Linguistic **I**ssues in **L**anguage **T**echnology – **LiLT**
Submitted, January 2012

Using hybrid logic for querying dependency treebanks

Anders Søgaard
Søren Lind Kristiansen

Published by CSLI Publications

LiLT volume 7, issue 5 January 2012

Using hybrid logic for querying dependency treebanks

Anders Søgaard, *University of Copenhagen,* Søren Lind Kristiansen, *University of Copenhagen*

1 Introduction

Existing logic-based querying tools for dependency treebanks use first order logic or monadic second order logic. We introduce a very fast model checker based on hybrid logic with operators $\downarrow$, @ and A and show that it is much faster than an existing querying tool for dependency treebanks based on first order logic, and much faster than an existing general purpose hybrid logic model checker. The querying tool is made publicly available.

1.1 Querying dependency treebanks

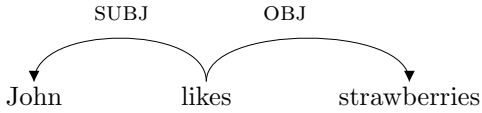
Dependency treebanks are collections of natural language sentences annotated with dependency trees. A dependency tree provides an analysis of a sentence in terms of grammatical functions, e.g. what is the subject of the main verb in the sentence. Dependency treebanks are used to train data-driven dependency parsers that are used in text mining, sentiment analysis, summarization, machine translation, etc.

More formally, a dependency tree for a sentence $x = w_1, \ldots, w_n$ is a tree $T = \langle \{0, 1, \ldots, n\}, A \rangle$ with $A \subseteq V \times V$ the set of dependency arcs. Each vertex corresponds to a word in the sentence, except 0 which is the root vertex, i.e. for any $i \leq n$ $\langle i, 0 \rangle \notin A$. Since a dependency tree is a tree it is acyclic. A tree is projective if every vertex has a

LiLT Volume 7, Issue 5, January 2012.
Using hybrid logic for querying dependency treebanks.

continuous projection, i.e. if and only if for every arc $\langle i, j\rangle \in A$ and node $k \in V$, if $i < k < j$ or $j < k < i$ then there is a subset of arcs $\{\langle i, i_1\rangle, \langle i_1, i_2\rangle, \ldots, \langle i_{k-1}, i_k\rangle\} \in A$ such that $i_k = k$. In dependency treebanks, the arcs are typically labeled with grammatical functions. Later on, we will also think of the linear order of the words that label the nodes of the graph as an unlabeled precedence relation.

Consider the following example:



The dependency structure tells us that the word 'John' is the subject (SUBJ) of the verb 'likes', and that 'strawberries' is its object (OBJ).

When linguists query collections of dependency trees, they typically want to find examples of complex syntactic phenomena, or they want to find the possible syntactic contexts of a particular phrase. Query languages for dependency treebanks are also used to validate treebanks. Consequently, a query language should be expressive enough to check for wellformedness and whether abstract linguistic principles are satisfied by all dependency trees. Finally, query languages may be applied to the output of dependency parsers in error analysis.

We list three prototypical queries for further reference:

(1) Are there any sentences in which the object comes before the subject which in turn comes before the verb, i.e. sentences in which the object is topicalized, e.g. 'Strawberries John likes'?

(2) Are there any sentences without subjects?

(3) Are there any cyclic dependency structures?

1.2 Logics for dependency trees

Propositional dynamic logic (PDL) seems to be the natural query language for trees without linear yields (Blackburn and Meyer-Viol, 1994), but natural queries in trees with linear yields seem to require nominals or intersection (Keller, 1993; Blackburn, 1994; Kracht, 1995). Consider also the query in (3) above. Cyclicity is not definable in PDL, but requires nominals or intersection.

Existing logic-based querying tools for dependency treebanks use first order logic or monadic second order logic (Kepser, 2004), but these logics have PSPACE-hard model checking problems. Since dependency treebanks may be very big (> 1M words) and queries arbitrarily complex, linear time model checking seems necessary. First order logic has also been argued to be inadequate, since it cannot define transitive clo-

sures (Kepser, 2004). Monadic second order logic, on the other hand, seems like shooting birds with cannons.

