Stanford typed dependencies manual

Marie-Catherine de Marneffe and Christopher D. Manning

September 2008
Revised for the Stanford Parser v. 3.3 in December 2013

1 Introduction

The Stanford typed dependencies representation was designed to provide a simple description of the grammatical relationships in a sentence that can easily be understood and effectively used by people without linguistic expertise who want to extract textual relations. In particular, rather than the phrase structure representations that have long dominated in the computational linguistic community, it represents all sentence relationships uniformly as typed dependency relations. That is, as triples of a relation between pairs of words, such as "the subject of *distributes* is *Bell*." Our experience is that this simple, uniform representation is quite accessible to non-linguists thinking about tasks involving information extraction from text and is effective in relation extraction applications.

Here is an example sentence:

> *Bell, based in Los Angeles, makes and distributes electronic, computer and building products.*

For this sentence, the Stanford Dependencies (SD) representation is:

nsubj(makes-8, Bell-1)
nsubj(distributes-10, Bell-1)
vmod(Bell-1, based-3)
nn(Angeles-6, Los-5)
prep_in(based-3, Angeles-6)
root(ROOT-0, makes-8)
conj_and(makes-8, distributes-10)
amod(products-16, electronic-11)
conj_and(electronic-11, computer-13)
amod(products-16, computer-13)
conj_and(electronic-11, building-15)
amod(products-16, building-15)
dobj(makes-8, products-16)
dobj(distributes-10, products-16)

These dependencies map straightforwardly onto a directed graph representation, in which words in the sentence are nodes in the graph and grammatical relations are edge labels. Figure 1 gives the graph representation for the example sentence above.

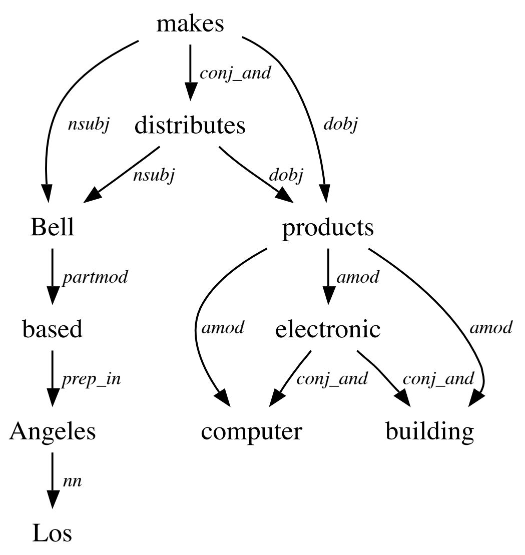


Figure 1: Graphical representation of the Stanford Dependencies for the sentence: *Bell, based in Los Angeles, makes and distributes electronic, computer and building products.*

Document overview: This manual provides documentation about the set of dependencies defined for English. (There is also a Stanford Dependency representation available for Chinese, but it is not further discussed here.) Section 2 of the manual defines the relations and the taxonomic hierarchy over them appears in section 3. This is then followed by a description of the several variant dependency representations available, aimed at different use cases (section 4), some details of the software available for generating Stanford Dependencies (section 5), and references to further discussion and use of the SD representation (section 6).

2 Definitions of the Stanford typed dependencies

The current representation contains approximately 50 grammatical relations (depending slightly on the options discussed in section 4). The dependencies are all binary relations: a grammatical relation holds between a *governor* (also known as a *regent* or a *head*) and a *dependent*. The grammatical relations are defined below, in alphabetical order according to the dependency's abbreviated name (which appears in the parser output). The definitions make use of the Penn Treebank part-of-speech tags and phrasal labels.

***acomp*: adjectival complement**
An adjectival complement of a verb is an adjectival phrase which functions as the complement (like an object of the verb).

"She looks very beautiful"	*acomp*(looks, beautiful)

***advcl*: adverbial clause modifier**
An adverbial clause modifier of a VP or S is a clause modifying the verb (temporal clause, consequence, conditional clause, purpose clause, etc.).

“The accident happened as the night was falling”	*advcl*(happened, falling)
“If you know who did it, you should tell the teacher”	*advcl*(tell, know)
“He talked to him in order to secure the account”	*advcl*(talked, secure)

***advmod*: adverb modifier**
An adverb modifier of a word is a (non-clausal) adverb or adverb-headed phrase that serves to modify the meaning of the word.

“Genetically modified food”	*advmod*(modified, genetically)
“less often”	*advmod*(often, less)

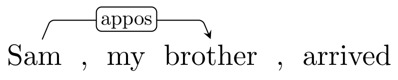
***agent*: agent**
An agent is the complement of a passive verb which is introduced by the preposition “by” and does the action. This relation only appears in the collapsed dependencies, where it can replace *prep_by*, where appropriate. It does not appear in basic dependencies output.

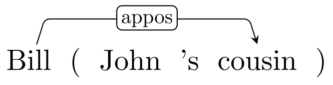
“The man has been killed by the police”	*agent*(killed, police)
“Effects caused by the protein are important”	*agent*(caused, protein)

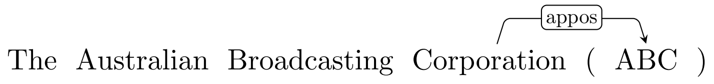
***amod*: adjectival modifier**
An adjectival modifier of an NP is any adjectival phrase that serves to modify the meaning of the NP.

“Sam eats red meat”	*amod*(meat, red)
“Sam took out a 3 million dollar loan”	*amod*(loan, dollar)
“Sam took out a $ 3 million loan”	*amod*(loan, $)

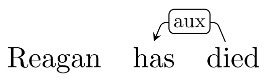
***appos*: appositional modifier**
An appositional modifier of an NP is an NP immediately to the right of the first NP that serves to define or modify that NP. It includes parenthesized examples, as well as defining abbreviations in one of these structures.

