ChatScript Pos-Parser Manual
© Bruce Wilcox, gowilcox@gmail.com
Revision 1/11/14 cs4.0

Perspective on State of the Art Pos-tagging and Parsing

Pos-tagging is a strange beast. Back in school we learned the various parts of speech (noun, verb, adjective, adverb, preposition …). They can have a range of 38 possible tags to over a hundred. Pos-taggers define a range of labels for their convenience which can be more or less than what you'd expect. On the more side, they typically separate verbs into their various tenses. On the less side, the standard Penn Treebank system lumps prepositions and subordinate conjunctions into the single label IN. So, following established tradition, I define my own tags for my own convenience.

Nowadays most people treat pos-taggers and parsers as a done deal. Statistical pos taggers like the Stanford parser are the gold standard and on Wall Street Journal type articles it is said to be about 97.3% accurate in tagging. Statistical parsers look at tuples of words to predict what part-of-speech a word will be. And because they do so well, pos-tagging is now separated into a separate phase from parsing. These "good" results are for kinds of material the tagger is trained for and degrades on other works until trained there. They are rarely trained on other kinds of work because the training data isn't there.

ChatScript faces uncommon problems for most Natural Language tools. The best taggers/parsers are trained on the Wall Street Journal, where everything is properly cased. If you pass "i like you" to the Stanford parser, it will tell you that "i" is a foreign word, "like" is a preposition, and "you" is a pronoun. Chatters often never use upper case and speech recognitions devices don't output it either. So ChatScript works to handle all cases of things.

The Stanford Parser (and I pick on it merely as representative of all statistical parsers of which it is considered one of the best) is 97.3% accurate in uniform ways. That is, it is flawed on all kinds of input from simple sentences to complex ones. Simple sentences can defeat it. It can handle *It is on me* but not *Is it on me* which becomes:

is/VBZ it/PRP on/RB me/PRP ?/.

making *on* an adverb instead of a preposition.

And it has weird variances: *Get it off of me* labels *off* as a particle (correctly) but *get the spider off of me* labels *off* as an adverb (differently and wrongly). Or if you say *I saw Dad do it* it correctly labels *Dad* as a proper noun. Same for *Father* and *Mother*. But *Mom* it labels as an ordinary singular noun, even while keeping it capitalized in its output.

Fine. It is strange in places because of word placements. Part of the 97.3%. But 97.3% accuracy in pos-tagging means that out of every 100 words, almost 3 are wrongly tagged. The average sentence length of populist publications like Reader's Digest in 1985 was 20-22 words. Heavy duty publications like the Wall Street Journal or the New York Times averaged around 26-27 words. So presumably one in every four WSJ sentences is wrongly parsed. Even if you pos-tag a sentence perfectly you can wrongly parse it. The stated accuracy of parsing of the Stanford parser is 56%. Almost half of what it reads is parsed incorrectly.

Consider this sentence: *he painted the car purple.* The tagger says: he/PRP painted/VBD the/DT

car/NN purple/NN which means it claims purple is a noun. I can live with that, though one might prefer purple be considered an adjective. The parse, however, becomes problematic. Should it be considered *subect verb indirectobject object* or *subject verb objectcomplement.* Or is it *subject verb directobject* (the object being *car purple*)? Its parse looks just like a parse for *he painted the bank teller.*

So one complaint I have is with accuracy. Another is speed and memory requirements. The simpler of the Stanford parsers is the PCFG one. On an Intel i5 2.53 Mhz machine, a 27 word sentence takes 1.5 seconds and 100 Mb. ChatScript takes under 10 milliseconds and 50Mb with a full dictionary. You can run with a large dictionary in 16Mb for a cellphone.

And a third is that Stanford's will always return a parse, even if the parse is not reasonable. The parse of *is it on me? g*iven that it treats *on* as an adverb instead of a preposition, is garbage and is similar to *is it me?* with an extra adverb tacked on. For other sentences it can't say "I find no verb so either this isn't a sentence or something is wrong". It has no way of knowing that maybe it failed when sometimes one clearly could if one knew grammar.

And fourth, the output is a tree structure, which then needs to be reinterpreted by some other tool.