Hybrid logic with operators $\downarrow, @$ and A is in our view a more natural choice. It has linear time model checking (Franceschet and de Rijke, 2005; Lange, 2006; Lange, 2009), easily defines cyclicity and can distinguish directed acyclic graphs from their unraveled trees (Blackburn et al., 2001). The logic is equivalent to PDL with nominals (Passy and Tinchev, 1991), and we add regular expressions over propositional variables and Kleene stars over modalities as syntactic sugar.

1.3 This work

This work describes an implementation of dependency treebank querying using hybrid logic with operators $\downarrow, @$ and A, henceforth HL-($\downarrow, @, A$). The implementation is made publicly available with an online interface and evaluated in a series of experiments. The paper is organized as follows:

Sect. 2 presents related work about logic-based querying of dependency treebanks and related work on applying hybrid logic to verification or querying problems. No one has to the best of our knowledge used hybrid logics for querying dependency treebanks.

Sect. 3 introduces HL-($\downarrow, @, A$) and discusses its computational properties, incl. the complexity of model checking. We also show that queries such as (1–3) are easily expressed in HL-($\downarrow, @, A$).

Sect. 4 describes our implementation of dependency treebank querying using HL-($\downarrow, @, A$) formulas made publicly available with an online interface.

Sect. 5 presents a series of experiments that compares the runtime of our system to two baseline systems, a state-of-the-art dependency treebank querying software (DTAG; Buch-Kromann, 2003) and a state-of-the-art hybrid logic model-checker (HLMC; Franceschet and de Rijke, 2005). We evaluate the three systems on two sets of one-operator formulas of depth $k \in \{3, 5, 10, 25, 50\}$, e.g. $A(A(A(A(A(\phi)))))$ and on formulas used to check acylicity of dependency trees. We use two dependency treebanks in our experiments: Penn-III (Marcus et al., 1993) for English and Talbanken05 (Nilsson et al., 2005) for Swedish. Penn-III is one of the largest available treebanks with 50,000 sentences ($\sim$ 1M words), whereas Talbanken04 is rather small ($\sim$ 200K words).

2 Related work

2.1 Treebank query languages

Common query tools for treebanks include CorpusSearch, ICECUP III Wallis and Nelson (2000) and TGrep2, but as pointed out in Kepser

(2004) the query languages used in these tools are not even expressive enough to perform arbitrary queries on context-free derivations. They are, according to Kepser (2004), all subsumed by the existential fragment of first order logic. Other more expressive logics that have been introduced to characterize context-sensitive grammar formalisms Reape (1994), Richter (2004) have model checking procedures with exponential runtime. The logics proposed in Reape (1994), Richter (2004) are both extensions of HL-$(\downarrow, @, A)$. See Søgaard and Lange (2009) for a polyadic dynamic logic specifically designed for linguistic theories in which grammatical structures may contain set valued attributes, with PSPACE-hard model checking, but an expressive linear time fragment.

DTAG Buch-Kromann (2003) is one of the most promising tools for querying dependency treebanks. It uses first order logic with binary relations for encoding user queries. It reads dependency treebanks in a variety of formats and has a number of interesting features, incl. graphical visualization of example sentences.

2.2 Querying with hybrid logic

The key concepts in hybrid logic date back to Arthur Prior, but were revived in the mid-1990s (Blackburn and Seligman, 1995; Blackburn et al., 2001). Most applications of this logic have been rather philosophical, and most theoretical work has been devoted to satisfiability and proof systems. Only recently researchers began to look into model checking with hybrid logic (Franceschet and de Rijke, 2005; Lange, 2006; Lange, 2009). Since model checking with hybrid logic is a new area of interest, there has to the best of our knowledge been almost no work on applying model checking with hybrid logic to practical problems, with the exception of Hoareau and Satoh (2007, 2008).

Hoareau and Satoh (2007, 2008) use model checking with hybrid logic in so-called location-aware software. In location-aware software, physical space is organized in trees where nodes represent relevant places. Queries can, for example, be:

(4)Where is the surgery?

(5)Where is the closest computer to the elevator?

(6)How to reach the roof?

Hybrid logics are used to name identified places. Since the hierarchical space graph forms a tree, there is no need to query for cycles. Hoareau and Satoh (2007, 2008) implement a model checker for their particular purposes in much the same way as we have done. They do not, however, compare the efficiency of their system to state-of-the-art systems in location-aware software.

