The Shortest Possible Introduction to Head-Driven Phrase Structure Grammar

Brent Woo
University of Washington
bwoo@uw.edu

Draft: 8/17/16 . Do not distribute. Do not cite.

1 Intro

Head-Driven Phrase Structure Grammar (HPSG) is a grammar framework. While most undergraduate syntax courses in North America teach syntax using some flavor of X-bar or Minimalist syntax framework, HPSG provides an alternative framework to describe the grammars of natural languages.

This introduction is based heavily on Sag, Wasow, and Bender (2003) (referred to as SWB). Anything in double quotation marks is cited from the book. Any error or misconstrual is mine.

2 Brackets and Revelations

The basic ingredient for HPSG is the **feature structure**, which encodes grammatical information. The description of a feature structure usually takes the form of an **argument-value matrix** (AVM). Conventionally, features are on the left and their respective values are on the right. Values can be atomic values, like *sg* (singular), binary (+ or -), lists; or values can even be other AVMs, giving rise to nested structure.

(1) $$\begin{bmatrix} \text{FEATURE}_1 & \text{value}_1 \\ \vdots & \vdots \\ \text{FEATURE}_n & \text{value}_n \end{bmatrix}$$

A **lexical entry** (a "word" in the "dictionary") is an ordered pair of a *form* and *feature structure description*:

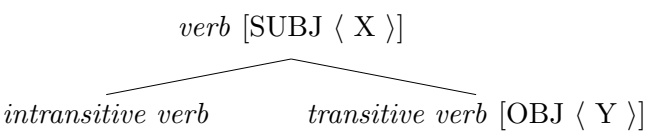
(2) $$\left\langle \text{book} , \begin{bmatrix} \text{POS} & \text{noun} \\ \text{NUM} & \text{sg} \end{bmatrix} \right\rangle$$

which here are sometimes abbreviated using just their form and type:

(3) book [*noun*]

Lexical entries, features and their values, rules, and all other formal constructs are organized in a single **type hierarchy**. For example, consider just transitive verbs and intransitive verbs. They can be organized into a hierarchy like this:

(4)



This is an **inheritance hierarchy**.[1]

(5) Properties of inheritance hierarchies:
   a. Daughters inherit whatever constraints their mothers have.
   b. It must be specified in lexical entries which specific type each word is.

This says, roughly: things of type *verb* must have *at the very least* exactly one thing ("X") on their SUBJ list (lists being denoted by angled brackets ⟨ ⟩). All daughter types will inherit this constraint. It has two daughter types, *intransitive verb* and *transitive verb*. In this hierarchy, *intransitive verb* has no further constraints. *transitive verb*, however, requires exactly one thing ("Y") on its OBJ list, in addition to the constraint inherited from its mother. Lexical entries of type *transitive verb*, ultimately have these two constraints:

(6) $\begin{bmatrix} \text{SUBJ} & \langle \text{X} \rangle \\ \text{OBJ} & \langle \text{Y} \rangle \end{bmatrix}$

Employing a type hierarchy allows for both a highly structured and organized lexicon and brief lexical entries, by identifying cross-cutting generalizations and using inheritance to reduce redundancy.

2.1 Constraints versus generation

With the basics of feature structures down, let's try to generate some sentences. For this section, let's stipulate that:

(7) SUBJ is what linearly precedes the verb and OBJ follows it.

[1] Readers who like computers will be pleased to learn this almost exactly parallels the behavior of inheritance in object-oriented programming languages. Is there multiple inheritance? There can be. See §16.2 in SWB.

I distinguish this here in order to later discard it specifically. In our small grammar, nouns have no distinguishing features so far. Given a lexicon containing just the following (brackets removed when unnecessary) entries, what can we generate?

(8) Lexicon:

a. devours [*transitive verb*]
b. sleeps [*intransitive verb*]
c. rains [*intransitive verb*]
d. Paul [*noun*]
e. Justin [*noun*]
f. crackers [*noun*]

(9) a. Paul sleeps.
b. Paul devours crackers.
c. Justin rains.

