp bfin)
```

Example:

He said that $\underline{\textit{the chocolate was delicious}}$.

8 Inventory of slot options

The slot options are:

```
a, agent, aj, av, bfin, binf, dt, en, ena,
fin, fina, finq, finv, ft, ger, gn, inf, ing, io, it,
itinf, itthatc, itwh, lo, n, na, nen, nmeas, nop, nummeas,
p, padj, pinf, pinfd, prflx, prop, pt, pthatc, pwh,
qt, rflx, sc, so, thatc, v, wh
```

Very roughly, each option for a slot corresponds to a category of phrases that can be a filler of the slot. For example, `n` corresponds (roughly) to noun phrases, `a` to adjective phrases, `p` to prepositional phrases, `ing` to present participial (-ing) verb phrases, etc. Many of the options are shared across several slots. That's why we are describing them here in a separate section. In some cases, a given option means something slightly different for different slots. We will now go through some of the most important options, and for each one say which slots it applies to and what it means for each slot.

Most options apply to the verb slots `obj` and `comp` and to the `obj` slot for nouns, adjectives, and adverbs (= `nobj`, `aobj`, `avobj`, respectively). This will be the case in all the following examples, unless we say otherwise.

`n`: For the verb POS, `n` can be used with the slots `subj`, `obj`, `iobj`, and `comp`. For all of these slots, it can be filled by noun phrases. For `subj` and `obj`, it can be filled by two NP-like kinds of adjective phrases. The first is illustrated by *the very rich* and has an adjective head word marked with the feature `adjnoun`. The second type is illustrated by *the happiest*, with a superlative head adjective. For the slots `subj`, `obj`, and `comp`, `n` can be filled also by wh-verb phrases like *what he tried to find* and *whatever they want*, which are introduced by a wh-word marked with the feature `whnom` in the lexicon. An example that uses the `n` option in all of `subj`, `obj`, and `comp` is:

$$\underbrace{\textit{They}}_{\texttt{(subj n)}} \quad \textit{elected} \quad \underbrace{\textit{her}}_{\texttt{(obj n)}} \quad \underbrace{\textit{president of the association.}}_{\texttt{(comp n)}}$$

The `pred` slot implicitly has an `n` option, since it can be filled by NPs. But we do not show it explicitly in the lexicon.

For the noun POS, `n` can be an option for the `obj` (= `nobj`) slot, where it is filled by `of`-PPs, as in *president of the association.*

For the `adj` POS, `n` can be filled by a noun phrase. This is a rare construction. An example is:

It is due $\underline{\textit{me}}$*.*

`io`: This is an option only for the verb `comp` slot. The slot+option `(comp io)` is filled by an NP and it is essentially the same as `(iobj n)`. It is provided for the `comp` slot because sometimes it is convenient to include `io` along with other `comp` options, for example as in:

```
throw < v obj (comp io lo)
```

Here the `lo` ("location") option (which we will come to soon) is filled by various PPs like *to the dog* and *into the yard.* So this includes examples like:

$$\textit{He threw } \underbrace{\textit{the dog}}_{\texttt{(comp io)}} \; \underbrace{\textit{a ball}}_{\texttt{(obj n)}} .$$

$$\textit{He threw } \underbrace{\textit{a ball}}_{\texttt{(obj n)}} \; \underbrace{\textit{to the dog}}_{\texttt{(comp lo)}} .$$

$$\textit{He threw } \underbrace{\textit{a ball}}_{\texttt{(obj n)}} \; \underbrace{\textit{into the yard.}}_{\texttt{(comp lo)}}$$

There is little difference between `iobj` (= `(iobj n to)`) and `(comp io (p to))`. But the `comp` version of the indirect object can include other options as well.

`p`: This option, shown in several examples already in this report, is used for PP fillers and normally has option tests that name specific prepositions, as in:

```
admonish < v obj (comp (p about against for))
```

The `p` option can actually be used without any tests (without naming specific prepositions), in which case it will be fillable by any PP; but this usage is rare. The slot-option combination

`(comp (p` $Prep_1$ $Prep_2$ `...` $Prep_n$`))`

can be written simply as:

`(p` $Prep_1$ $Prep_2$ `...` $Prep_n$`)`

and this is quite common. So the above example for `admonish` could be written as:

```
admonish < v obj (p about against for)
```

For nouns, adjectives, and adverbs, we can use an abbreviation similar to the one for `comp` above. For example, we could write either

```
admonition < n obj (obj (p about against for))
```

or simply

```
admonition < n obj (p about against for)
```

`pt`: This option can be filled by particles (definition given above). The situation is quite similar to that with `p`. We also allow abbreviations like those for `p`. Example:

```
cluster < ... < v obj (pt up together) (p around round)
```

`lo:` As mentioned, this option stands for "location". It is filled by three types of phrases: (1) Particles. These are listed above. (2) PPs based on certain prepositions. These are roughly supposed to be prepositions that indicate a location (or direction). Precisely, they are the single-word prepositions that are *not* marked with the feature `nonlocp` in the lexicon and the multiword prepositions that are marked with the feature `locmp` in the lexicon. (3) NPs that have the feature `locnoun`. These can be, for example, nouns of linear measure like *mile*. So for example if we have the entry