3 Hybrid logic with operators $\downarrow$, @ and A

The key intuition behind our use of hybrid logic for querying dependency treebanks is that dependency trees can be thought of as Kripke models. In fact they are a subclass of the set of binary multimodal Kripke models (Blackburn et al., 2001). Sentence positions are nominals that denote the nodes in dependency structure. The words are propositional variables that may denote several nodes in a tree. Typically nodes are also labeled by part-of-speech tags, i.e. syntactic categories such as NNP (subclass of nouns) or VBN (subclass of verbs), which also can be seen as propositional variables that denote a subset of the nodes of a dependency structure. The dependency arcs are labeled by modalities.

The syntax of HL-$(\downarrow, @, A)$ is as follows:

$$\phi \quad \doteq \quad p|i|\phi \wedge \psi|\neg\phi|\langle a\rangle\phi|\downarrow_i \phi|@_i\phi|A\phi$$

with $p \in \mathbb{PROP}$ a propositional variable, $i \in \mathbb{NOM}$ a nominal and $a \in \mathbb{LBL}$ a dependency label. The letters $i, j, k, \ldots$ will be used to denote nominals, i.e. propositional variables that can only be true in a single node in a dependency tree, or equivalently in a single world in the Kripke model.

We augment this language in two ways: Since many of our propositional variables are part-of-speech-tags with some internal structure, e.g. NNP and NN both denote nominal elements, it is useful to be able to query with regular expressions such as '$\langle a\rangle(\mathrm{NN}^*)$'. Similarly we add a PDL-style Kleene star over modalities (programs) to be able to ask for dominance along dependency arcs with a certain label or precedence in a syntactically simple fashion. None of these augmentations have any impact on the complexity of model checking.

The semantics of HL-$(\downarrow, @, A)$ is, except for the $\downarrow$ operator, a straight-forward extension of modal logic semantics with valuation function $\mathcal{V} : \mathbb{PROP} \cup \mathbb{NOM} \to \mathbb{W}^2$.

$$\begin{array}{rll} M, w \models p & \text{iff} & w \in \mathcal{V}(p) \\ M, w \models i & \text{iff} & \mathcal{V}(i) = \{w\} \\ M, w \models \phi \wedge \psi & \text{iff} & M, w \models \phi \& M, w \models \psi \\ M, w \models \neg\phi & \text{iff} & M, w \not\models \phi \\ M, w \models \langle a\rangle\phi & \text{iff} & \exists w'.R_a(w, w') \& M, w' \models \phi \\ M, w \models @_i\phi & \text{iff} & M, w' \models \phi \wedge \mathcal{V}(i) = \{w'\} \\ M, w \models A\phi & \text{iff} & \forall w' \in \mathbb{W}.M, w' \models \phi \end{array}$$

For the $\downarrow$ operator, we introduce a valuation function $g : \mathbb{NOM} \to \mathbb{W}$ and write:

$$M, g, w \models \downarrow_i \phi \quad \text{iff} \quad M, g', w \models \phi \wedge g' =_i g \wedge g'(x) = w$$

In other words, g' assigns i to w, but otherwise agree with g in all respects. Intuitively, the $\downarrow$ operator thus names a world. The @ operator takes you to named worlds, whereas A takes you everywhere in a connected graphs. In dependency treebanks, the nodes are words labeled with word forms and POS tags, and the operators are thus used to move around between the words in the dependency structures.

Example. Consider the following hybrid logic formulas, all true of the second node in our running example, i.e. the word decorated with the verb *likes*:

1. $\langle \text{OBJ} \rangle \top$
2. $\langle \text{OBJ} \rangle \downarrow_i \top \wedge \langle \text{RIGHT} \rangle i$
3. $\downarrow_j \langle \text{OBJ} \rangle \downarrow_i @_i \langle \text{LEFT} \rangle j$

The first formula is true in the second node, because the verb *likes* has an object, namely the third word in the sentence. In other words $(w_2, w_3) \in R_{\text{OBJ}}$. The second formula says that *likes* has an object that is immediately right of it. The nominal enforces the identity. Finally, the third formula says that if you go the object of *likes*, there is a left arc back to *likes*. Or, equivalently, *likes* is immediately left of its object.

Let us introduce a R_{HEAD} such that $(i \wedge \langle \text{OBJ} \rangle j) \rightarrow (j \wedge \langle \text{HEAD} \rangle i)$; and similarly, for all other dependency relations. R_{HEAD} is now the mirror image or converse of the union of all the dependency relations. It now holds that $A\langle \text{HEAD}^* \rangle root$, where *root* is the name of the root node. This imposes connectivity on dependency structures.