It overgenerates. First, let's discuss what went right. In (9a), the verb *sleeps* is of type *intransitive verb*, and according to the type hierarchy in (4), it has the constraint, inherited from its mother type, [SUBJ ⟨ X ⟩], that there must be exactly one thing on its SUBJ list. We met that constraint by inserting the noun *Paul* before it, in our so-called 'subject position'. The sentence is licensed. In (9b), the verb *devours* is of type *transitive verb*, which means it is paired with the feature structure in (6). Its constraints were resolved by putting a noun *Paul* in subject position and a noun *crackers* in object position.

There are many other mistakes in this simple grammar, but let's focus on (9c). Why did this happen? Generative grammar as a general enterprise concerns the description of the infinite system of human language with a finite set of formal tools. If those tools are not sufficiently constrained, they will describe a set of utterances larger than the set considered to comprise the target language: overgeneration. HPSG is no different. We have no other choice but to *constrain*, and the way HPSG does it is by adding constraints in an organized, systematic fashion, represented by a type hierarchy.

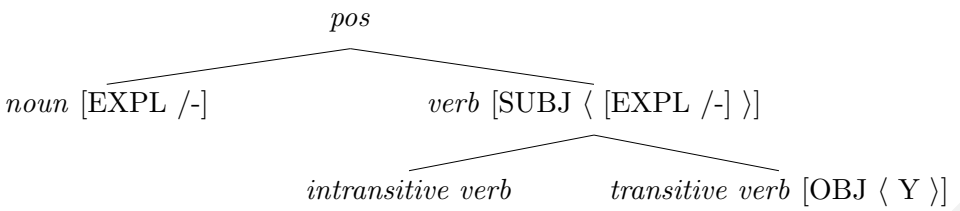
2.2 Grammar design

We have identified a problem with our system: it overgenerates. Informally stated, our goal in this section is:

(10) a. Restrict verbs like *rains* to take only expletive subjects.
b. Other verbs should not take expletive subjects.

This is one possible way to execute (10a). Let's expand the type hierarchy to include nouns, with a new supertype *pos* (parts of speech).

(11)



A new feature and value has appeared. EXPL (expletive) is a binary feature, with only + and - as possible values. The slash character (/) before the value means that the feature is **defeasibly constrained**: by default all lexical items of that type will have that value, but if the lexical item specifies otherwise, then it will be that value. Here, nouns are by default [EXPL -], that is, they are not expletive. Defeasible constraints can be *overridden*.

The hierarchy in (11) also has an added constraint on the *verb* type. What this new constraint says, informally, is that the item in its subject position is defeasibly constrained to be [EXPL -]. This analysis directly reflects the observation that the majority of verbs take non-expletive subjects.

Let's introduce one new lexical item and revise another. Here are the new lexical items:

(12) Lexicon additions:

a. $\left\langle \text{it} , \begin{bmatrix} \textit{noun} \\ \text{EXPL} + \end{bmatrix} \right\rangle$

b. $\left\langle \text{rains} , \begin{bmatrix} \textit{intransitive verb} \\ \text{SUBJ} \left\langle \left[\text{EXPL} + \right] \right\rangle \end{bmatrix} \right\rangle$ (revised (8c))

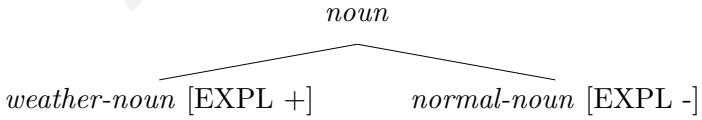
In both of these lexical entries, we override the defeasible constraints. The lexical entry *it* is valued as an expletive noun, and *rains* inherits the defeasible constraint from its mother type *verb*, but overrides the value. Just like other verbs, it needs exactly one thing in its subject position, but it is more needy: it needs that thing to be [EXPL +].

With these additions to the type hierarchy and lexicon, the grammar can now successfully generate:

(13) It rains.

We succeeded in our plan set out in (10a). But there are other strategies to achieve that goal. Why not create a new type in the hierarchy for these words, and split their respective parts of speech into [EXPL +] and [EXPL -] classes, like so?