```
drive < obj (comp lo)
```

we are allowed examples like:

$$\textit{She drove her } \underbrace{\textit{car}}_{\texttt{obj}} \; \underbrace{\textit{out}}_{\texttt{(comp lo)}} .$$

$$\textit{She drove her } \underbrace{\textit{car}}_{\texttt{obj}} \; \underbrace{\textit{onto the interstate}}_{\texttt{(comp lo)}} .$$

$$\textit{She drove her } \underbrace{\textit{car}}_{\texttt{obj}} \; \underbrace{\textit{fifty miles}}_{\texttt{(comp lo)}} .$$

As with `p` and `pt`, we have abbreviations: Simply `lo` can be used in place of `(comp lo)` for verbs and in place of `(obj lo)` for the other open-class POSs (nouns, adjectives, adverbs).

`inf:` This option is filled by `infto` phrases, like *to eat chocolate*. Examples:

```
admonish < v obj (comp inf (p about against for))
admonition < n obj (obj inf (p about against for))
able < adj (obj inf)
```

`binf:` This is filled by infinitive VPs (without `to`), like *wash the dishes*. It is used for example with slots `comp` and `auxcomp`. Example: *Alice made Billy wash the dishes*.

`ing:` This is filled by present-participial (-ing) VPs, like *eating the chocolate*. Example: *Bob likes eating chocolate*.

`ger:` This is filled by ing-VPs that have the feature `gerund` – they are modified by a determiner `the` or a possessive pronoun determiner. This option is used only for the verb `obj` slot.

`en:` This is filled by passive past-participial VPs. Example: *They wanted the food taken away*.

`ena:` This is filled by active past-participial VPs, as in *He has eaten the chocolate*. It is relevant only for the `auxcomp` slot for verbs.

`fin`: For verbs, this is used for the `obj` and `comp` slots, and is filled by finite VPs, as in *He said she was smart* and that-clauses, as in *He said that she was smart*. Also, for the verb `obj` slot, it can be filled by `so`, as in *Alice said so*. For the other open-class POSs, `fin` is used for `obj`, but then it is filled only by that-clauses.

`bfin`: For ESG this is used only with the `thatcomp` slot, and is filled by a finite VP, as in *that she was smart*.

`finv`: This option is used mainly for the `obj` slot for nouns, adjectives and adverbs, where it is filled by both finite VPs and that-clauses. For the verb `obj` slot, it is filled only by finite VPs.

`finq`: This is filled, for verb slots `obj` and `comp`, by quoted finite VPs.

`thatc`: This is filled by that-clauses.

`wh`: This is filled by wh-VPs, wh-nouns, wh-adverbs, and `if` and `whether` clauses. Examples: *He knows what they want. He knows whether they will go*.

`ft`: This is filled by `forto` phrases, as in: *This allows for them to take it*.

`prop`: This is filled by `infto` phrases, `forto` phrases, or that-phrases.

`v`: This is used only for the (verb) `subj` slot. It allows `subj` to be filled by a wide range of “verbal” phrases – `ing` VPs, `infto` phrases, that-clauses, wh-clauses, `forto` phrases – plus wh-adverbs.

`a`: This is filled by adjective phrases, as in

$$\textit{He drove her } \underbrace{\textit{crazy}}_{\texttt{(comp a)}}\ .$$

`av`: This is filled by adverb phrases, as in

$$\textit{She will do } \underbrace{\textit{very well}}_{\texttt{(comp av)}}\ .$$

`qt`: This is filled by quoted NPs, for the verb `obj` and the noun `obj` slots.

`it`: This is filled by the pronoun `it`. It is used only for the verb `obj` and `subj` slots, as in *I like it that he's there*.

`so`: This is filled by the adverb `so`, as in *He did so*.

`sc`: This is similar to the `p` option, but it is for subordinate conjunctions (`subconj`) instead of prepositions. Example entry:

```
feel < v (comp a en (p like (up to) for)
                   (sc as\ if as\ though like)
```

The escape character '\' is used here before the blanks in the `subconj`s `as if` and `as though`. So the `sc` option here allows examples like:

$$\textit{I feel } \underbrace{\textit{as if I could eat a pound of chocolate}}_{\texttt{(comp sc)}}.$$

We will explain the nature of the construction `(up to)` in the next section. It allows *They didn't feel* <u>*up to it*</u>.

`nummeas:` This is filled by a number – a pure number (in either digital or literal form), or a percentage in the form of digits followed by `%`, or a dollar amount in the form of `$` followed by digits.

`nop:` This option ("no option") is a kind of catch-all for unusual slots, and a default in case the option does not get set. For example, the top node of a parse tree is marked as filling the "slot" `top` with option `nop`.

9 Option tests

We have already mentioned that a slot option can have tests associated with it, like so:

$(OptionName\ Test_1\ Test_2\ \cdots\ Test_n)$

where the tests are interpreted disjunctively, and one of them must succeed in order for the slot-filling with that option to succeed. We have mentioned that the most common kind of option test is an atomic symbol which is taken to be the (required) head word of the filler, and we have given several examples of this for the `p` option. Let us now describe the other possible option tests. All the other option tests will be specified by lists ($x\ y\ \cdots$). In most cases, the first member x of the list is a special operator, and the other members $y\ \cdots$ are arguments. The arguments may recursively be other tests. Several of the operators can take an arbitrary number of arguments. The allowable operators are:

```
st, fe, f, of, nf, hd, sn, ph, wds, sa, ev, &, |, ^
```

Let us go through these one by one:

`st:` This can take any number of arguments. The default case is that it checks that the filler phrase has each argument as a semantic type, where the check uses the hierarchy of the *ontology*, which we will explain in Section 10 below on features. In addition there is a special way that `st` can numerically reward or penalize the presence or absence of a semantic type for the filler. If the test is of the form

`(st` x m`)`

where m is a number, then the test always succeeds, and a reward of m is added if x is a feature of the filler. (If m is negative, it acts as a penalty.) If the test is of the form

`(st` x m n`)`

where m and n are numbers, then the test always succeeds; a reward of m is added if x is a feature of the filler, else a reward of n is added. These special forms of `st` allow semantic type tests to be "soft" – not being absolute requirements, but just preferred or not preferred. Examples: The sense frame

```
eat < v (obj (n (st food)))
```

would allow a direct object for `eat` exactly when that object satisfies the conditions for the `n` option (most typically an NP) and has the feature `food`. But the sense frame

```
eat < v (obj (n (st food 1)))
```

would not *require* the `obj` to be a `food`, but simply give it a reward of `1` if it has that type.

`fe`: Takes any number of arguments, which should be morphosyntactic features. It tests that all of these are features of the filler phrase.

`f`: This is like a combination of `st` and `fe`. It takes any number of arguments, which could be morphosyntactic features, parts of speech, or semantic types, and it checks that all of them are features of the filler.

`of`: Like `f`, but checks that at least one of the arguments is a feature of the filler.

`nf`: Like `f`, but checks that no argument is a feature of the filler.

`hd`: Takes any number of arguments, which should be citation forms of words. It tests that at least one of them is the citation form of the head word of the filler. This is the same test as would be made with a top-level option test that is atomic, but in a complex test, we need to use `hd`.

`sn`: Takes any number of arguments, and tests that at least one is the sense name of head word of the filler.

`ph`: The general form of a `ph` expression is:

`(ph` *HeadWord LeftWords RightWords*`)`

Example:

```
(ph bull (a) (in a china shop))
```

Generally, `ph` puts a requirement on the words of the filler phrase. The *HeadWord* must be (the citation form of) the head word; the *LeftWords* are the words to the left of the head in the filler phrase (but they are specified in reverse order); the *RightWords* are the words to the right of the head in the filler (specified in normal order). So the previous example would require the filler to match the (idiomatic) phrase *a bull in a china shop* (*china* = "fine dishes", etc.). The `ph` expression may have the simpler form

(`ph` *HeadWord LeftWords*)

in which case there is no requirement on the *RightWords*.

`wds:` The general form of a `wds` expression is:

(`wds` $Word_1$ $Word_2$ $\cdots$ $Word_n$)

Example:

```
(wds a bull in a china shop)
```

Like `ph`, `wds` puts a requirement on all the words of the filler phrase. The argument words must match exactly the (lower case forms of) the words of the whole filler phrase, from left to right. Obviously, `wds` is easier to use than `ph`, but it is slightly less efficient.

`sa:` This tests the current subject areas in effect, and the tests can be absolute requirements, or create penalties or rewards. This is exactly the same kind of subject area test that can appear on the top level in a sense frame; it will be described in Section 11, page 32, below.

`ev:` Same as `sa`. Both `ev` and `sa` can set parse scores or test subject areas, or do a mixture. See Section 11.

`&:` This can take any number of arguments, and succeeds iff all of the arguments succeed.

`|:` This can take any number of arguments, and succeeds iff at least one argument succeeds.

`^:` This can take any number of arguments, and succeeds iff all of the arguments fail.

We have now looked at the option tests based on special operators. A further kind of option test is also specified as a list:

(w_1 w_1 $\cdots$ w_n)

but the w_i's are all words. This kind of test is satisfied when the w_i's match the initial words of the filler up through the head word. This appeared in an example above:

```
feel < v (comp a en (p like (up to) for)
               (sc as\ if as\ though like)
```

One of the option tests for the `p` option is `(up to)`. So this will be satisfied when the PP has the initial words `up` and `to`. The particle `up` can premodify the preposition `to` via an adjunct slot called `padv`.

10 Features

In a sense frame, the atomic symbols that are not the POS or slot names are understood to be features, which are being asserted to be features of that word sense. So features can have two complementary roles in the lexicon: They can be used as *tests*, as in the option tests discussed in Section 9. Or they can be used as *assertions*, as we are discussing in the current section. They can also be tested and asserted in the syntactic component of the grammar – see [12].

As indicated in Section 9, features can be divided roughly into *grammatical* (or *morphosyntactic*) features and *semantic* features. Grammatical features may represent morphological characteristics of words, or may have more to do with the syntactic usage of the words. Semantic features (also called *semantic types* or *concepts*) are names for sets of things in the world. For instance the semantic feature `st_anim` stands for the set of animate things – including humans, cats, birds, etc. These "things in the world" can include events, states, etc., and so for example an event referred to by a verb sense could have a semantic type. It is most typical (and useful) to mark semantic types on noun senses, but they could be marked on any part of speech.

When a semantic type is asserted of a proper noun, then it is understood that the referent of the proper noun is a *member* of the semantic type (as a set). But otherwise, the word sense is understood as a *subset* of the semantic type.

Let us look now at the inventory of the main grammatical features possible in an SG lexicon. After that, we will discuss semantic features in more detail.

We try to use as uniform a set of grammatical features across the different languages as possible. Of course some features are not applicable to all the languages. The SG system basically works with the union of all the possible features, and some of them just do not get used in all the *X***SG**s. The main grammatical features are as follows, and we organize them largely by part of speech.

Shared features. There are some features that are shared across several parts of speech. These include: `sg` (singular), `pl` (plural), `dual`, `cord` (coordinated), `wh`, and `whnom`. The last is used on determiners, nouns, and verbs; its ultimate purpose is to mark clauses, like *what you see*, that can be used essentially anywhere an NP can.

Verb features. We list these in alphabetical order.

`badvenadj`. A past participle of such a verb does not easily fill the `nadj` slot. Example: *said*.

`ingprep`. A verb whose *ing* form behaves like a preposition. Example: *concern*.

`invertv`. A verb (like *arise* or *come*) that allows its subject on the right, with a `comp` PP or *there* left modifier of the verb.

`npref`. Prefers to modify nouns over verbs, as head of non-finite VP.

`objpref`. Verb preferring NP object over finite clause or *that*-complement.

`postcomp`. Means that the verb cannot have a `comp` slot filler preceding a filler of `(obj n)`.

`sayv`. Verb of saying.

`sta`. Stative verb.

`thatcpref`. Verb allowing both finite VP or *that*-complement but preferring the latter.

`transalt`. Verb (like *increase*) allowing transitivity alternation. The theme (the entity undergoing change) can be either the direct object or the subject (when no direct object is given).

`vcond`. Conditional mood.

`ven`. Past participle.

`vfin`. Finite verb.

`vfut`. Future tense (for Latin languages).

`vimperf`. Imperfect (for Latin languages).

`vimpr`. Imperative VP.

`vind`. Indicative mood.

`vinf`. Infinitive.

`ving`. Present participle.

`vpass`. Passive `ven` verb, as in *he was* <u>*taken*</u>.

`vpast`. Past tense.

`vpers1`. First person.

`vpers2`. Second person.

`vpers3`. Third person.

`vpluperf`. Pluperfect (for Latin languages).

`vpref`. Prefers to modify verbs over nouns, as head of non-finite VP.

`vpres`. Present tense.

`vrelv`. Verb that easily allows a relative clause modifier of its subject to be right-extraposed.

`vsbjnc`. Subjunctive.

`whever`. A clause like *whatever you see* or *whichever road you take* that is modified by an extraposed `wh-ever` NP.

Noun features.

`acc`. Accusative.

`advnoun`. A noun that can behave adverbially. Main examples: time nouns and locative nouns.

`cn`. Common noun.

`dat`. Dative.

`def`. Definite pronoun.

`detr`. Noun requiring a determiner when it (the noun) is singular.

`dy`. Day.

`encprn`. Can be an enclitic (for the Latin languages).

`f`. Feminine.

`gen`. Genitive.

`goodap`. Can easily be a right conjunct in comma coordination even though it itself is not coordinated.

`h`. Human.

`hplmod`. Noun that allows a plural `nnoun` modifier (adjunct slot most commonly filled by common nouns).

`indef`. Indefinite pronoun.

`iobjprn`. Can be non-clitic iobj (Latin languages).

`lmeas`. Linear measure.

`loc`. Indicates a location.

`locnoun`. Nouns that can be used adverbially because they specify a location, measure, etc.

`m`. Masculine.

`meas`. Measure.

`mf`. Masculine or feminine.

`mo`. Month.

`nadjpn`. Pronoun that can fill `nadj` and must agree with head noun (Latin languages).

`nadvn`. Noun that can fill `nadv` and must agree with head noun (Latin languages).

`nom`. Nominative.

`nonn`. A noun that cannot have an `nnoun` modifier.

`notnnoun`. A noun that cannot be an `nnoun` modifier.

`npremod`. Can be a noun premodifier of a noun even though it is also an adjective.

`nt`. Neuter.

`num`. A number noun.

`objpprn`. Can be object of preposition (Latin languages).

`objprn`. Can be non-clitic obj (Latin languages).

`oreflprn`. Can be use *only* as a reflexive (Latin languages).

`percent`. A percent number noun.

`perspron`. Personal pronoun.

`pers1`. First person.

`pers2`. Second person.

`pers3`. Third person.

`plmod`. Noun that can be an `nnoun` even when it is plural.

`posit`. Position (like *middle* or *end*).

`poss`. Possessive pronoun.

`procprn`. Can be proclitic (Latin languages).

`pron`. Pronoun.

`propcn`. A common noun, like *society*, that has certain proper noun qualities.

`propn`. Proper noun.

`quantn`. Denotes a quantity (like *all* or *half*).

`reflprn`. Reflexive pronoun.

`relnp`. An NP that can be the relativizer of a relative clause.

`title`. A title for a human name, like *Doctor* or *Dr.*.

`tm`. Time noun.

`tma`. Time noun that is more pronoun-like, like *yesterday*.

`tmdetr`. Time noun requiring a determiner.

`tmrel`. Time noun allowing certain finite clauses as `nrel` modifiers.

`tonoun`. A noun, like *school*, that can by itself (without premodifiers) be the `objprep` of *to*, even though it is also a verb.

`uif`. Uninflected.

`way`. Manner/way.

`whevern`. An NP like *whatever* or *whichever road*.

Adjective features.

`adjnoun`. Adjectives like *poor* that can have a *the* premodifier and act like the head of an NP.

`adjpass`. An adjective like *delighted* that is also a past participle, but the past participle is not allowed to fill `pred(en)`.

`aqual`. (For German) an adjective, like *denkbar*, that can premodify an adverb (filling `advpre`).

`compar`. Comparative.

`detadj`. Adjectives like *next* and *last* that have an implicit definite article.

`erest`. Can use *-er* and *-est* for comparative and superlative. Applies to adverbs too.