In our experiments, we will query for cyclic structures, but we may now reconsider the question in (2) above, i.e. whether there are any sentences without subjects? The shortest formula encoding that is probably $A[\text{SUBJ}]\bot$. If this formula comes out true in a dependency structure, there are no subjects in that dependency structure.

4 Querying dependency treebanks in hybrid logic

Dependency treebanks typically come in what is known as CoNLL format. The format uses eight columns to encode dependency structures with one row per word. The English and Swedish treebanks used in our experiments only make use of five of these columns. Our running example is presented in CoNLL format in Figure 1. The CONLL format is described in more detail in Buchholz and Marsi (2006).

When querying dependency structures in CONLL format we think

	WORD	LEMMA	CPOS	POS	FEATS	HEAD	DEPREL
1	John	-	NP	-	-	2	SUBJ
2	likes	-	V	-	-	0	ROOT
3	strawberries	-	NP	-	-	2	OBJ

FIGURE 1 Running example in CoNLL format.

of each structure as a Kripke structure. A Kripke structure is thus represented as n rows of eight columns. As already mentioned, the dependency relations form a tree, but incl. precedence relations the dependency graphs may contain cycles.

Contrary to HLMC our system uses a top-down approach, checking each world one by one. This means we can do lazy evaluation. For example, for a disjunction like $\phi \vee \psi$ we need not test ψ if ϕ turns out to be true. Similar lazy evaluation can be done for conjunction and the conditional. This approach is also suggested in Lange (2009).

This does not work well with the A operator, since we would easily end up checking the same world over and over again. We have therefore implemented a mechanism that keeps track of what subformulas have been checked, including the world they were checked in and the variables bound at the time. This allows us to skip some rechecking when we have nested As. Furthermore we have implemented a preprocessing step that can collapse A operators if subexpressions turn out to have constant value. This preprocessing step reduces all the A-formulas in the test to the same size. Of course, in more realistic settings, the gain from the preprocessing step will be much smaller.

We have added a few operators to the grammar designed specifically for querying dependency treebanks in CONLL format. These include unary operators WORD ϕ and POS ϕ for checking for specific word forms and POS tags, resp. We have also added WORDRE ϕ and POSRE ϕ that work the same way except that the check is carried out using regular expressions. Our system is written in C#.

5 Experiments

5.1 Data

We use two dependency treebanks in our experiments: Penn-III (Marcus et al., 1993) for English and Talbanken05 (Nilsson et al., 2005) for Swedish. Penn-III is one of the largest available treebanks with 50,000 sentences ($\sim$ 1M words), whereas Talbanken04 is rather small ($\sim$ 200K words). Penn-III does not have any non-projective dependency arcs, but 9.8% of the dependency arcs in the Swedish treebank are non-projective.

5.2 Baseline system

Our two baseline systems are publicly available, namely DTAG[1] and HLMC[2].

HLMC is to the best of our knowledge the only publicly available hybrid logic model checker. It uses XML-formatted Kripke models as input format, and we implemented an efficient wrapper to feed it with queries and XML-formatted dependency trees. HLMC implements the algorithms in Franceschet and de Rijke (2005). Contrary to HLMC our system uses a top-down approach very similar to the approach taken in Lange (2009), checking each world one by one. HLMC also implements more hybrid logic operators.

DTAG (Buch-Kromann, 2003) is a widely used tool for constructing and querying dependency treebanks. DTAG reads formulas from stdin. Consequently, we only test formulas of limited depth ($k \leq 5$). This, however, is enough to illustrate that our model checker is considerably faster than DTAG.

5.3 Experiments

Test formulas

The test formulas used in HLMC and our system are identical, since our model checker supports the syntax used in HLMC. The formulas were generated automatically using the algorithms presented in Figure 2–4:

- The algorithm in Figure 2 generates formulas that checks for cycles of depth k. This kind of formulas is motivated by our application. Checking for cycles is important when validating a treebank. Our model checker implements transitive closure over relations (PDL-style Kripke star over modalities), but since HLMC does not, we do not use transitive closure in our formulas. The algorithm, for example, generates the following formula for $k = 2$:

  $$\downarrow i_0(\top \rightarrow (\neg\langle a_0\rangle(i_0) \wedge \neg\langle a_0\rangle(\langle a_0\rangle(i_0))))$$

  where R_{a_0} is the union of all dependency relations; or, more intuitively, an unlabeled dependency relation. The formula checks that we cannot get back to the current state in less than three $(k + 1)$ steps.
- The algorithm in Figure 3 generates tautologies with k embeddings of $\downarrow$ operators. These formulas are not motivated by our application, but $\downarrow$ operators are frequently embedded in linguistic queries.