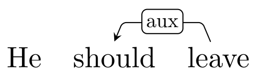






***aux*: auxiliary**
An auxiliary of a clause is a non-main verb of the clause, e.g., a modal auxiliary, or a form of “be”, “do” or “have” in a periphrastic tense.





***auxpass*: passive auxiliary**
A passive auxiliary of a clause is a non-main verb of the clause which contains the passive information.

"Kennedy has been killed"	*auxpass*(killed, been)
	aux(killed,has)
"Kennedy was/got killed"	*auxpass*(killed, was/got)

***cc*: coordination**
A coordination is the relation between an element of a conjunct and the coordinating conjunction word of the conjunct. (Note: different dependency grammars have different treatments of coordination. We take one conjunct of a conjunction (normally the first) as the head of the conjunction.) A conjunction may also appear at the beginning of a sentence. This is also called a cc, and dependent on the root predicate of the sentence.

"Bill is big and honest"	*cc*(big, and)
"They either ski or snowboard"	*cc*(ski, or)
"And then we left."	*cc*(left, And)

***ccomp*: clausal complement**
A clausal complement of a verb or adjective is a dependent clause with an internal subject which functions like an object of the verb, or adjective. Clausal complements for nouns are limited to complement clauses with a subset of nouns like "fact" or "report". We analyze them the same (parallel to the analysis of this class as "content clauses" in Huddleston and Pullum 2002). Such clausal complements are usually finite (though there are occasional remnant English subjunctives).

"He says that you like to swim"	*ccomp*(says, like)
"I am certain that he did it"	*ccomp*(certain, did)
"I admire the fact that you are honest"	*ccomp*(fact, honest)

***conj*: conjunct**
A conjunct is the relation between two elements connected by a coordinating conjunction, such as "and", "or", etc. We treat conjunctions asymmetrically: The head of the relation is the first conjunct and other conjunctions depend on it via the *conj* relation.

"Bill is big and honest"	*conj*(big, honest)
"They either ski or snowboard"	*conj*(ski, snowboard)

***cop*: copula**
A copula is the relation between the complement of a copular verb and the copular verb. (We normally take a copula as a dependent of its complement; see the discussion in section 4.)

"Bill is big"	*cop*(big, is)
"Bill is an honest man"	*cop*(man, is)

***csubj*: clausal subject**
A clausal subject is a clausal syntactic subject of a clause, i.e., the subject is itself a clause. The

governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb. In the two following examples, “what she said” is the subject.

“What she said makes sense”	*csubj*(makes, said)
“What she said is not true”	*csubj*(true, said)

***csubjpass*: clausal passive subject**
A clausal passive subject is a clausal syntactic subject of a passive clause. In the example below, “that she lied” is the subject.

“That she lied was suspected by everyone”	*csubjpass*(suspected, lied)

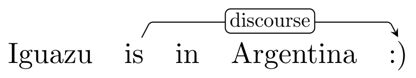
***dep*: dependent**
A dependency is labeled as *dep* when the system is unable to determine a more precise dependency relation between two words. This may be because of a weird grammatical construction, a limitation in the Stanford Dependency conversion software, a parser error, or because of an unresolved long distance dependency.

“Then, as if to show that he could, ...”	*dep*(show, if)

***det*: determiner**
A determiner is the relation between the head of an NP and its determiner.

“The man is here”	*det*(man, the)
“Which book do you prefer?”	*det*(book, which)

***discourse*: discourse element**
This is used for interjections and other discourse particles and elements (which are not clearly linked to the structure of the sentence, except in an expressive way). We generally follow the guidelines of what the Penn Treebanks count as an INTJ. They define this to include: interjections (*oh*, *uh-huh*, *Welcome*), fillers (*um*, *ah*), and discourse markers (*well*, *like*, *actually*, but not *you know*).



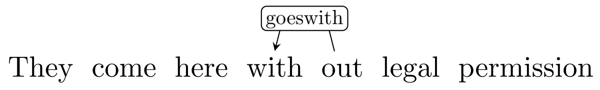
***dobj*: direct object**
The direct object of a VP is the noun phrase which is the (accusative) object of the verb.

“She gave me a raise”	*dobj*(gave, raise)
“They win the lottery”	*dobj*(win, lottery)

***expl*: expletive**
This relation captures an existential “there”. The main verb of the clause is the governor.

"There is a ghost in the room"	*expl*(is, There)

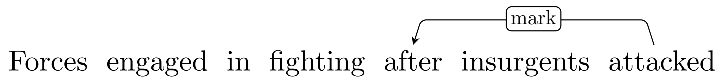
***goeswith*: goes with**
This relation links two parts of a word that are separated in text that is not well edited. We follow the treebank: The GW part is the dependent and the head is in some sense the "main" part, often the second part.

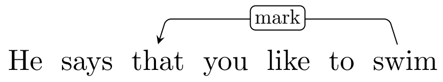


***iobj*: indirect object**
The indirect object of a VP is the noun phrase which is the (dative) object of the verb.

"She gave me a raise"	*iobj*(gave, me)

***mark*: marker**
A marker is the word introducing a finite clause subordinate to another clause. For a complement clause, this will typically be "that" or "whether". For an adverbial clause, the marker is typically a preposition like "while" or "although". The mark is a dependent of the subordinate clause head.





***mwe*: multi-word expression**
The multi-word expression (modifier) relation is used for certain multi-word idioms that behave like a single function word. It is used for a closed set of dependencies between words in common multi-word expressions for which it seems difficult or unclear to assign any other relationships. At present, this relation is used inside the following expressions: *rather than, as well as, instead of, such as, because of, instead of, in addition to, all but, such as, because of, instead of, due to*. The boundaries of this class are unclear; it could grow or shrink a little over time.

