# Is the Stanford Dependency Representation Semantic?

**Rachel Rudinger**[1] and **Benjamin Van Durme**[1,2]
Center for Language and Speech Processing[1]
Human Language Technology Center of Excellence[2]
Johns Hopkins University
rudinger@jhu.edu, vandurme@cs.jhu.edu

## Abstract

The Stanford Dependencies are a deep syntactic representation that are widely used for semantic tasks, like Recognizing Textual Entailment. But do they capture all of the semantic information a meaning representation ought to convey? This paper explores this question by investigating the feasibility of mapping Stanford dependency parses to Hobbsian Logical Form, a practical, event-theoretic semantic representation, using only a set of deterministic rules. Although we find that such a mapping is possible in a large number of cases, we also find cases for which such a mapping seems to require information beyond what the Stanford Dependencies encode. These cases shed light on the kinds of semantic information that are and are not present in the Stanford Dependencies.

## 1 Introduction

The Stanford dependency parser (De Marneffe et al., 2006) provides "deep" syntactic analysis of natural language by layering a set of hand-written post-processing rules on top of Stanford's statistical constituency parser (Klein and Manning, 2003). Stanford dependency parses are commonly used as a semantic representation in natural language understanding and inference systems.[1] For example, they have been used as a basic meaning representation for the Recognizing Textual Entailment task proposed by Dagan et al. (2005), such as by Haghighi et al. (2005) or MacCartney (2009) and in other inference systems (Chambers et al., 2007; MacCartney, 2009).

Because of their popular use as a semantic representation, it is important to ask whether the Stanford Dependencies do, in fact, encode the kind of information that ought to be present in a versatile semantic form. This paper explores this question by attempting to map the Stanford Dependencies into Hobbsian Logical Form (henceforth, HLF), a neo-Davidsonian semantic representation designed for practical use (Hobbs, 1985). Our approach is to layer a set of hand-written rules on top of the Stanford Dependencies to further transform the representation into HLFs. This approach is a natural extension of the Stanford Dependencies which are, themselves, derived from manually engineered post-processing routines.

The aim of this paper is neither to demonstrate the semantic completeness of the Stanford Dependencies, nor to exhaustively enumerate their semantic deficiencies. Indeed, to do so would be to presuppose HLF as an entirely complete semantic representation, or, a perfect semantic standard against which to compare the Stanford Dependencies. We make no such claim. Rather, our intent is to provide a qualitative discussion of the Stanford Dependencies as a semantic resource through the lens of this HLF mapping task. It is only necessary that HLF capture some subset of important semantic phenomena to make this exercise meaningful.

Our results indicate that in a number of cases, it is, in fact, possible to directly derive HLFs from Stanford dependency parses. At the same time, however, we also find difficult-to-map phenomena that reveal inherent limitations of the dependencies as a meaning representation.

## 2 Background

This section provides a brief overview of the HLF and Stanford dependency formalisms.

### 2.1 Hobbsian Logical Form

The key insight of event-theoretic semantic representations is the *reification* of events (Davidson, 1967), or, treating events as entities in the world. As a logical, first-order representation, Hobbsian

[1]Statement presented by Chris Manning at the *SEM 2013 Panel on Language Understanding http://nlpers.blogspot.com/2013/07/the-sem-2013-panel-on-language.html.



*Proceedings of the 2nd Workshop on EVENTS: Definition, Detection, Coreference, and Representation*, pages 54–58,
Baltimore, Maryland, USA, June 22-27, 2014. 

Logical Form (Hobbs, 1985) employs this approach by allowing for the reification of *any* predicate into an event variable. Specifically, for any predicate $p(x_1, \cdots, x_n)$, there is a corresponding predicate, $p'(E, x_1, \cdots, x_n)$, where $E$ refers to the predicate (or event) $p(x_1, \cdots, x_n)$. The reified predicates are related to their non-reified forms with the following axiom schema:

$$(\forall x_1 \cdots x_n) p(x_1 \cdots x_n) \quad \leftrightarrow \quad (\exists e) Exist(e) \wedge p'(e, x_1 \cdots x_n)$$