[1] http://www.buch-kromann.dk/matthias/dtag/
[2] http://www.luigidragone.com/hlmc/

```
1: print "↓ i0(⊤ → ("
2: for i ∈ {1...k} do
3:    if i ≠ 1 then
4:       print "∧"
5:    end if
6:    print "¬"
7:    for j ∈ {1...i} do
8:       print ⟨a0⟩("
9:    end for
10:   print "i0"
11:   for j ∈ {1...i} do
12:      print ")"
13:   end for
14: end for
15: print "))"
```

FIGURE 2 Algorithm for generating cycle check formulas.

```
1: print "↓ i0("
2: for i ∈ {1...k − 1} do
3:    print "↓ ii(ii−1∧"
4: end for
5: print "ik−1"
6: for j ∈ {1...k − 1} do
7:    print )"
8: end for
9: print ")"
```

FIGURE 3 Algorithm for generating ↓-formulas.

- The algorithm in Figure 4 generates tautologies with k embeddings of A operators. These formulas are not motivated by our application, but A operators are frequently embedded in linguistic queries.

As already mentioned, we only test formulas with $k = 3$ and $k = 5$ in DTAG. Furthermore, DTAG is a querying tool for dependency treebanks, not a hybrid logic model checker, so we only test DTAG performance on cycle check formulas.

Hardware

Since HLMC currently does not support Mac OS X or Windows, we had to use different operating systems and different hardware in our comparison. HLMC was run under Linux on a small 2.66GHz, 16GB

```
print "A("
for i ∈ {1...k − 1} do
    print "A(⊤∧"
end for
print "⊤"
for j ∈ {1...k − 1} do
    print )"
end for
print ")"
```

FIGURE 4 Algorithm for generating A-formulas.

PTB-III					
	3	5	10	25	50
DTAG	17:28	63:20	-	-	-
HLMC	01:18	02.03	05:56	10K:	∞
Ours	00:02	00:04	00:08	00:25	00:41
T05					
	3	5	10	25	50
DTAG	2:50	11:03	-	-	-
HLMC	00:14	00:21	00:51	2.3K:	∞
Ours	00:00	00:01	00:01	00:03	00:06

FIGURE 5 Performance on cycle check formulas, excl. I/O and parsing the input. Very high numbers estimated after partial runs.

RAM server with eight cores. We deliberately used *weaker* hardware for our own system, namely a 2.66GHz Intel Core i7 with Mac OS X (4GB RAM). Furthermore, the model checker was run virtually in Windows 7 (32 bit) using only 1 CPU and 2GB RAM. DTAG was also run on considerably stronger hardware, namely a 3.06 GHz Intel Core 2 Duo with Mac OS X (8 GB RAM). Consequently, if our system is faster than HLMC and DTAG in this set-up, it will be even faster on a comparable set-up.

5.4 Results

Our results show that our model checker is *much* faster than HLMC, especially for formulas of considerable depth. Even for formulas of limited depth, our system seems quite consistently to be more than 50 times faster than HLMC on cycle check formulas.

HLMC seems to have serious problems with formulas with nested $\downarrow$ operators. The runtimes for A formulas are somewhat incomparable

PTB-III					
	3	5	10	25	50
HLMC	50:37	22K:	∞	∞	∞
Ours	00:01	00:01	00:02	00:03	00:07
T05					
	3	5	10	25	50
HLMC	05:17	2.6K:	∞	∞	∞
Ours	00:00	00:00	00:00	00:00	00:01

FIGURE 6 Runtimes on $\downarrow$-formulas (mm:ss), excl. I/O and parsing the input. Very high numbers estimated after partial runs.