"I like dogs as well as cats"	*mwe*(well, as)
	mwe(well, as)
"He cried because of you"	*mwe*(of, because)

***neg*: negation modifier**
The negation modifier is the relation between a negation word and the word it modifies.

"Bill is not a scientist"	*neg*(scientist, not)
"Bill doesn't drive"	*neg*(drive, n't)

***nn*: noun compound modifier**

A noun compound modifier of an NP is any noun that serves to modify the head noun. (Note that in the current system for dependency extraction, all nouns modify the rightmost noun of the NP – there is no intelligent noun compound analysis. This is likely to be fixed once the Penn Treebank represents the branching structure of NPs.)

"Oil price futures"	*nn*(futures, oil)
	nn(futures, price)

***npadvmod*: noun phrase as adverbial modifier**

This relation captures various places where something syntactically a noun phrase (NP) is used as an adverbial modifier in a sentence. These usages include: (i) a measure phrase, which is the relation between the head of an ADJP/ADVP/PP and the head of a measure phrase modifying the ADJP/ADVP; (ii) noun phrases giving an extent inside a VP which are not objects; (iii) financial constructions involving an adverbial or PP-like NP, notably the following construction *$5 a share*, where the second NP means "per share"; (iv) floating reflexives; and (v) certain other absolutive NP constructions. A temporal modifier (tmod) is a subclass of npadvmod which is distinguished as a separate relation.

"The director is 65 years old"	*npadvmod*(old, years)
"6 feet long"	*npadvmod*(long, feet)
"Shares eased a fraction"	*npadvmod*(eased, fraction)
"IBM earned $ 5 a share"	*npadvmod*($, share)
"The silence is itself significant"	*npadvmod*(significant, itself)
"90% of Australians like him, the most of any country"	*npadvmod*(like, most)

***nsubj*: nominal subject**

A nominal subject is a noun phrase which is the syntactic subject of a clause. The governor of this relation might not always be a verb: when the verb is a copular verb, the root of the clause is the complement of the copular verb, which can be an adjective or noun.

"Clinton defeated Dole"	*nsubj*(defeated, Clinton)
"The baby is cute"	*nsubj*(cute, baby)

***nsubjpass*: passive nominal subject**

A passive nominal subject is a noun phrase which is the syntactic subject of a passive clause.

"Dole was defeated by Clinton"	*nsubjpass*(defeated, Dole)

***num*: numeric modifier**

A numeric modifier of a noun is any number phrase that serves to modify the meaning of the noun with a quantity.

"Sam ate 3 sheep"	*num*(sheep, 3)
"Sam spent forty dollars"	*num*(dollars, 40)
"Sam spent $ 40"	*num*($, 40)

***number*: element of compound number**

An element of compound number is a part of a number phrase or currency amount. We regard a number as a specialized kind of multi-word expression.

"I have four thousand sheep"	*number*(thousand, four)
"I lost $ 3.2 billion"	*number*(billion, 3.2)

***parataxis*: parataxis**

The parataxis relation (from Greek for "place side by side") is a relation between the main verb of a clause and other sentential elements, such as a sentential parenthetical, a clause after a ":" or a ";", or two sentences placed side by side without any explicit coordination or subordination.

"The guy, John said, left early in the morning"	*parataxis*(left, said)
"Let's face it we're annoyed"	*parataxis*(Let, annoyed)

***pcomp*: prepositional complement**

This is used when the complement of a preposition is a clause or prepositional phrase (or occasionally, an adverbial phrase). The prepositional complement of a preposition is the head of a clause following the preposition, or the preposition head of the following PP.

"We have no information on whether users are at risk"	*pcomp*(on, are)
"They heard about you missing classes"	*pcomp*(about, missing)

***pobj*: object of a preposition**

The object of a preposition is the head of a noun phrase following the preposition, or the adverbs "here" and "there". (The preposition in turn may be modifying a noun, verb, etc.) Unlike the Penn Treebank, we here define cases of VBG quasi-prepositions like "including", "concerning", etc. as instances of *pobj*. (The preposition can be tagged a FW for "pace", "versus", etc. It can also be called a CC – but we don't currently handle that and would need to distinguish from conjoined prepositions.) In the case of preposition stranding, the object can precede the preposition (e.g., "What does CPR stand for?").

"I sat on the chair"	*pobj*(on, chair)

***poss*: possession modifier**

The possession modifier relation holds between the head of an NP and its possessive determiner, or a genitive 's complement.

"their offices"	*poss*(offices, their)
"Bill's clothes"	*poss*(clothes, Bill)

***possessive*: possessive modifier**

The possessive modifier relation appears between the head of an NP and the genitive 's.

“Bill’s clothes”	*possessive*(John, ’s)

***preconj*: preconjunct**
A preconjunct is the relation between the head of an NP and a word that appears at the beginning bracketing a conjunction (and puts emphasis on it), such as “either”, “both”, “neither”).

“Both the boys and the girls are here”	*preconj*(boys, both)

***predet*: predeterminer**
A predeterminer is the relation between the head of an NP and a word that precedes and modifies the meaning of the NP determiner.

“All the boys are here”	*predet*(boys, all)

***prep*: prepositional modifier**
A prepositional modifier of a verb, adjective, or noun is any prepositional phrase that serves to modify the meaning of the verb, adjective, noun, or even another prepositon. In the collapsed representation, this is used only for prepositions with NP complements.

“I saw a cat in a hat”	*prep*(cat, in)
“I saw a cat with a telescope”	*prep*(saw, with)
“He is responsible for meals”	*prep*(responsible, for)

***prepc*: prepositional clausal modifier**
In the collapsed representation (see section 4), a prepositional clausal modifier of a verb, adjective, or noun is a clause introduced by a preposition which serves to modify the meaning of the verb, adjective, or noun.

“He purchased it without paying a premium”	*prepc_without*(purchased, paying)

***prt*: phrasal verb particle**
The phrasal verb particle relation identifies a phrasal verb, and holds between the verb and its particle.

“They shut down the station”	*prt*(shut, down)

***punct*: punctuation**
This is used for any piece of punctuation in a clause, if punctuation is being retained in the typed dependencies. By default, punctuation is not retained in the output.