Hence ChatScript has a built-in pos-tagger/parser using rules of grammar along with statistics. This is the tack taken by *Constraint Grammar* parsers, which can achieve high degrees of pos-tagging accuracy (99.x%) and run rapidly. Their primary flaw is you have to hand code all the rules of grammar (say around 1500 of them) and need a lot of grammatical expertise. And there are no open-source versions of them.

Proper name merging

This gets even messier given that ChatScript will attempt named entity extraction – it will decide some sequences of words represent a single upper-case compound name.

By default ChatScript attempts to detect named entities given as multiple words, and merge them into a single token with underscores. "united states of america" becomes "United_States_of_America". This is not without its hazards. The WORLDDATA ships with a large number of names of works like movies, tv shows, books, etc. These can collide. "Don't lie to me" can see "lie to me" as a possible TV show name. The merging code will ALWAYS merge quoted strings and capitalized sequences, so "Lie to Me" will get merged. It will also merge any sequence whose result is marked with the property ALWAYS_PROPER_NAME_MERGE. Currently world data adds this on to all location names (countries, cities, states) so they are always a single token no matter what case is used. The merging code will not automatically merge the names of TV shows. Nevertheless the sequence will be marked with its appropriate concepts because the system checks sequences of words even if they are not merged. So you would get "lie" "to" "me" "Lie to Me" all marked appropriately and it is up to the script to distinguish.

I personally run script early on that sees if ~NOUN_TITLE_OF_WORK matches. If it does, it determines if it was used in a sentence involving relevant verbs like "I watched lie to me yesterday" or if the user entered it capitalized or quoted or if we were already in the TV topic. If none of those are true, the script erases the ~TV topic mark using ^unmark. This prevents the system from erroneously trying the TV topic. You can also just turn off proper name merging by changing the value of $TOKEN

removing the | #DO_PROPERNAME_MERGE value usually applied, at the cost of having to deal with named sequences yourself in script.

ChatScript Parser

The ChatScript parser runs by default on all input, doing what it can to parse it. If it believes it failed to parse correctly then it will set the tokenflags appropriately. You can test for this like this:
u: (%tokenflags&#FAULTY_PARSE)

You can suppress the parser by setting $token to NOT have the request to parse | #DO_PARSE

If the parser is not run, then the pos-tags of all words will still be partially performed, reducing the set of word interpretations as much as it can without risking losing any correct pos-tag. But you will not get any parse information..

The fragments of a successful parse are retrievable using concepts. The main sentence, because it may not contiguous and is special, does not have a composite retrieval. Instead you can retrieve ~mainsubject, ~mainverb, ~mainindirect, ~maindirect. Things which are contiguous chunks are phrases, clauses, and verbals. If

ChatScript has a complexity limit, and cannot handle sentences near 255 words. Also, sentences containing more than 7 of some particular concept will not mark after the first 7. So a sentence with 8 nouns will not have a ~noun after the 7th. If the nouns were a mixture of singular and plural, then it will represent them, up to the 7 limit.

ChatScript recognizes these structures: the main sentence, prepositional phrases (called phrases), subordinate clauses (called clauses) and various verbal expressions (called verbals) like gerunds, adjective participles, and noun infinitives.

POS TAGS

Words will have pos-tags attached, specififying both generic and specific tag attributes, eg., ~noun and ~noun_singular. In addition to normal generic kinds of pos tags, words which are serving a pos-tag role different from their putative word type are marked as members of the major tag they act as part of. E.g,
~noun_gerund – verb used as a ~noun
~noun_infinitive – verb used as a ~noun
~noun_omitted_adjective – an adjective used as a collective noun (eg the beautiful are kind)
~adjectival_noun (noun used as adjective like bank “bank teller”
~adjective_participle (verb participle used as an adjective)
Auxilliary verbs are segmented into normal ones and special ones. Normal ones give their tense directly. Special ones give their root word.