PTB-III					
	3	5	10	25	50
HLMC	00:43	00:44	00:54	01:44	04:37
Ours	00:01	00:01	00:01	00:01	00:01
T05					
	3	5	10	25	50
HLMC	00:08	00:08	00:09	00:18	00:47
Ours	00:00	00:00	00:00	00:00	00:00

FIGURE 7 Runtimes on A-formulas (mm:ss), excl. I/O and parsing the input. Very high numbers estimated after partial runs.

because of the strategy mentioned above that reduces A formulas to a shorter equivalent form.

The most worrying result from a practical point of view is that HLMC is very slow when it searches for long cycles. Searching for cycles with maximum length 25, the HLMC takes about 10,000 minutes, i.e. about a week, to search the entire PTB-III. Our model checker does that in 25 seconds.

6 Conclusion

We have implemented a very fast top-down hybrid logic model checker for querying dependency treebanks. It has a number of features that makes it easy to write linguistic queries and validate dependency treebanks. In a series of experiments, we show that our model checker is considerably faster than a commonly used querying tool for dependency treebanks (DTAG) and a general purpose hybrid logic model checker (HLMC). The system is publicly available for on-line use at http://www.modelchecker.dk.

Acknowledgments

This paper was presented at a meeting in Hybrid Logic and Applications, The 25th Annual IEEE Symposium on Logic in Computer Science (LICS), but never published. We thank the audience for their helpful comments. Thanks also to the three anonymous reviewers.

References

Blackburn, Patrick. 1994. Structures, languages and translations: the structural approach to feature logic. In C. J. Rupp, R. Johnson, and M. Rosner, eds., *Constraints, language and computation*, pages 1–29. London: Academic Press.

Blackburn, Patrick, Maarten de Rijke, and Yde Venema. 2001. *Modal logic*. Cambridge, England: Cambridge University Press.

Blackburn, Patrick and Wilfried Meyer-Viol. 1994. Linguistics, logic and finite trees. *Logic Journal of IGPL* 2(1):3–29.

Blackburn, Patrick and Jerry Seligman. 1995. Hybrid languages. *Journal of Logic, Language and Information* 4:251–272.

Buch-Kromann, Matthias. 2003. The Danish Dependency Treebank and the DTAG Treebank Tool. In *TLT*.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X Shared Task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. New York City, NY.

Franceschet, Massimo and Maarten de Rijke. 2005. Model checking for hybrid logics. *Journal of Applied Logic* 4(3):279–304.

Hoareau, Christian and Ichiro Satoh. 2007. A model checking-based approach for location query processing in pervasive computing environments. In *On the Move to Meaningful Internet Systems 2007*, vol. 4806 of *LNCS*, pages 866–875. Berlin, Germany: Springer.

Hoareau, Christian and Ichiro Satoh. 2008. Hybrid logics and model checking: A recipe for query processing in location-aware environments. In *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 130–137. Washington, DC, USA.

Keller, Bill. 1993. *Feature logics, infinitary descriptions and grammar*. Stanford, California: CSLI Publications.

Kepser, Stephan. 2004. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language and Information* 13:457–470.

Kracht, Marcus. 1995. Is there a genuine modal perspective on feature structures? *Linguistics & Philosophy* 18:401–458.

Lange, Martin. 2006. Model checking propositional dynamic logic with all extras. *Journal of Applied Logic* 4:39–49.

Lange, Martin. 2009. Model checking for hybrid logic. *Journal of Logic, Language and Information* 18(4):465–491.

Marcus, Mitchell, Mary Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19(2):313–330.

Nilsson, Jens, Jens Hall, and Joakim Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *NODALIDA*. Joensuu, Finland.

Passy, Solomon and Tinko Tinchev. 1991. An essay in combinatory dynamic logic. *Information and Computation* 93:262–332.

Reape, Mike. 1994. A feature value logic with intensionality, nonwellfoundedness and functional and relational dependencies. In *Constraints, language and computation*, pages 77–110. San Fransisco, CA: Academic Press.

Richter, Frank. 2004. *A mathematical formalism for linguistic theories with an application in head-driven phrase structure grammar*. Phd thesis (2000), Universität Tübingen, Tübingen, Germany.

Søgaard, Anders and Martin Lange. 2009. Polyadic dynamic logics for HPSG parsing. *Journal of Logic, Language and Information* 18(2):159–198.

Wallis, Sean and Gerald Nelson. 2000. Exploiting fuzzy tree fragment queries in the investigation of parsed corpora. *Literary and Linguistic Computing* 15(3):339–361.