“Go home!”	*punct*(Go, !)

***quantmod*: quantifier phrase modifier**

A quantifier modifier is an element modifying the head of a QP constituent. (These are modifiers in complex numeric quantifiers, not other types of "quantification". Quantifiers like "all" become det.)

"About 200 people came to the party"	*quantmod*(200, About)

***rcmod*: relative clause modifier**

A relative clause modifier of an NP is a relative clause modifying the NP. The relation points from the head noun of the NP to the head of the relative clause, normally a verb.

"I saw the man you love"	*rcmod*(man, love)
"I saw the book which you bought"	*rcmod*(book,bought)

***ref*: referent**

A referent of the head of an NP is the relative word introducing the relative clause modifying the NP.

"I saw the book which you bought"	*ref*(book, which)

***root*: root**

The root grammatical relation points to the root of the sentence. A fake node "ROOT" is used as the governor. The ROOT node is indexed with "0", since the indexation of real words in the sentence starts at 1.

"I love French fries."	*root*(ROOT, love)
"Bill is an honest man"	*root*(ROOT, man)

***tmod*: temporal modifier**

A temporal modifier (of a VP, NP, or an ADJP is a bare noun phrase constituent that serves to modify the meaning of the constituent by specifying a time. (Other temporal modifiers are prepositional phrases and are introduced as prep.)

"Last night, I swam in the pool"	*tmod*(swam, night)

***vmod*: reduced non-finite verbal modifier**

A reduced non-finite verbal modifier is a participial or infinitive form of a verb heading a phrase (which may have some arguments, roughly like a VP). These are used to modify the meaning of an NP or another verb. They are not core arguments of a verb or full finite relative clauses.

"Points to establish are . . . "	*vmod*(points, establish)
"I don't have anything to say to you"	*vmod*(anything, say)
"Truffles picked during the spring are tasty"	*vmod*(truffles, picked)
"Bill tried to shoot, demonstrating his incompetence"	*vmod*(shoot, demonstrating)

***xcomp*: open clausal complement**

An open clausal complement (*xcomp*) of a verb or an adjective is a predicative or clausal complement without its own subject. The reference of the subject is necessarily determined by an argument external to the xcomp (normally by the object of the next higher clause, if there is one, or else by the subject of the next higher clause. These complements are always non-finite, and they are complements (arguments of the higher verb or adjective) rather than adjuncts/modifiers, such as a purpose clause. The name *xcomp* is borrowed from Lexical-Functional Grammar.

"He says that you like to swim"	*xcomp*(like, swim)
"I am ready to leave"	*xcomp*(ready, leave)
"Sue asked George to respond to her offer"	*xcomp*(ask, respond)
"I consider him a fool"	*xcomp*(consider, fool)
"I consider him honest"	*xcomp*(consider, honest)

***xsubj*: controlling subject**

A controlling subject is the relation between the head of a open clausal complement (*xcomp*) and the external subject of that clause. This is an additional dependency, not a basic depedency.

"Tom likes to eat fish"	*xsubj*(eat, Tom)

3 Hierarchy of typed dependencies

The grammatical relations defined in the above section stand in a hierarchy. The most generic grammatical relation, dependent (*dep*), will be used when a more precise relation in the hierarchy does not exist or cannot be retrieved by the system.

- *root* - root
- *dep* - dependent
  - *aux* - auxiliary
    - *auxpass* - passive auxiliary
    - *cop* - copula
  - *arg* - argument
    - *agent* - agent
    - *comp* - complement
      - *acomp* - adjectival complement
      - *ccomp* - clausal complement with internal subject
      - *xcomp* - clausal complement with external subject
      - *obj* - object
        - *dobj* - direct object
        - *iobj* - indirect object
        - *pobj* - object of preposition
    - *subj* - subject
      - *nsubj* - nominal subject
        - *nsubjpass* - passive nominal subject

    - *csubj* - clausal subject
      - *csubjpass* - passive clausal subject
- *cc* - coordination
- *conj* - conjunct
- *expl* - expletive (expletive “there”)
- *mod* - modifier
  - *amod* - adjectival modifier
  - *appos* - appositional modifier
  - *advcl* - adverbial clause modifier
  - *det* - determiner
  - *predet* - predeterminer
  - *preconj* - preconjunct
  - *vmod* - reduced, non-finite verbal modifier
  - *mwe* - multi-word expression modifier
    - *mark* - marker (word introducing an *advcl* or *ccomp*
  - *advmod* - adverbial modifier
    - *neg* - negation modifier
  - *rcmod* - relative clause modifier
  - *quantmod* - quantifier modifier
  - *nn* - noun compound modifier
  - *npadvmod* - noun phrase adverbial modifier
    - *tmod* - temporal modifier
  - *num* - numeric modifier
  - *number* - element of compound number
  - *prep* - prepositional modifier
  - *poss* - possession modifier
  - *possessive* - possessive modifier (*’s*)
  - *prt* - phrasal verb particle
- *parataxis* - parataxis
- *punct* - punctuation
- *ref* - referent
- *sdep* - semantic dependent
  - *xsubj* - controlling subject

4 Different styles of dependency representation

Five variants of the typed dependency representation are available in the dependency extraction system provided with the Stanford parser. The representations follow the same format. In the plain text format, a dependency is written as *abbreviated_relation_name*(governor, dependent) where the governor and the dependent are words in the sentence to which a number indicating the position of the word in the sentence is appended.[1] The parser also provides an XML format which captures the same information. The differences between the five formats are that they

[1]In some cases, an apostrophe is added after the word position number: see section 4.6 for more details.

range from a more surface-oriented representation, where each token appears as a dependent in a tree, to a more semantically interpreted representation where certain word relationships, such as prepositions, are represented as dependencies, and the set of dependencies becomes a possibly cyclic graph.