(14)



This is one possible alternative. There are many others. I offer no persuasion one way or the other in this paper, but I hope this is illustrative of the fact that in building an HPSG grammar, these are the elements of design that must be taken into consideration. It is your choice to either ostracize the weather nouns or simply grant them a special badge.

2.3 A note on the lexicon

Let's look at what the lexicon looks like right now. We use the overall type hierarchy in conjunction with the lexical entries to compute lexical items. We are using the hierarchy in (11). Here are two lexical entries, repeated from above:

(15) a. Justin [*noun*]
b. devours [*transitive verb*]

Lexical entries can be as minimal as specifying the form and type, as these two entries do. The type hierarchy does all the work of fleshing out these entries. Recall the properties of inheritance hierarchies (5). These principles mean that entries like above give rise to lexical sequences like the following for *devours*:

(16) $$\left\langle \text{devours} , \begin{bmatrix} \textit{transitive verb} \\ \text{OBJ} \left\langle \text{Y} \right\rangle \\ \text{SUBJ} \left\langle \left[\text{EXPL -} \right] \right\rangle \end{bmatrix} \right\rangle$$

Understanding how this sequence is computed is critical. *devours* is of type *transitive verb*. This type is specified to have the constraint [OBJ ⟨ Y ⟩]. Its supertype ("mother" type) *verb* has the constraint [SUBJ ⟨ [EXPL /-] ⟩]. All daughters of this type, including *transitive verb*, inherit this constraint. This is how (16) is computed.

Why does (16) have a strict [EXPL -] constraint, but the supertype has a defeasible [EXPL /-] constraint? "Defeasibility" is a notion only accessible by the type hierarchy. Constraints such as this may only be overridden by types or lexical entries.

These lexical sequences are what the syntax has access to, and will serve the function of "words" in our little model here. That is, these "words" are what join together to build syntactic structures.

Theories more advanced than the one in this paper, which is every theory, make use of a far richer lexical computational process, which can involve inflectional rules, *lexemes*, "families of lexical sequences", and so on.

2.4 Unification

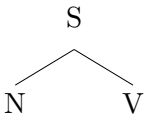
What about the second part of our goal (10b) *Other verbs should not take expletive subjects*, and the problematic sentences like (9c) *Justin rains*? They are already ruled out by the work we've done, by their failure to **unify**.

Let's look at our sentence *It rains* in closer detail. Given the lexical entries, how do they actually "combine" and produce a sentence? The name of the game, or at least three-fourths of it, is *phrase structure grammar*. We do make use of phrase structure rules:

(17) **Subject rule**: S → N V

which means a sentence (S) is composed of a noun (N) followed by a verb (V).[2] Our familiar representation of this is with a tree diagram:

(18)



Finally, this is a phrase structure tree, which will look familiar to the student of X-bar syntax. Before we can insert our lexical items into the tree, here are two more important notational concepts: labels and tags.

Labels, such as S, NP, VP, etc., are designated *abbreviations* for feature structures that meet certain criteria. They are of course abbreviated for human convenience, as parsers and computer systems which employ feature structure grammars don't need them. We can declare that for example, an S is a feature structure that is *saturated*, or complete – its SUBJ and OBJ lists are empty.[3]

(19) $$\text{S} = \begin{bmatrix} \text{SUBJ} \langle\rangle \\ \text{OBJ} \langle\rangle \end{bmatrix}$$

and the other two labels are abbreviations for things of type *noun* and *verb*.

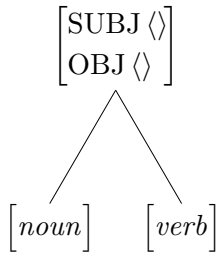
(20) a. N = [*noun*] b. V = [*verb*]

Now going backwards, we can reconstruct our rule using our formalism. Replacing the abbreviations in (18), we get a tree representation of our rule:

[2] An anonymous but happy reviewer asks whether we assume such plain linearization strategies like "to the right of" and "precedes". The answer would appear to be yes. Lists are crucial in this regard, and sophisticated manipulation of them in tandem with tags allow us to describe long-distance dependencies ("wh-movement") and other phenomena. An important conceptual point to be made later is necessary to fully address this here: as a non-derivational formalism, HPSG need make no appeal to notions of "c-command" or other concepts only available throughout the course of a derivation. Moreover, phrase structure rules take on a very different form in developed theories that dispense with the arrow notation and deal exclusively with adding to lists in a linear fashion.