In HLF, "A boy runs" would be represented as:

$$(\exists e, x) Exist(e) \wedge run'(e, x) \wedge boy(x)$$

and the sentence "A boy wants to build a boat quickly" (Hobbs, 1985) would be represented as:

$$(\exists e_1, e_2, e_3, x, y) Exist(e_1) \wedge want'(e_1, x, e_2) \wedge quick'(e_2, e_3) \wedge build'(e_3, x, y) \wedge boy(x) \wedge boat(y)$$

### 2.2 Stanford Dependencies

A Stanford dependency parse is a set of triples consisting of two tokens (a *governor* and a *dependent*), and a labeled syntactic or semantic relation between the two tokens. Parses can be rendered as labeled, directed graphs, as in Figure 1. Note that this paper assumes the *collapsed* version of the Stanford Dependencies.[2]

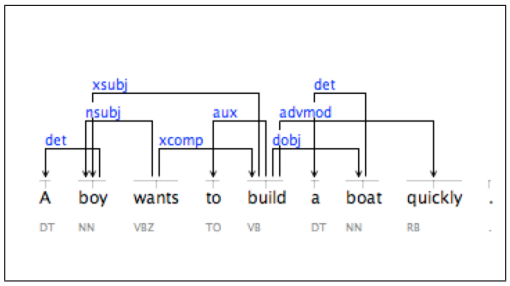


Figure 1: Dependency parse of "A boy wants to build a boat quickly."

## 3 Mapping to HLF

We describe in this section our deterministic algorithm for mapping Stanford dependency parses to HLF. The algorithm proceeds in four stages: *event extraction*, *argument identification*, *predicate-argument assignment*, and *formula construction*. We demonstrate these steps on the above example sentence "A boy wants to build a boat quickly."[3] The rule-based algorithm operates on the sentence level and is purely a function of the dependency parse or other trivially extractible information, such as capitalization.

### 3.1 Event Extraction

The first step is to identify the set of event predicates that will appear in the final HLF and assign an event variable to each. Most predicates are generated by a single token in the sentence (e.g., the main verb). For each token $t$ in the sentence, an event $(e_i, p_t)$ (where $e_i$ is the event variable and $p_t$ is the predicate) is added to the set of events if any of the following conditions are met:

1. $t$ is the dependent of the relation $root$, $ccomp$, $xcomp$, $advcl$, $advmod$, or $partmod$.
2. $t$ is the governor of the relation $nsubj$, $dobj$, $ccomp$, $xcomp$, $xsubj$, $advcl$, $nsubjpass$, or $agent$.

Furthermore, an event $(e_i, p_r)$ is added for any triple $(rel, gov, dep)$ where $rel$ is prefixed with "prep_" (e.g., $prep\_to$, $prep\_from$, $prep\_by$, etc.).

Applying this step to our example sentence "A boy wants to build a boat quickly." yields the following set:

$$(e_1, wants), (e_2, quickly), (e_3, build)$$

### 3.2 Argument Identification

Next, the set of entities that will serve as predicate arguments are identified. Crucially, this set will include some event variables generated in the previous step. For each token, $t$, an argument $(x_i, t)$ is added to the set of arguments if one of the following conditions is met:

1. $t$ is the dependent of the relation $nsubj$, $xsubj$, $dobj$, $ccomp$, $xcomp$, $nsubjpass$, $agent$, or $iobj$.
2. $t$ is the governor of the relation $advcl$, $advmod$, or $partmod$.

[2] The collapsed version is more convenient for our purposes, but using the uncollapsed version would not significantly affect our results.

[3] Hobbs (1985) uses the example sentence "A boy wanted to build a boat quickly."

Applying this step to our example sentence, we get the following argument set:

$$(x_1, boat), (x_2, build), (x_3, boy)$$

Notice that the token $build$ has generated both an event predicate and an argument. This is because in our final HLF, $build$ will be both an event predicate that takes the arguments $boy$ and $boat$, as well as an argument to the intensional predicate $want$.

### 3.3 Predicate-Argument Assignment

In this stage, arguments are assigned to each predicate. $p_t.arg_i$ denotes the $i^{th}$ argument of predicate $p_t$ and $arg(t)$ denotes the argument associated with token $t$. For example, $arg(boy) = x_2$ and $arg(quickly) = e_3$. We also say that if the token $t_1$ governs $t_2$ by some relation, e.g. $nsubj$, then $t_1$ $nsubj$-governs $t_2$, or $t_2$ $nsubj$-depends on $t_1$. Note that $arg_i$ refers to any slot past $arg_2$. Arguments are assigned as follows.