4.1 Basic

The basic typed dependencies use the dependencies defined in section 2, and form a tree structure. That is, there are no crossing dependencies. This is also referred to as a *projective* dependency structure. Each word in the sentence (except the head of the sentence) is the dependent of one other word. For the sentence, "Bell, a company which is based in LA, makes and distributes computer products.", the basic typed dependencies will be:

```
nsubj(makes-11, Bell-1)
det(company-4, a-3)
appos(Bell-1, company-4)
nsubjpass(based-7, which-5)
auxpass(based-7, is-6)
rcmod(company-4, based-7)
prep(based-7, in-8)
pobj(in-8, LA-9)
root(ROOT-0, makes-11)
cc(makes-11, and-12)
conj(makes-11, distributes-13)
nn(products-15, computer-14)
dobj(makes-11, products-15)
```

4.2 Collapsed dependencies

In the collapsed representation, dependencies involving prepositions, conjuncts, as well as information about the referent of relative clauses are collapsed to get direct dependencies between content words. This "collapsing" is often useful in simplifying patterns in relation extraction applications. For instance, the dependencies involving the preposition "in" in the above example will be collapsed into one single relation:

```
   prep(based-7, in-8)
   pobj(in-8, LA-9)
```

will become

```
   prep_in(based-7, LA-9)
```

Moreover, additional dependencies are considered, even ones that break the tree structure (turning the dependency structure into a *directed graph*) as well as non-projective dependencies. So in the above example, the following relation will be added:

```
   ref(company-4, which-5)
```

That relation does not appear in the basic representation since it creates a cycle with the `rcmod` and `nsubjpass` relations. Relations that break the tree structure are the ones taking into account elements from relative clauses and their antecedents (as shown in this example), the controlling (*xsubj*) relations, and the (*pobj*) relation in the case of preposition stranding.

according to	as per	compared to	instead of	preparatory to
across from	as to	compared with	irrespective of	previous to
ahead of	aside from	due to	next to	prior to
along with	away from	depending on	near to	pursuant to
alongside of	based on	except for	off of	regardless of
apart from	because of	exclusive of	out of	subsequent to
as for	close by	contrary to	outside of	such as
as from	close to	followed by	owing to	thanks to
as of	contrary to	inside of	preliminary to	together with

Table 1: List of two-word prepositions that the system can collapse.

English has some very common multi-word constructions that function like prepositions. These are also collapsed as prepositional relations. At the moment, the system handles the multi-word prepositions listed in Tables 1 and 2.

The same happens for dependencies involving conjunction:

```
   cc(makes-11, and-12)
   conj(makes-11, distributes-13)
```

become

```
   conj_and(makes-11, distributes-13)
```

A few variant conjunctions for "and (not)" are collapsed together in this representation as shown in Table 3.

The information about the antecedent of the relative clause (`ref(company-4, which-5)`) will serve to expand the following dependency:

```
   nsubjpass(based-7, which-5)
```

becomes

```
   nsubjpass(based-7, company-4)
```

In the end the collapsed dependencies that the system gives you for the sentence are:

```
nsubj(makes-11, Bell-1)
det(company-4, a-3)
appos(Bell-1, company-4)
nsubjpass(based-7, company-4)
auxpass(based-7, is-6)
rcmod(company-4, based-7)
prep_in(based-7, LA-9)
root(ROOT-0, makes-11)
conj_and(makes-11, distributes-13)
nn(products-15, computer-14)
dobj(makes-11, products-15)
```

by means of	in case of	in place of	on behalf of	with respect to
in accordance with	in front of	in spite of	on top of	
in addition to	in lieu of	on account of	with regard to	

Table 2: List of three-word prepositions that the system can collapse.

Mapped to				
conj_and	as well as	not to mention	but also	&
conj_negcc	but not	instead of	rather than	but rather

Table 3: Mapping of select conjunct relations in the collapsed representation.

4.3 Collapsed dependencies with propagation of conjunct dependencies

When there is a conjunction, you can also get propagation of the dependencies involving the conjuncts. In the sentence here, this propagation should add two dependencies to the collapsed representation; due to the conjunction between the verbs "makes" and "distributes", the subject and object relations that exist on the first conjunct ("makes") should be propagated to the second conjunct ("distributes"):

```
nsubj(distributes-13, Bell-1)
dobj(distributes-13, products-15)
```

However, at present, our converter handles this imperfectly and only generates the first of these two dependencies (in general, it is hard to determine if object dependencies should be distributed or not in English).

Since this representation is an extension of the collapsed dependencies, it does not guarantee a tree structure.

4.4 Collapsed dependencies preserving a tree structure

In this representation, dependencies which do not preserve the tree structure are omitted. As explained above, this concerns relations between elements of a relative clause and its antecedent, as well as the controlling subject relation (*xsubj*), and the object of preposition (*pobj*) in the case of preposition stranding. This also does not allow propagation of conjunct dependencies. In our example, the dependencies in this representation are actually identical to the ones in the collapsed representation:

```
nsubj(makes-11, Bell-1)
det(company-4, a-3)
appos(Bell-1, company-4)
nsubjpass(based-7, which-5)
auxpass(based-7, is-6)
rcmod(company-4, based-7)
prep_in(based-7, LA-9)
root(ROOT-0, makes-11)
conj_and(makes-11, distributes-13)
nn(products-15, computer-14)
```

```
dobj(makes-11, products-15)
```

4.5 Non-collapsed dependencies

This representation gives the basic dependencies as well as the extra ones (which break the tree structure), without any collapsing or propagation of conjuncts. There are options to get the extra dependencies separated from the basic dependencies (see section 5). At print time, the dependencies in this representation can thus look as follows:

```
nsubj(makes-11, Bell-1)
det(company-4, a-3)
appos(Bell-1, company-4)
nsubjpass(based-7, which-5)
auxpass(based-7, is-6)
rcmod(company-4, based-7)
prep(based-7, in-8)
pobj(in-8, LA-9)
root(ROOT-0, makes-11)
cc(makes-11, and-12)
conj(makes-11, distributes-13)
nn(products-15, computer-14)
dobj(makes-11, products-15)
======
ref(company-4, which-5)
```

4.6 Alteration of the sentence semantics

In some cases, collapsing relations introduces a slight alteration of the semantics of the sentence. In all the representation styles involving collapsing, the two following phenomena may appear.