Generic	**Specific**
~noun	~noun_singular, ~noun_plural, ~noun_proper_singular, ~noun_proper_plural, ~noun_gerund, ~noun_number, ~noun_infinitive. ~noun_omitted_adjective
~verb	~verb_present, ~verb_present_3ps, ~verb_infinitive, ~verb_present_participle, ~verb_past, ~verb_past_participle
~aux_verb	~aux_verb_present, ~aux_verb_past, ~aux_verb_future (~aux_verb_tenses) ~aux_be, ~aux_have, ~aux_do

The tense of the be/have/do verbs can be had via ^properties() and testing for verb tenses
~adjective ~adjective_normal, ~adjective_number, ~adjective_noun, ~adjective_participle
Adjectives in comparative form will also have ~more_form or ~most_form.
~adverb ~adverb_normal
Adverbs in comparative form will also have ~more_form or ~most_form.
~pronoun ~pronoun_subject, ~pronoun_object
~conjunction_bits ~conjunction_coordinate , ~conjunction_subordinate
~determiner_bits ~determiner, ~pronoun_possessive, ~predeterminer,
~possessive (covers ' and 's at end of word)
~to_infinitive ("to" when used before a noun infinitive)
~preposition
~particle (free-floating preposition tied to idiomatic verb)
~comma
~quote (covers ' and " when not embedded in a word)
~paren (covers opening and closing parens)
~foreign_word (some unknown word)
~there_existential (the word there used existentially)

For ~noun_gerund in *I like swimming* the verb gerund *swimming* is treated as a noun (hence called noun-gerund) but retains verb sense when matching keywords tagged with part-of-speech (i.e., it would match swim~v as well as swim~n).

~number is not a part of speech, but is comprise of ~noun_number (a normal number value like 17 or seventeen) and ~adjective_number (also a normal numeral value and also ~placenumber) like *first.*

To can be a preposition or it can be special. When used in the infinitive phrase *To go*, it is marked ~to_infinitive and is followed by ~noun_infinitive.

~verb_infinitive refers to a match on the infinitive form of the verb (I hear John *sing* or I will *sing*).

~There_existential refers to the use of where not involving location, meaning the existence of, as in *There is no future.*

~Particle refers to a preposition piece of a compound verb idiom which allows being separated from the verb. If you say *"I will call off the meeting", call_off* is the composite verb and is a single token. But if you split it as in *"I will call the meeting off*, then there are two tokens. The original form of the verb will be *call and the* canonical form of the verb will be *call_off,* while the free-standing *off* will be labeled ~particle.

~verb_present will be used for normal present verbs not in third person singular like *I walk* and ~verb_present_3ps will be used for things like *he walks*

~possesive refers to *'s* and *'* that indicate possession, while possessive pronouns get their own labeling *~pronoun_possessive*. ~pronoun_subject is a pronoun used as a subject (like *he)* while pronoun_object refers to objective form like (*him)*

Individual words serve roles, which are retrievable. These include
~mainsubject, ~mainverb, ~mainindirect, ~maindirect
~subject2, ~verb2, ~indirectobject2, ~object2

~subject_complement – (adjective object of sentence involving linking verb)
~object_complement – (2ndary noun or infinitive verb filling modifying mainobject or object2)

~conjunct_noun, ~conjunct_verb, ~conjunct_adjective, ~conjunct_adverb
~conjunct_phrase ~conjunct_clause, ~conjunct_sentence

~postnominalAdjective - adjective occuring AFTER the noun it modified
~reflexive - (reflexive pronouns)
~not

~address – noun used as addressee of sentence
~appositive – noun restating and modifying prior noun

~absolutephrase – special phrase describing whole sentence
~omittedtimeprep – modified time word used as phrase but lacking preposition (*Next tuesday* I will go)
~phrase – a prepositional phrase start (except
~clause – a subordinate clause start
~verbal – a verb phrase

Because ChatScript cannot store match position information on sequences of words longer than 5, only the leading word of a clause, phrase, or verbal is marked, using it as a role. You can get the entire sequence using ^getparse() passing it a match variable. That function can also return just interesting roles of the sequence. A subordinate clause is ~clause, a verbal is ~verbal, and a prepositional phrase depends on its type.

~phrase – ordinary phrase headed by a prepositional
~absolutephrase – leading or tailing phrase with preposition omitted
~timephrase – time phrase with preposition omitted