[3] Here is an actual, real-life label abbreviation in *type-description language* (TDL): s-label := label & [SYNSEM.LOCAL [CAT [HEAD verb, VAL [COMPS ¡ ¿, SUBJ ¡ ¿]]], LABEL-NAME "S"].

(21)



And our restated (equivalent!) rule from (17):

(22) **Subject rule**: $\begin{bmatrix}\text{SUBJ } \langle\rangle \\ \text{OBJ } \langle\rangle\end{bmatrix} \rightarrow \begin{bmatrix}noun\end{bmatrix} \quad \begin{bmatrix}verb\end{bmatrix}$

We can finesse this rule even further by adding **tags**. We may observe that, among many other faults, this rule is inadequate for describing clause-subject sentences.

(23) That the chia sprouted stinks. (= It stinks that the chia sprouted)

where what precedes a verb is not a noun, but a clause. We won't be able to describe such sentences in detail, but we can move towards them by generalizing our rule. If the subject is what precedes the verb, and the subject is always needed on verbs (verify this in the type hierarchy in (11)) we can declare an *identity relationship* between the two elements using **tags**, which can graphically take the form of boxed, arbitrary numbers.

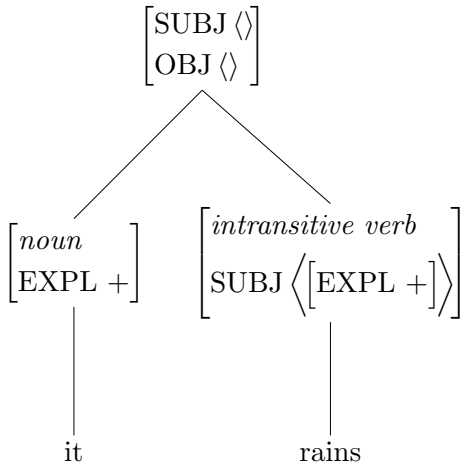
(24) **Subject rule**: $\begin{bmatrix}\text{SUBJ } \langle\rangle \\ \text{OBJ } \langle\rangle\end{bmatrix} \rightarrow \boxed{1} \quad \begin{bmatrix}verb \\ \text{SUBJ } \langle \boxed{1} \rangle\end{bmatrix}$

This rule now says that a sentence (S) is composed of a verb, preceded by exactly what is on its SUBJ list. This is the version of the rule we will use in our grammar here. Rules are also included in the type hierarchy, but require further abstraction.

Remember that rules are stipulations. We've declared that our grammar licenses structures of the type in (24), and declaring that an S has these requirements is, in effect, a hypothesis in our larger theory.

We can now probe into the structure of *It rains*. Using the lexical items from (12), we can insert those words into our rule in (24), in tree format. For appearance purposes the form of words appears as a terminal node and the feature structure associated with that form is the non-branching node immediately dominating it.

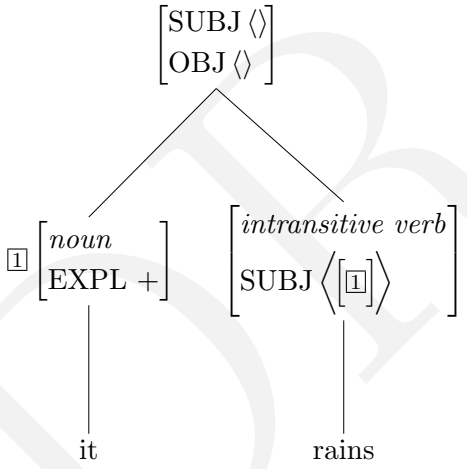
(25)



The tree is "verified", then, if, given the particular lexical items, phrase structure rule(s), and all the constraints on the playing field, everything **unifies**. Constraints may unify if they don't clash, that is, specify conflicting values. A sentence is ruled ungrammatical by the grammar if *any* interacting constraints cannot unify.