Introduction of copy nodes marked with an apostrophe. A copy node will be introduced in the case of PP conjunction as in "Bill went over the river and through the woods". In this example, the two prepositions "over" and "through" are conjoined and governed by the verb "went". To avoid disjoint subgraphs when collapsing the relations (preposition and conjunction), sentences like this are transformed into VP coordination, which requires making a copy of the word "went". A copy node will be marked with one or more apostrophes in the plain text output or by a `copy` attribute in the XML output. This gives the following representation, which corresponds to a sentence like "Bill went over the river and went through the woods":

prep_over(went-2, river-5)
prep_through(went-2', woods-10)
conj_and(went-2, went-2')

Distortion in governors of preposition modifiers. Another instance where collapsing sacrifices some linguistic fidelity is the case of preposition modifiers. When turning the preposition into a relation, the preposition does not appear as a word of the sentence anymore. Therefore preposition modifiers become dependent on the head of the clause in which they appear, and not on the preposition itself. In *He left his office just before lunch time*, *just* will be an adverbial modifier of the verb *left*. This induces some distortion in the exact semantics of the sentence.

4.7 The treatment of copula verbs

The design philosophy of SD has been to maximize dependencies between content words, and so we normally regard a copula verb like *be* as an auxiliary modifier, even when its complement is an adjective or predicative noun (see the references in section 6 for more discussion and motivation). However, some people do not like this because then the head of some sentences is no longer a verb. In the dependency conversion software, you can ask for the copula to remain the head when its complement is an adjective or noun by giving the flag `-makeCopulaHead`. Uses of the verb *be* as in auxiliary in passives and progressives will still be treated as a non-head auxiliary.

4.8 Comparison of the representation styles

To facilitate comparison, the table below shows the dependencies for the four variants for the example sentence "Bell, a company which is based in LA, makes and distributes computer products". The "non-collapsed" variant (see section 4.5) contains all the relations in the "basic" variant plus one extra dependency: ref(company-4, which-5).

basic	**collapsed**	**propagation**	**collapsed tree**
nsubj(makes, Bell)	nsubj(makes, Bell)	nsubj(makes, Bell)	nsubj(makes, Bell)
		nsubj(distributes, Bell)	
det(company, a)	det(company, a)	det(company, a)	det(company, a)
appos(Bell, company)	appos(Bell, company)	appos(Bell, company)	appos(Bell, company)
nsubjpass(based, which)	nsubjpass(based, company)	nsubjpass(based, company)	nsubjpass(based, which)
auxpass(based, is)	auxpass(based, is)	auxpass(based, is)	auxpass(based, is-)
rcmod(company, based)	rcmod(company, based)	rcmod(company, based)	rcmod(company, based)
prep(based, in)			
	prep_in(based, LA)	prep_in(based, LA)	prep_in(based, LA)
pobj(in, LA)			
root(ROOT, makes)	root(ROOT, makes)	root(ROOT, makes)	root(ROOT, makes)
cc(makes, and)			
	conj_and(makes, distributes)	conj_and(makes, distributes)	conj_and(makes, distributes)
conj(makes, distributes)			
nn(products, computer)	nn(products, computer)	nn(products, computer)	nn(products, computer)
dobj(makes, products)	dobj(makes, products)	dobj(makes, products)	dobj(makes, products)

4.9 Graph-theoretic properties

Dependency syntax representations are naturally thought of as "directed graphs", but some of the precise formal properties of Stanford dependencies graphs can surprise people, so here we summarize the main graph-theoretic properties. The unusual properties are all things that occur with relative clauses. A summary of the properties is shown in table 4. To cover the collapsed representations, you need what is commonly referred to as a labeled, directed multigraph.

The collapsed and CCprocessed dependencies are not a DAG. The graphs can contain small cycles between two nodes (only). These don't seem eliminable given the current representational choices. They occur with relative clauses such as *the woman who introduced you*. The cycles occur once you wish to represent the referent of *who*. In the basic plus extras representation,

	basic	collapsed	CCprocessed	collapsed tree	basic plus extras
Connected?	Yes	Yes	Yes	Yes	Yes
All tokens are nodes?	Yes	No	No	No	Yes
Rooted?	Yes	Yes	Yes	Yes	Yes
Acyclic	Yes	No	No	Yes	Yes
Multigraph	No	No	No	No	Yes
Tree	Yes	No	No	Yes	No
Self-loops?	No	No	No	No	No

Table 4: Graph-theoretic properties of different versions of SD graphs.

you get *rcmod*(woman, introduced), *nsubj*(introduced, who), and *ref*(woman, who).[2] In the collapsing process, *ref* arcs are collapsed, and so there is then a two node cycle: *rcmod*(woman, introduced) and *nsubj*(introduced, woman). These cycles can occur at the "top" of the graph when an NP is the head of the sentence, given the treatment of copula verbs (as in *She is the woman who introduced me.*). This used to mean that the dependency graph didn't have a clear root. This was fixed after version 1.6.8 by explicitly adding a *root* arc to the representation.

There can be multiple arcs with the same label from a node. For instance, this occurs when a noun has several adjective modifiers, each of which gets an *amod* relation, as in *its third consecutive monthly decline.*

In the basic plus extras representation, a word can be the dependent of two different words. For example, a relative word will be a *ref* of the head of the noun phrase it modifies and will have a role in the relative clause. For example you might get both the arcs *ref*(researchers-5, who-6) and *nsubj*(studied-7, who-6). You can even get two arcs between the same pair of words, though these normally result from bugs in the converter.