For each predicate $p_t$ (corresponding to token $t$):

1. If there is a token $t'$ such that $t$ $nsubj$-, $xsubj$-, or $agent$-governs $t'$, then $p_t.arg_1 = arg(t')$.
2. If there is a token $t'$ such that $t$ $dobj$-governs $t'$, then $p_t.arg_2 = arg(t')$.
3. If there is a token $t'$ such that $t$ $nsubjpass$-governs $t'$, then $p_t.arg_i = arg(t')$.
4. If there is a token $t'$ such that $t$ $partmod$-depends on $t'$, then $p_t.arg_2 = arg(t')$.
5. If there is a token $t'$ such that $t$ $iobj$-governs $t'$, then $p_t.arg_i = arg(t')$.
6. If there is a token $t'$ such that $t$ $ccomp$- or $xcomp$-governs $t'$, then $p_t.arg_i = arg(t')$
   (a) UNLESS there is a token $t''$ such that $t'$ $advmod$-governs $t''$, in which case $p_t.arg_i = arg(t'')$.
7. If there is a token $t'$ such that $t$ $advmod$- or $advcl$-depends on $t'$, then $p_t.arg_i = arg(t')$.

And for each $p_r$ generated from relation $(rel, gov, dep)$ (i.e. all of the "prep_" relations):

1. $p_r.arg_1 = arg(gov)$
2. $p_r.arg_i = arg(dep)$

After running this stage on our example sentence, the predicate-argument assignments are as follows:

$$wants(x_3, e_2), build(x_3, x_1), quickly(e_3)$$

Each predicate can be directly replaced with its reified forms (i.e., $p'$):

$$wants'(e_1, x_3, e_2), build'(e_3, x_3, x_1), \\ quickly'(e_2, e_3)$$

Two kinds of non-eventive predicates still need to be formed. First, every entity $(x_i, t)$ that is neither a reified event nor a proper noun, e.g., $(x_3, boy)$, generates a predicate of the form $t(x_i)$. Second, we generate Hobbs's $Exist$ predicate, which identifies which event actually occurs in the "real world." This is simply the event generated by the dependent of the $root$ relation.

### 3.4 Formula Construction

In this stage, the final HLF is pieced together. We join all of the predicates formed above with the *and* conjunction, and existentially quantify over every variable found therein. For our example sentence, the resulting HLF is:

*A boy wants to build a boat quickly.*
$(\exists e_1, e_2, e_3, x_1, x_3)[Exist(e_1) \wedge boat(x_1) \wedge boy(x_3) \wedge wants'(e_1, x_3, e_2) \wedge build'(e_3, x_3, x_1) \wedge quickly'(e_2, e_3)]$

## 4 Analysis of Results

This section discusses semantic phenomena that our mapping does and does not capture, providing a lens for assessing the usefulness of the Stanford Dependencies as a semantic resource.

### 4.1 Successes

Formulas 1-7 are correct HLFs that our mapping rules successfully generate. They illustrate the diversity of semantic information that is easily recoverable from Stanford dependency parses.

Formulas 1-2 show successful parses in simple transitive sentences with active/passive alternations, and Formula 3 demonstrates success in parsing ditransitives. Also easily recovered from the dependency structures are semantic parses of sentences with adverbs (Formula 4) and reporting verbs (Formula 5). Lest it appear that these phenomena may only be handled in isolation, Equations 6-7 show successful parses for sentences

with arbitrary combinations of the above phenomena.

***A boy builds a boat.***
$(\exists e_1, x_1, x_2)[Exist(e_1) \wedge boy(x_2) \wedge boat(x_1) \wedge builds'(e_1, x_2, x_1)]$ (1)

***A boat was built by a boy.***
$(\exists e_1, x_1, x_2)[Exist(e_1) \wedge boat(x_2) \wedge boy(x_1) \wedge built'(e_1, x_1, x_2)]$ (2)

***John gave Mary a boat.***
$(\exists e_1, x_1)[Exist(e_1) \wedge boat(x_1) \wedge gave'(e_1, John, x_1, Mary)]$ (3)

***John built a boat quickly.***
OR ***John quickly built a boat.***
$(\exists e_1, e_2, x_1)[Exist(e_1) \wedge boat(x_1) \wedge quickly(e_2, e_1) \wedge built'(e_1, John, x_1)]$ (4)