Consider the Subject rule in (24). Does it license the structure in (25)? Does the tag describe this structure? The verb is preceded by the thing on its SUBJ list, which is [EXPL +]. *it* is [EXPL +]. *it* is also specified as type *noun*, but this does not destructively clash with any constraint on the verb's SUBJ list; our verbs are indifferent about what type their subjects are. We can illustrate the identity relationship by using tags in trees.

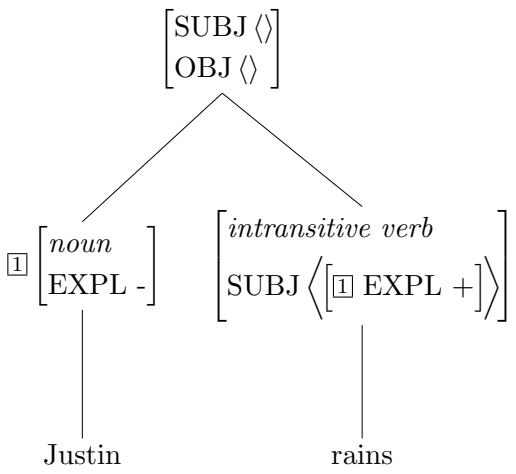
(26)



The *intransitive verb* type is a subtype of *verb*, and so can appear wherever its supertype *verb* can.

Finally for this section, we return to the question asked in the beginning: how is (9c) *Justin rains* ruled out? It won't ever unify. Consider the ill-formed structure:

(27)



The [EXPL -] constraint on *Justin* and the [EXPL +] constraint on *rains* clash, because the Subject rule tag requires that they be identical.

2.5 Non-derivation

Although I frequently employ tree diagrams, keep in mind, in HPSG a central theoretical idea is that there is no "combinatory operation" or in fact, any *process* at all. There is only instantaneous unification. In one fell swoop, we have identified appropriate words (described by the lexical entries), picked our goal phrase structure rule, and evaluated whether or not they all unify and successfully generate the target sentence. Nothing depends on an order of operations, and no proper HPSG theory should be described in narrative prose using the words "and then".

An important conceptual consequence of this is that the tree diagrams should be understood not as bottom-up or top-down, but as simply "there". Computational systems may process a "tree" one way or the other for generation or parsing purposes. But the theory does not make a distinction either way. In sum, HPSG is **non-derivational**.

We haven't touched on wh-questions, topicalization, or other common cases of displacement in grammar, but rest assured HPSG has the means to capture displacement without using transformational operations that manipulate the structure, like movement. In this way, HPSG is also **non-transformational**.

This has important consequences. It's not possible to formulate constraints on the grammar, such as island constraints, "EPP" conditions, or binding, in terms of a pre- or post-movement layer or representation, like D- or S-structure. That is, a theorist working in HPSG must conceive of a way to capture island phenomenon without recourse to a concept like "banning the movement out of an island". This means all constraints on the grammar must be **representational**, and not **derivational**.

In this section I've touched on many of the core concepts involved in building an HPSG grammar. As theories develop, the type hierarchy grows, phrase structure rules are added, and the lexicon is fleshed out. In this way, we keep tight control of every part of our grammar,

from how words are organized to how rules interact and feed each other, and everything is precisely verifiable by whether it all unifies in the end.

3 Keywords

1. **generative grammar**: HPSG sprouts from the same tradition as X-bar and Minimalist syntax, where language is said to be *generated* from a series of formal rules.
2. **highly lexicalist**: Virtually all of the machinery in the language is encoded in information-rich lexical entries. These entries are organized according to a hierarchy to reduce redundancy.
3. **constraint-based**: nearly everything, from the governing principles to the lexical items, is stated in terms of a constraint from something more general to something more specific; ultimately the grammar should converge on the appropriate semantic representation.
4. **non-derivational**: transformational, step-wise operations that manipulate a structure, like "movement", are unavailable.
5. **phonology, syntax, semantics** are encoded.

4 Resources

- Visit the English Resource grammar: `http://erg.delph-in.net/logon` and make some trees of your own!
- DELPH-IN is the most prominent research consortium focusing on developing HPSG models of grammar: `http://www.delph-in.net/wiki/index.php/Home`