All graphs should be connected (if it's not, it's a bug!). There are no self-loops in the graphs.

5 In practice

In practice, two classes can be used to get the typed dependencies of a sentence using the code in the Stanford parser (downloadable at http://nlp.stanford.edu/software/lex-parser.shtml).

⋆ edu.stanford.nlp.parser.lexparser.LexicalizedParser

If you need to parse texts and want to get different formatting options for the parse tree, you should use this class. To get the dependencies, add `typedDependencies` in the `-outputFormat` option. By default, this will give you collapsed dependencies with propagation of conjunct dependencies. If you want another representation, specify it in the `-outputFormatOptions` using the following commands according to the type of dependency representation you want:

`basicDependencies` Basic dependencies.

`collapsedDependencies` Collapsed dependencies (not necessarily a tree structure).

[2]Arguably, that third dependency should already have been represented the other way around as *ref*(who, woman), giving a three node cycle, but it wasn't.

`CCPropagatedDependencies` Collapsed dependencies with propagation of conjunct dependencies (not necessarily a tree structure). [This representation is the default, if no option is specified.]

`treeDependencies` Collapsed dependencies that preserve a tree structure.

`nonCollapsedDependencies` Non-collapsed dependencies: basic dependencies as well as the extra ones which do not preserve a tree structure.

`nonCollapsedDependenciesSeparated` Non-collapsed dependencies where the basic dependencies are separated from the extra ones (by "======").

You should also use the `-retainTmpSubcategories` option to get best performance in recognizing temporal dependencies. In the following command, `file.txt` contains your input sentences. (With this command-line, the parser will attempt to tokenize and sentence-break them. There are options to the parser to specify that this has already been done.) The `penn` option will also give you the context-free phrase structure grammar representation of the sentences.

Command line example:
```
java -mx200m edu.stanford.nlp.parser.lexparser.LexicalizedParser
-retainTmpSubcategories -outputFormat "penn,typedDependencies"
-outputFormatOptions "basicDependencies" englishPCFG.ser.gz file.txt
```

Java example:
```
LexicalizedParser lp = LexicalizedParser.loadModel(
                    "edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz",
                    "-maxLength", "80", "-retainTmpSubcategories");
TreebankLanguagePack tlp = new PennTreebankLanguagePack();
GrammaticalStructureFactory gsf = tlp.grammaticalStructureFactory();

String[] sent =  "This", "is", "an", "easy", "sentence", "." ;
Tree parse = lp.apply(Sentence.toWordList(sent));
GrammaticalStructure gs = gsf.newGrammaticalStructure(parse);
Collection⟨TypedDependency⟩ tdl = gs.typedDependenciesCCprocessed();
System.out.println(tdl);
```

⋆ edu.stanford.nlp.trees.EnglishGrammaticalStructure

If you already have Penn treebank-style trees (whether hand-annotated or as output from another parser), you can use this class to get the Stanford dependencies.

Command-line usage. Use the `-treeFile` option as shown in the command line example below. The options to get the different types of representation are as follows:

`-basic`	basic dependencies
`-collapsed`	collapsed dependencies (not necessarily a tree structure)
`-CCprocessed`	collapsed dependencies with propagation of conjunct dependencies (not necessarily a tree structure)
`-collapsedTree`	collapsed dependencies that preserve a tree structure

`-nonCollapsed`	non-collapsed dependencies: basic dependencies as well as the extra ones which do not preserve a tree structure
`-conllx`	dependencies printed out in CoNLL X (CoNLL 2006) format

If you want the non-collapsed version of the dependencies where the basic ones are separated from the extra ones, add the flag `-extraSep`. This will print the basic dependencies, a separator (======) and the extra dependencies. By default, punctuation dependencies are not printed. If you want them, give the option `-keepPunct`.

Command line example:

```
java edu.stanford.nlp.trees.EnglishGrammaticalStructure -treeFile
file.tree -collapsedTree -CCprocessed -keepPunct
```

By default, the CoNLL format retains punctuation. When the CoNLL format is used with collapsed dependencies, words of the sentences which have been collapsed into the grammatical relations (such as prepositions and conjunctions) still appear in the list of words but are given an "erased" grammatical relation:

```
1   Bell         _  NNP  NNP  _  11  nsubj      _  _
2   ,            _  ,    ,    _  1   punct      _  _
3   a            _  DT   DT   _  4   det        _  _
4   company      _  NN   NN   _  7   nsubjpass  _  _
5   which        _  WDT  WDT  _  0   erased     _  _
6   is           _  VBZ  VBZ  _  7   auxpass    _  _
7   based        _  VBN  VBN  _  4   rcmod      _  _
8   in           _  IN   IN   _  0   erased     _  _
9   LA           _  NNP  NNP  _  7   prep_in    _  _
10  ,            _  ,    ,    _  1   punct      _  _
11  makes        _  VBZ  VBZ  _  0   root       _  _
12  and          _  CC   CC   _  0   erased     _  _
13  distributes  _  VBZ  VBZ  _  11  conj_and   _  _
14  computer     _  NN   NN   _  15  nn         _  _
15  products     _  NNS  NNS  _  11  dobj       _  _
16  .            _  .    .    _  11  punct      _  _
```

This class can read files that contain Stanford dependencies in the CoNLL format (i.e., the basic Stanford dependencies), and transform them into another representation (e.g., the CCprocessed representation). To do this, you need to pass the input file using the option `-conllxFile`.

You can also use this class to parse texts, but the input has to be formatted as strictly one sentence per line, and you will not be able to specify options for the parse tree output on the command line. You will only be able to specify the type of the dependencies. Use the option `-sentFile` instead of `-treeFile`. You will need to specify the parser file using the `-parserFile` option. You can print the parse tree by using the `-parseTree` option.