***John told Mary that a boy built a boat.***
$(\exists e_1, e_2, x_1, x_4)[Exist(e_1) \wedge boy(x_1) \wedge boat(x_4) \wedge built'(e_2, x_1, x_4) \wedge told'(e_1, John, Mary, e_2)]$ (5)

***John told Mary that Sue told Joe that Adam loves Eve.***
$(\exists e_1, e_2, e_3)[Exist(e_1) \wedge told'(e_2, Sue, Joe, e_3) \wedge loves'(e_3, Adam, Eve) \wedge told'(e_1, John, Mary, e_2)]$ (6)

***John was told by Mary that Sue wants Joe to build a boat quickly.***
$(\exists e_1, e_2, e_3, e_4, x_7)[Exist(e_1) \wedge boat(x_7) \wedge build'(e_2, Joe, x_7) \wedge told'(e_1, Mary, John, e_4) \wedge wants'(e_4, Sue, e_3) \wedge quickly'(e_3, e_2)]$ (7)

### 4.2 Limitations

Though our mapping rules enable us to directly extract deep semantic information directly from the Stanford dependency parses in the above cases, there are a number of difficulties with this approach that shed light on inherent limitations of the Stanford Dependencies as a semantic resource.

A major such limitation arises in cases of event nominalizations. Because dependency parses are syntax-based, their structures do not distinguish between eventive noun phrases like "the bombing of the city" and non-eventive ones like "the mother of the child"; such a distinction, however, would be found in the corresponding HLFs.

Certain syntactic alternations also prove problematic. For example, the dependency structure does not recognize that "window" takes the same semantic role in the sentences "John broke the mirror." and "The mirror broke." The use of additional semantic resources, like PropBank (Palmer et al., 2005), would be necessary to determine this.

Prepositional phrases present another problem for our mapping task, as the Stanford dependencies will typically not distinguish between PPs indicating arguments and adjuncts. For example, "Mary stuffed envelopes with coupons" and "Mary stuffed envelopes with John" have identical dependency structures, yet "coupons" and "John" are (hopefully for John) taking on different semantic roles. This is, in fact, a prime example of how Stanford dependency parses may resolve syntactic ambiguity without resolving semantic ambiguity.

Of course, one might manage more HLF coverage by adding more rules to our system, but the limitations discussed here are fundamental. If two sentences have different semantic interpretations but identical dependency structures, then there can *be* no deterministic mapping rule (based on dependency structure alone) that yields this distinction.

## 5 Conclusion

We have presented here our attempt to map the Stanford Dependencies to HLF via a second layer of hand-written rules. That our mapping rules, which are purely a function of dependency structure, succeed in producing correct HLFs in some cases is good evidence that the Stanford Dependencies do contain some practical level of semantic information. Nevertheless, we were also able to quickly identify aspects of meaning that the Stanford Dependencies did not capture.

Our argument does not require that HLF be an optimal representation, only that it capture worthwhile aspects of semantics and that it not be readily derived from the Stanford representation. This is enough to conclude that the Stanford Dependencies are not complete as a meaning representation. While not surprising (as they are intended as a syntactic representation), we hope this short study will help further discussion on what the community wants or needs in a meaning representation: what gaps are acceptable, if any, and whether a more "complete" representation is needed.

## Acknowledgments

This material is partially based on research sponsored by the NSF under grant IIS-1249516 and DARPA under agreement number FA8750-13-2-0017 (the DEFT program).

## References


Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D Manning. 2007. Learning alignments and leveraging natural logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 165–170. Association for Computational Linguistics.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*.

Donald Davidson. 1967. The logical form of action sentences. In *The Logic of Decision and Action*, pages 81–120. Univ. of Pittsburgh Press.

Marie-Catherine De Marneffe, Bill MacCartney, and Christopher D Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Aria D Haghighi, Andrew Y Ng, and Christopher D Manning. 2005. Robust textual inference via graph matching. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 387–394. Association for Computational Linguistics.

Jerry R Hobbs. 1985. Ontological promiscuity. In *Proceedings of the 23rd annual meeting on Association for Computational Linguistics*, pages 60–69. Association for Computational Linguistics.

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

Bill MacCartney. 2009. *Natural language inference*. Ph.D. thesis, Stanford University.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.