`lmeasadj`. Adjective like *high* allowing constructions like *three feet high*.

`noadv`. (For German) an adjective that cannot be used as an adverb.

`noattrib`. Not allowed as filler of `nadj`.

`nocompa`. Not allowed as filler of `comp(a)`.

`nocompare`. Not allowing comparison (for Latin languages).

`nopred`. Not allowed as filler of `pred`.

`nqual`. Adjectives like *medium* that allow constructions like *a medium quality car*.

`post`. Adjective like *available* that can easily postmodify a noun, as in *the first car available*.

`soadjp`. Gets put by the grammar on an adjective phrase like *so good* to allow that phrase to fill `nadv` in an NP like *so good a person*. This is done when the adjective (like *good*) is premodified by a qualifier marked `soqual`.

`superl`. Superlative.

`tmadj`. A time adjective like *early*.

`toadj`. Similar to `tonoun`.

Adverb features.

`badadjmod`. Preferred not to modify adjective.

`compar`. Comparative.

`detadv`. Can modify a determiner.

`initialmod`. Modifies on left only as first modifier.

`introadv`. Adverb like *hello* that easily left-coordinates by comma-coordination.

`invertadv`. Adverb that allows certain constructions *Adv Verb Subj*.

`loadv`. Easily modifies a locational prep or adverb (particle).

`locadv`. Locative adverb like *above*.

`notadjmod`. Cannot modify an adjective.

`noadvpre`. Cannot have (qualifier) premodifier.

`nopadv`. Cannot modify preposition.

`notinitialmod`. Cannot appear clause-initially.

`notleftmod`. Cannot modify verb on left.

`notnadv`. Cannot premodify a noun.

`notrightmod`. Cannot modify verb on right.

`nounadv`. Can modify noun.

`nperadv`. Can fill `nper` slot for nouns, like *apiece* and *each*.

`npost`. Can postmodify a noun in slot `nadjp`.

`partf`. Adverb that can be a particle (also applies to prepositions that can be particles).

`post`. Can postmodify (like *enough*).

`ppadv`. Can modify preposition (with no penalty).

`prefadv`. Adverb analysis as `vadv` is preferred over noun analysis as `obj`.

`reladv`. For Spanish, an adverb like *cuando* that can create a relative clause.

`superl`. Superlative.

`thereprep`. Adverb like *thereafter, thereof, ...*

`tmadv`. Time adverb like *before, early, ...*

`vpost`. Cannot modify a finite verb on the left.

`interj`. An interjection.

Determiner features.

`all`. Only for the determiner *all*.

`ingdet`. Marked on `possdet`s and *the*. Can premodify present participle verbs.

`possdet`. Possessive pronoun as determiner.

`prefdet`. Preferred as determiner (over other parts of speech).

`reldet`. For Spanish, a determiner like *cuyo* that can create an NP serving as relativizer.

`the`. Marked only on *the*.

Qualifier features.

`badattrib`. If it modifies an adjective, then that adjective cannot fill `nadj` slot.

`c.` Modifies only comparative adverbs.

`post.` Can postmodify.

`pre.` Can premodify (the default).

`soqual.` See `soadjp` for adjectives above.

Subordinate conjunction (`subconj`) features.

`assc.` For *as* (or analogs in other languages).

`comparsc.` For `assc` or `thansc`.

`finsc.` Allows only finite clause complements (filling `sccomp`).

`notleftsc.` A clause with this as head cannot left-modify a clause (as `vsubconj`). Example: *for*.

`okadjsc.` Allows adjective complement.

`oknounsc.` Allows noun complement.

`oknsubconj.` Can fill `nsubconj`.

`poorsubconj.` Preferred not as `subconj`.

`sbjncsc.` For the Latin languages: Suppose a `subjconj` S has a finite clause complement C (C is filler of `sccomp`). Then S must be marked `sbjncsc` if C is marked `vsbjnc` (subjunctive) but not `vind` (indicative). And if S is marked `sbjncsc` and C is marked `vsbjc`, then remove `vind` from C if it is present.

`thansc.` For *than* (or analogs in other languages).

`tosc.` Allows `infto` complement.

`whsc.` A wh-`subconj` (like *whether*).

Preposition features.

`accobj.` (For German) allows the `objprep` to be accusative.

`adjobj.` (For German) allows the `objprep` to be an adjective or past participle phrase.

`asprep.` For *as* and analogs in other languages.

`badobjping.` Cannot have a `ving` `objprep` (under certain conditions).

`daprep`. For German. For PPs like *dabei* and *darauf*. A word *da+Prep* is unfolded to a PP with head *Prep* marked `daprep` and `objprep` filled by *es*.

`datobj`. (For German) allows the `objprep` to be dative.

`genobj`. (For German) allows the `objprep` to be genitive.

`hasadjobj`. (For German) the `objprep` is an adjective or past participle phrase.

`infobj`. (For Latin languages) has an `objprep` that is an infinitive VP.

`locmp`. Used for multiword prepositions that fill `comp(lo)`.

`motionp`. Prefers not to fill `comp` if the matrix verb is marked `sta`.

`nonlocp`. Cannot be a filler of `comp(lo)`.

`notwhpp`. Cannot have wh `objprep` (under certain conditions).

`pobjp`. The `objprep` can be a PP itself. Example: *from*.

`ppost`. The preposition can follow the `objprep`. If the preposition is marked `ppost` but not `ppre`, then the preposition *must* be on the right.

`ppre`. The preposition can precede the `objprep`. There is no need to mark this feature on the preposition, in order to allow the preposition to be on the left, unless the preposition is marked `ppost`.

`pprefn`. Prefers to modify nouns.

`pprefv`. Prefers to modify verbs.

`preflprn`. Marked by the grammar on a PP when the `objprep` is a reflexive pronoun.

`relpp`. A PP that can serve as a relativizer for a relative clause.

`staticp`. Preposition like *in* or *on* that is used normally to represent static location vs. goal of motion (as with *into* and *onto*).

`timep`. Common modifier of time nouns, as in *two days* $\underline{\textit{after}}$ *the meeting*.

`timepp`. For certain PPs where the `objprep` is a time noun.

`woprep`. Like `daprep`, but with *wo* instead of *da*.

Normally semantic types should be taken from the *ontology lexicon* `ont.lx`. This lexicon is the same for all the languages. The ontology includes not only a set of semantic types (or concepts), but also a specification of the subset relation between the types:

$Type_1 \subset Type_2$

For instance the set `h` of humans is a subset of the set of animate beings: `h` $\subset$ `st_anim`. The SG ontology is basically open-ended. Currently it contains about 450 concepts, but it may be expanded considerably.

Generally, semantic types stand for sets (or classes) of things in world that are important in determining word senses, producing better parses, choosing target words in machine translation, etc. For example for better parsing we might put semantic constraints on fillers of slots, as described in Section 9, page 21, or even make them "soft" constraints as described there. For MT, we can use semantic type tests for lexical transfer. For example, the English verb `eat` translates into German `essen` or `fressen` depending roughly on whether the (logical) subject is human or not. If nouns are marked with semantic features that show whether they are human or not, then the MT system can look at the logical subject of `eat` (if present) in a parse tree to make this deciding test.

The subset relation $\subset$ for semantic types, mentioned above, is represented in `ont.lx` as follows. For each semantic type f, `ont.lx` has an entry of the form:

f `<` f_1 `<` f_2 `< ... <` f_n

where the f_i's are all the features g satisfying $f \subset g$, i.e. all the features g of which f is a subset. The f_i's need not be in increasing order, or even related among one another; we are merely listing all features which are supersets of f. For example, we might have:

```
man < male < human < animate
```

And here the set of males is not a subset of the set of humans.

When the SG parser checks an option test `(st` g`)`, it looks at each feature f of (the head word of) the filler phrase, looks up f in the ontology lexicon, and checks whether g is either f or among the features listed with f (as supersets of f).

It is good to use the features in `ont.lx` as semantic types in the lexicon, so that the processing can take advantage of the hierarchy in `ont.lx`. However, it is not really necessary to use only these as semantic types. You can make them up *ad libitum* as long as you do not need to depend on a hierarchy of types.

11 Subject area tests

Consider the sentence:

The bat flew out of his hands.

If the subject area is sports, or even more explicitly baseball, then *bat* is likely to mean *baseball bat*. If instead the subject area is something like nature studies, then *bat* is more likely to mean a certain kind of flying mammal.

Generally, subject areas, or domains, are general topics that we may talk about, write about, work in, deal with, be interested in, etc. Typical examples are entertainment, arts, finance, clothing, eating, cooking, sports, politics, world news, travel, economics, computers, mathematics, biology, etc. The topics of courses in schools and universities are subject areas. The major categories on the home page of a web service like Yahoo! are subject areas.

Typical subject areas are not like typical semantic types in an ontology. The most typical semantic types are common noun senses, but subject areas seem to behave more like proper nouns. For instance, mathematics can be considered a unique thing in the world. Some languages indicate that distinction by using a definite article before a subject area, like *die Mathematik* in German. One possible way though of viewing a subject area as a semantic type is to think of the subject area as standing for the set of all activities in that subject area. For instance we could view the subject area `mathematics` as being the set of all mental activities, writings, discussions, etc., in the mathematical domain.

When SG is used, subject area settings can be made for some duration of the processing – e.g. for a whole document, or a certain portion of a document. If a web page has meta tags that show the subject area(s), then those might be set for processing of that web page. More than one subject area may be set or "in effect" at one time.

A sense frame can specify one or more subject area tests, in the form

`(sa` *Test*`)`

where *Test* is either the name of a subject area, or a *flag test* (explained below), or a Boolean combination of such tests using `&` (and), `|` (or), or `^` (not). Examples:

`(sa computers)`
: Tests that the subject area is (includes) `computers`.

`(sa (| computers finance))`
: Tests that the subject area is `computers` or `finance`.

If `(sa` *Test*`)` fails, then the sense frame containing it will not be used in parsing the current sentence.

A *flag test* (within an `(sa ...)` expression) is of the form

`(fl` *Flag*`)`

where *Flag* is an SG flag name (for example `html`, meaning that HTML text is assumed as input). Two special flags are also allowed: (a) `ucseg` is on when in the current segment all the words are in uppercase. (b) `lcseg` is on when in the current segment there is at least one noun or verb in lowercase. So we might have an entry like the following for "US" in the sense of an abbreviation for "United States":