Command line example:

```
java -mx100m edu.stanford.nlp.trees.EnglishGrammaticalStructure
```

```
-sentFile file.txt -collapsedTree -CCprocessed -parseTree -parserFile
englishPCFG.ser.gz
```

API usage. Basic API usage was already illustrated in the `LexicalizedParser` usage above. If you have a Tree object, the steps for converting it to dependencies are like this:

```
 // One time setup
TreebankLanguagePack tlp = new PennTreebankLanguagePack();
GrammaticalStructureFactory gsf = tlp.grammaticalStructureFactory();
// For each Tree
Tree parseTree; // assumed to come from a treebank or parser
GrammaticalStructure gs = gsf.newGrammaticalStructure(parse);
Collection⟨TypedDependency⟩ tdl = gs.typedDependencies();
```

The `PennTreebankLanguagePack` vends an `EnglishGrammaticalStructureFactory`. The only common option to pass in when creating one is a punctuation filter. Pass in a `Filters.⟨String⟩ acceptFilter()` to keep punctuation dependencies. A `GrammaticalStructure` is created for each `Tree`. The methods on a `GrammaticalStructure` for each kind of dependencies is as follows:

basic	`gs.typedDependencies()`
nonCollapsed	`gs.allTypedDependencies()`
collapsed	`gs.typedDependenciesCollapsed(true)`
CCPropagated	`gs.typedDependenciesCCprocessed()`
tree	`gs.typedDependenciesCollapsedTree()`

⋆ GrammarScope

Bernard Bou has written GrammarScope, a GUI interface to the Stanford Dependencies representation, which allows not only viewing dependencies, but altering their definitions. This is a separate download. It is available at: http://grammarscope.sourceforge.net/.

⋆ Other parsers

A number of dependency parsers have now been trained to parse directly to the basic Stanford Dependencies, including MaltParser, DeSR, and MSTParser. Several of these parsers distribute models trained for Stanford Dependencies parsing, including MaltParser and the Easy First Parser. If desired, these parses can then be postprocessed to the collapsed or CCprocessed representation using the `-conllxFile` option of `EnglishGrammaticalStructure`, as discussed above.

Any Penn Treebank constituency parser can be used to produce Stanford Dependencies by using our conversion tool to convert the output of other constituency parsers to the Stanford Dependencies representation. For more information on other parser options, see:

http://nlp.stanford.edu/software/stanford-dependencies.shtml

6 Further references for Stanford Dependencies

The Stanford Dependencies representation was first made available in the 2005 version of the Stanford Parser. Subsequent releases have provided some refinements to and corrections of the relations defined in the original release. The initial written presentation was (?). A more thorough discussion of the motivations behind the design of the representation appears in (?).

The SD representation has seen considerable use within the biomedical text mining community. It has been used to give a task relevant evaluation scheme for parsers (??) and as a representation for relation extraction (??????????????). ? develops a version of the BioInfer corpus annotated with (a slight variant of) the SD scheme. It is available for download at http://mars.cs.utu.fi/BioInfer/. A small amount of SD gold standard annotated data was separately prepared for the Parser Evaluation Shared Task of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation (http://lingo.stanford.edu/events/08/pe/) and is discussed in (?). This data is available from the Stanford Dependencies page: http://nlp.stanford.edu/software/stanford-dependencies.shtml, but the BioInfer corpus is the main source of gold-standard SD data which is currently available. In the recent BioNLP 2009 Shared Task, many of the leading teams built their relation extraction systems over the Stanford Dependency representation (?). It was used by the teams that came 1st, 3rd, 4th, and 5th in Task 1, by the team who came first in Task 2, and by the teams who came 1st and 2nd in Task 3. In the BioNLP 2011 shared task, every team used it (?).

The SD representation has also been used in other domains. It is a common representation for extracting opinions, sentiment, and relations (????????????), as well as specific information (such as event, time or dialogue acts) (???). The tool has been consistently used by several groups in the PASCAL/NIST challenges targeting textual entailment (???????????). It is also used for a variety of other tasks, such as unsupervised semantic parsing (?), coreference resolution, disagreement detection and word sense induction (???), as well as being part of the preprocessing for machine translation systems by several groups (???).

The Stanford dependency representation has also served as a model for developing dependency schemes in other languages. Recently schemes based on the Stanford dependency representation have been proposed for Finnish (??), Thai (?), Persian (?), and French (?). Indeed, there has been increasing interest in defining a consistent cross-linguistic set of relations suitable for Stanford Dependencies (??). The Stanford Dependencies representation was also used to evaluate dependency parsers in the 2012 shared task on parsing the web (?).

7 Recent changes

This section summarizes recent changes. This may help if you see old versions of the dependencies, or need to update your code.

abbrev was removed as a relation. It is now a particular case of an *appos*.

attr has been removed as a relation. *attr* was a relation intended for the complement of a copular verb such as "to be", "to seem", "to appear". Mainly, it was used for WHNP complements. (The relation *attr* was meant to stand for "attributive" but that was arguably a misuse of the word.)

complm was removed as a relation. It is now a particular case of *mark*. This follows HPSG-like usage, where the complementizer is a mark on the clause.

discourse was introduced. The lack of a dependency type for interjections was an omission even in the early versions, but it became more essential as we expanded our consideration of informal text types.

goeswith was introduced. It is useful on badly edited text.

infmod was remode as a relation. It has been generalized as a case of *vmod*.

partmod was remode as a relation. It has been generalized as a case of *vmod*.

purpcl was removed as a relation. It is now a particular case of an *advcl*.

rel has been removed as a relation. *rel* was the relation between the main verb of a relative clause and the head of the *Wh*-phrase. Now, the converter resolves the grammatical relation (*nsubj*, *dobj*, or *pobj*) for simple cases, and the rest are left unresolved as a *dep* relation.

vmod has been introduced as a relation generalizing over non-finite verbal modifiers that are participial in form (formerly *partmod*) or infinitival (formerly *infmod*).