```
US < propn st_country (sa (^ (fl ucseg)))
```

This prevents "US" from being taken as a proper noun (and ESG will get only the pronoun "us") when the segment is all uppercase.

There is actually a more general form of an `sa` test which can result in penalizing (or rewarding) a sense frame instead of ruling it out totally. A penalty or reward can be given to the sense frame which enters into the numerical scoring system that SG uses during parsing. A penalty can say that the current sense frame is less likely to be a valid sense frame. Penalties are positive numbers, and rewards are negative numbers. Zero is equivalent to no penalty or reward. An example of an `sa` test with penalty is this:

```
(sa computers 0 2)
```

This says: "If the subject area is `computers`, then give a penalty of `0` (okay); else impose a penalty of `2` (and succeed)". Another example is:

```
(sa computers 2 finance)
```

This says: "If the subject area is `computers`, then give a penalty of `2` (and succeed); else test whether the subject area is `finance` and succeed or fail accordingly".

In general if we have:

```
(sa $SubjAreaTest_1$ $Penalty_1$ $SubjAreaTest_2$ $Penalty_2$ ... )
```

then it behaves like so:

```
if ($SubjAreaTest_1$)
   $Penalty_1$, Succeed;
else if ($SubjAreaTest_2$)
   $Penalty_2$, Succeed;
   ...
```

If the last thing is a subject area test, then it has to succeed in order for the `sa` to succeed. If you have

```
(sa $Test_1$ $Penalty_1$ $Penalty_2$)
```

then this always succeeds, and it is like

```
if ($Test$)
   $Penalty_1$;
else
   $Penalty_2$;
```

More generally, the algorithm works like so: `sa` can be given any number of arguments:

(sa x_1 x_2 x_3 $\cdots$)

Each argument x_i can be either a number or a subject area test. The "interpreter" goes through the arguments from left to right and behaves as follows:

```
(1) If you are at the end of the argument list, then sa succeeds.
(2) Else let $x$ be the next argument and advance the argument list by one.
    (A) If $x$ is a subject area test, then evaluate $x$.
        (a) If $x$ is true, then go to (1).
        (b) If $x$ is false, then
            (i) If there are no remaining arguments, then sa fails
                (and the lexical analysis is not used).
            (ii) Else skip over the next argument and go to (1).
    (B) Else if $x$ is a number, then it takes that number as a penalty
        (or reward if the number is negative) and succeeds.
```

Note that this algorithm actually allows the special case:

(sa n)

where n is a number, and it then just succeeds, taking n as the penalty. Because of this, there is another name, `ev` (evaluation), for `sa`. In any of the uses of `sa` described above, the symbol `ev` can be used instead of `sa`.

12 Sense names

A sense frame may (but need not) contain a sense name specification, of the form:

(sn *SenseName*)

which assigns that sense name to the word, as it is used with the given sense frame in parsing. Example:

```
have < v (auxcomp ena) (sn have_perf) < ...
```

This sense frame is used for the perfect auxiliary sense of `have`, and this sense then gets the sense name `have_perf`.

If no sense name specification is given for a sense frame, then the lexical processor assigns one automatically, by taking the index word and appending a number. The number is just the ordinal number of the given sense frame in the list of the sense frames given for that word. For example if we have the entry:

```
man < n h m < v obj1
```

then the first (noun) sense would get the sense name `man1`, and the second (verb) sense would get the sense name `man2`.

Sense names show up in parse output as predicates in word sense predications.

13 Inflectional elements

The index word for a sense frame is always viewed as a citation form of a word. It is the uninflected form of the word. For English verbs, this form can always be used as the infinitive of the verb, and for all verbs except *be* it can be used in the present tense for all person-number combinations except third person singular. For English nouns, the citation form is the singular form of the noun – unless the noun exists only in a plural form. SG has a morphological analyzer, so that regular inflections of words do not need to be given in the lexicon. However, for some languages, including English, irregular forms of words are listed as index words in the lexicon and there are corresponding *inflectional elements* that show the nature of the inflection and the corresponding citation form.

Examples:

```
ate < (ved eat)
eaten < (ven eat)
made < (veden make)
rating < (ving rate)
am < (vpers1 be)
is < (vsg be)
are < (vpl be)
men < (npl man)
better < (compar good)
best < (superl good)
```

These show, respectively, that

`ate` is the past tense of `eat`,
`eaten` is the past participle of `eat`,
`made` is both the past tense and past participle of `make`,
`rating` is the present participle of `rate`,
`am` is the first person singular present of `be`,
`is` is the third person singular present of `be`,
`are` is the plural present of `be`,
`men` is the (noun) plural of `man`,
`better` is the comparative form of the adjective `good`,
`best` is the superlative form of the adjective `good`.

These examples illustrate all of the inflectional operators, and their meanings should be clear.

Sometimes the same word form may be a citation form for one word sense and an inflected form for another word sense. In this case we show both a sense frame and an inflectional element in the same entry. Example:

```
saw
   < (ved see)
```

```
      < n
      < v obj (pt down off up)
```

There are many cases in the ESG lexicons where the past participial form or the present participial form of a verb can also be used clearly as an adjective. In such cases we show both an adj sense frame and an inflectional element. Examples:

```
   surprised
      < adj (obj fin inf (p at about by))
      < (veden surprise)
   surprising
      < adj (obj prop)
      < (ving surprise)
```

We decide that e.g. `surprised` has an adjective use because we can say *He was very surprised* or *He was so surprised*. If we give these words `adj` sense frames, then we also have to show the inflectional elements for them, because the ESG morphological analyzer does not analyze a word any further if it finds it in the lexicon.

14 Support word frames

Many nouns have *support verbs* that go with them. Examples are:

make use of
make reference to
take advantage of
take or have a bath
have an argument with
have access to
take account of
make an addition to
make an adjustment to
have admiration for
give advice to
make or have an alliance with
make an amendment to

Most of the semantic content of such a construction resides in the noun. The corresponding support verbs are typically very common verbs like *have*, *make*, *take*, *give*. Which verb goes with the noun in this way is rather idiosyncratic. Often the verb cannot be translated independently into another language. For example, why do we <u>*take*</u> *a bath*? It is not a standard meaning for *take*. Often the combination of support verb plus noun can also be expressed by a single verb, for example:

> *make use of = use*
> *take a bath = bathe*
> *make an amendment to = amend*

When we have a support verb construction, it would be possible to store it under the verb; but this would be unwise because there are very few support verbs, and the entries for each would become tremendous. It is better to store the information under the noun in question. We do this like so for *make use of*.

```
use < ... < sup make v (obj1 (n use)) (p1 of)
```

In general, a support word frame is of the form:

`sup` *SupportWord* *SenseFrameForSupportWord*

In other words, we show the special symbol `sup`, followed by the support word w, and then give a sense frame for w. The system will automatically construct a special sense name, like `make_use`, that shows both the support word and the index word.

We could accomplish some of the same results by writing multiwords like this:

```
=make use of < v obj
```

But there are two problems with this. (1) We do have to make `make` the head word of the multiword, because it is the head word is what gets inflected. But then the multiword will be stored internally under this head word (and will therefore overload `make`). (2) The support word construction allows many variants that the multiword doesn't, like:

> *He made very good use of it.*
> *This is the use that he made of it.*
> *How much use did you say he made of it?*

For all of these, the ESG parser will show the `make_use` sense.

References

[1] Veronica Dahl and Michael C.McCord. Treating coordination in logic grammars. *Computational Linguistics*, 9:69–91, 1983.

[2] Jeffrey S. Gruber. *Studies in Lexical Relations*. PhD thesis, MIT, 1965.

[3] Ray S. Jackendoff. *Semantics and Cognition*. MIT Press, Cambridge, MA, 1983.

[4] Shalom Lappin and Michael C. McCord. Anaphora resolution in Slot Grammar. *Computational Linguistics*, 16:197–212, 1990.

[5] Shalom Lappin and Michael C. McCord. A syntactic filter on pronominal anaphora for Slot Grammar. In *Proceedings of the 28th Annual Meeting of the ACL*, pages 135–142, 1990.

[6] Michael C. McCord. Slot Grammars. *Computational Linguistics*, 6:31–43, 1980.

[7] Michael C. McCord. Using slots and modifiers in logic grammars for natural language. *Artificial Intelligence*, 18:327–367, 1982.

[8] Michael C. McCord. Modular logic grammars. In *Proceedings of the 23rd Annual Meeting of the ACL*, pages 104–117, 1985.

[9] Michael C. McCord. Slot Grammar: A system for simpler construction of practical natural language grammars. In R. Studer, editor, *Natural Language and Logic: International Scientific Symposium, Lecture Notes in Computer Science*, pages 118–145. Springer Verlag, Berlin, 1990.

[10] Michael C. McCord. Heuristics for broad-coverage natural language parsing. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 127–132. Morgan-Kaufmann, 1993.

[11] Michael C. McCord. Word sense disambiguation in a Slot Grammar framework. Technical report, IBM T. J. Watson Research Center, 2004. RC 23397.

[12] Michael C. McCord. A formal system for Slot Grammar. Technical report, IBM T. J. Watson Research Center, 2006. RC 23976.

[13] Michael C. McCord. Using Slot Grammar. Technical report, IBM T. J. Watson Research Center, 2006. RC 23978.

[14] Michael C. McCord and Susanne Wolff. The lexicon and morphology for LMT, a prolog-based MT system. Technical report, IBM T. J. Watson Research Center, 1988. RC 13403.

[15] Adrian Walker, Michael C. McCord, John F. Sowa, and Walter G. Wilson. *Knowledge Systems and Prolog: A Logical Approach to Expert Systems and Natural Language Processing.* Addison-Wesley, Reading, MA, 1987.